Парадигма развития науки

Методологическое обеспечение

А. Е. Кононюк

ДИСКРЕТНО-НЕПРЕРЫВНАЯ МАТЕМАТИКА

Книга 10

Алгоритмы

Часть 2

Прикладная теория алгоритмов

Киев
«Освіта України»
2017

УДК 51 (075.8)
ББК В161.я7
К213

Рецензенты:

В. В. Довгай — к-т физ.-мат. наук, доц. (Национальный тех— нический университет «КПІ»);

В. В. Гавриленко — д-р физ.-мат. наук, проф.,

О. П. Будя — к-т техн. наук, доц. (Киевский университет эко— номики, туризма и права);

Н. К. Печурин — д-р техн. наук, проф. (Национальный ави— ационный университет).

Кононюк А. Е.
К213 Дискретно-непрерывная математика. (Алгоритмы). —В 12-и кн. Кн. 10,Ч.2— К.: 2017. —544 с.

ISBN 978-966-373-693-8 (многотомное издание)
ISBN 978-966-373-694-11 (книга 10, ч.2)

Многотомная работа содержит систематическое изложение математических дисциплин, исспользуемых при моделировании и исследованиях математических моделей систем.

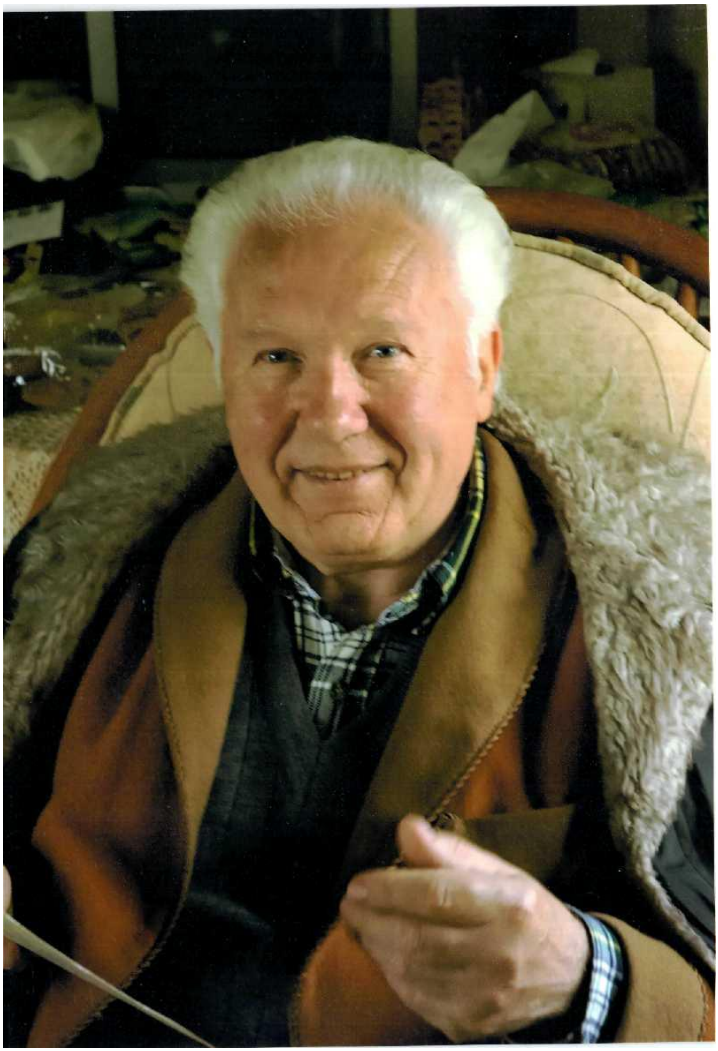
В работе излагаются основы теории множеств, отношений, поверхностей, пространств, алгебраических систем, матриц, графов, математической логики, теории вероятностей и массового обслуживания, теории формальных грамматик и автоматов, теории алгоритмов, которые в совокупности образуют единную методолгически взамосвязанную математическую систему «Дискретно-непрерывная математика».

Для бакалавров, специалистов, магистров, аспирантов, докторантов и просто ученых и специалистов всех специальностей.

УДК 51 (075.8)
ББК В161.я7

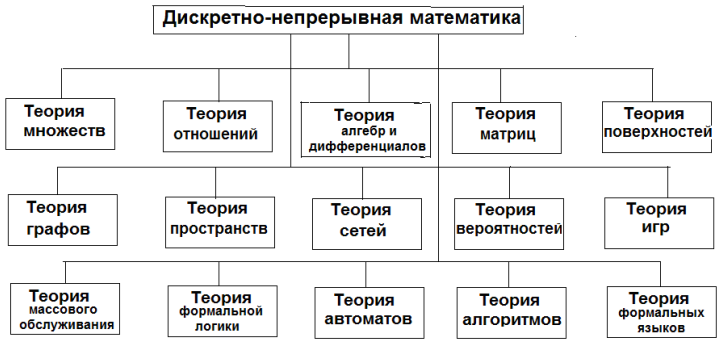
ISBN 978-966-373-693-8 (многотомное издание)
ISBN 978-966-373-694-11 (книга 10, ч. 2)

© Кононюк А. Е., 2017
© Освіта України, 2017



Кононюк Анатолий Ефимович

Структура
открытой развивающейся панмедийной системы математических наук (дисциплин)
"Дискретно-непрерывная математика"



Оглавление

Введение

Стандартные формы представления алгоритмов, подобные нормальному алгоритму Маркова, в силу их чрезвычайно высокой степени детализации непригодны для инженерной практики. Машина Тьюринга является удобной абстрактной моделью реализации любого алгоритма человеком или вычислительной машиной, но в реальных условиях любой вид памяти и время функционирования жестко ограничены. В то же время при разработке и реализации конкретных алгоритмов в инженерной практике достаточно исходить из их общих свойств.

Прикладная теория алгоритмов мало озабочена собственно существованием алгоритмов (обычно это просто подразумевается), а направляет усилия, главным образом, на разработку практически наиболее эффективных методов их описания, преобразования и реализации. Алгоритм рассматривается как совокупность определенным образом связанных между собой *операторов,* представляющих элементарные операции, которые производятся над множеством подвергающихся переработке объектов. Способы реализации операторов считаются известными (как правило, операторы сами являются некоторыми стандартными алгоритмами), а при конкретной реализации алгоритма задаются также значения исходных данных и параметров, входящих в описание операторов.

Для описания алгоритмов используются различные методы, отличающиеся степенью детализации и формализации. Теоретическое описание обычно дается в повествовательно-формульном изложении, цель которого — обосновать без излишних подробностей процедуру, предлагаемую в качестве алгоритма. Для наглядного представления структуры алгоритмов широко применяются графические средства: графы, блок-схемы, сети. Формальное и полное описание алгоритмов осуществляется на специально разработанных для этой цели *алгоритмических языках;* оно содержит всю необходимую для реализации алгоритма информацию, но не связано непосредственно со специфическими особенностями вычислительных машин. Машинная реализация алгоритма требует перевода его на язык, свойственный данной машине, в виде программы. Роль автоматических переводчиков с алгоритмических языков играют специальные программы, называемые *трансляторами.* Часто общее описание алгоритма непосредственно переводится на машинный язык путем расшифровки операторов алгоритма в операции вычислительной машины.

В отличие от абстрактной теории алгоритмов, прикладная теория рассматривает не только детерминированные, но также *эвристические* и *вероятностные* (статистические) алгоритмы. Эвристический алгоритм, кроме детерминированных или статистически заданных правил, включает также содержательные указания о целесообразном направлении процесса.

1. Эвристические методы поиска решений

Традиционными методами поиска решений считаются: методы поиска в пространстве состояний на основе различных эвристических алгоритмов, методы поиска на основе предикатов (метод резолюции и др.), поиск решений в продукционных системах, поиск решений в семантических сетях и т. д. Рассмотрим эти методы подробно.

1.1. Методы поиска решений в пространстве состояний

Методы поиска решений в пространстве состояний начнем рассматривать с простой задачи о миссионерах и людоедах. Три миссионера и три людоеда находятся на левом берегу реки и им нужно переправиться на правый берег, однако у них имеется только одна лодка, в которую могут сесть лишь 2 человека. Поэтому необходимо определить план, соблюдая который и курсируя несколько раз туда и обратно, можно переправить всех шестерых. Однако если на любом берегу реки число миссионеров будет меньше, чем число людоедов, то миссионеры будут съедены. Решения принимают миссионеры, людоеды их выполняют.

Основой метода являются следующие этапы.

1. Определяется конечное число состояний, одно из состояний принимается за начальное и одно или несколько состояний определяются как искомое (конечное, или терминальное). Обозначим состояние S тройкой S=(x,y,z), где x и y - число миссионеров и людоедов на левом берегу, z= {L,R} - положение лодки на левом (L) или правом (R) берегах. Итак, начальное состояние S_0=(3,3, L) и конечное (терминальное) состояние S_k=(0,0, R).

2. Заданы правила перехода между группами состояний. Введем понятие действия M:[u, v]w, где u - число миссионеров в лодке, v - число людоедов в лодке, w - направление движения лодки (R или L).
3. Для каждого состояния заданы определенные условия допустимости (оценки) состояний: x >= y ; 3-x >= 3-y ; u+v <= 2.
4. После этого из текущего (исходного) состояния строятся переходы в новые состояния, показанные на рис. 1. Два новых состояния следует сразу же вычеркнуть, так как они ведут к нарушению условий допустимости (миссионеры будут съедены).
5. При каждом переходе в новое состояние производится оценка на допустимость состояний и если при использовании правила перехода для текущего состояния получается недопустимое состояние, то производится возврат к тому предыдущему состоянию, из которого было достигнуто это текущее состояние. Эта процедура получила название бэктрекинг (bac tracing или BACKTRACK).

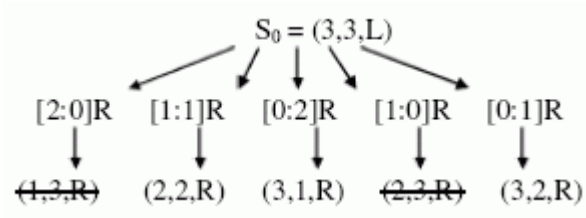


Рис. 1. Переходы из исходного состояния

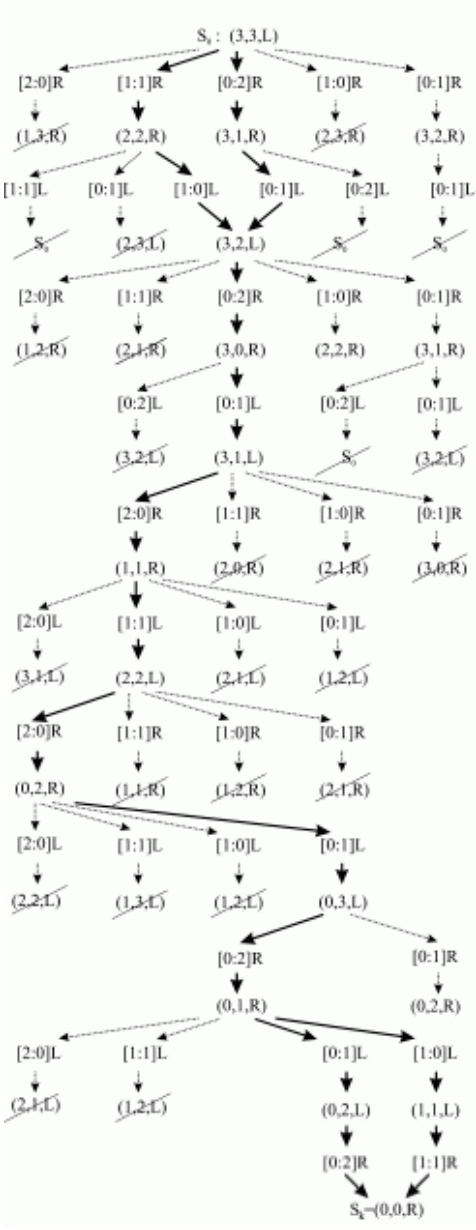
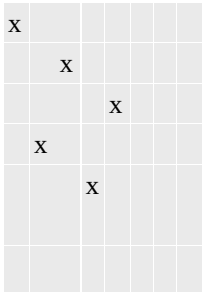


Рис. 2. Метод поиска в пространстве состояний

Теперь мы можем проанализировать полностью алгоритм простейшего поиска решений в проблемном пространстве, описанный группами состояний и переходами между состояниями на рис. 2. Решение задачи выделено на рис. 2 жирными стрелками. Такой метод поиска $S_0 => S_k$ называется прямым методом поиска. Поиск $S_k => S_0$ называют обратным поиском. Поиск в двух направлениях одновременно называют двунаправленным поиском.

Как уже упоминалось, фундаментальным понятием в методах поиска в ИС является идея рекурсии и процедура BACKTRACK. В качестве примера многоуровневого возвращения рассмотрим задачу размещения на доске 8 x 8 восьми ферзей так, чтобы они не смогли "съесть" друг друга.

Допустим, мы находимся на шаге размещения ферзя в 6 ряду и видим, что это невозможно. Процедура BACKTRACK пытается переместить ферзя в 5 строке и в 6 строке опять неудача. Только возврат к 4 строке и нахождение в ней нового варианта размещения приведет к решению задачи. Читатель сам может завершить решение этой задачи на основе процедуры BACKTRACK.



1.2. Универсальные задачи

О трудоемкости алгоритмов. В классической теории алгоритмов задача считается разрешимой, если существует решающий ее алгоритм. Однако для реализации некоторых алгоритмов при любых разумных с точки зрения физики предположениях о скорости выполнения элементарных шагов может потребоваться больше

времени, чем по современным воззрениям существует Вселенная. Поэтому возникает потребность конкретизировать понятие разрешимости и придать ему оценочный, количественный характер, т. е. ввести такие характеристики алгоритмов, которые позволяли бы судить о их фактической реализуемости и говорить о том, насколько они трудоемки, т. е. хороши или плохи при реализации. На практике ответ на этот вопрос зависит от того, на какой машине решаются задачи, как быстро нужно получать решения и какие размерности (объемы входных и выходных данных) имеют рассматриваемые варианты. Теоретические исследования касаются характеристик, имеющих смысл для любых машин— например, зависимости количества действий от размерности задачи.

Естественно, что с ростом размерности задачи количество действий (и следовательно, время решения) тоже растет. При этом можно выделить алгоритмы двух классов. Для алгоритмов первого класса время решения растет, как не слишком большая степенная функция размерности. У алгоритмов второго класса это время растет экспоненциально или еще быстрее. Алгоритмы первого класса практически пригодны для решения задач довольно большой размерности, в то время как размерность задач, решаемых алгоритмами второго класса, как правило, не больше нескольких десятков. Поэтому алгоритмы с числом действий, растущим как некоторый полином от размерности, можно считать хорошими, а алгоритмы, у которых число действий растет быстрее любого полинома от размерности — плохими. Отметим, что на практике границы между этими классами алгоритмов — не всегда четкие. Иногда предлагаются алгоритмы с временем решения, растущим, как высокая, например 10-я степень размерности. С точки зрения практики их надо отнести ко второму классу. С другой стороны, широко применяемый метод ветвей и границ относится ко второму классу, так как для него нет оценки числа действий, кроме экспоненциальной. Однако он достаточен для решения многих прикладных задач довольно большой размерности. Дело в том, что экспоненциальная оценка трудоемкости решения задач, решаемых этим методом, часто достигается лишь для специально подобранных примеров, не встречающихся на практике.

Задачи, решаемые методом перебора.

В дальнейшем размерностью задачи будем называть заранее заданное число *n,* ограничивающее длину записи аргумента *x* и ответа *y.* Таким образом, ответ задачи, если он существует, — это одно из $(t + 1)^n$ слов длины *n,* состоящих из *t* символов языка описания ответа и пустого

символа, которым дополняется ответ, если его длина меньше *n.*
Определение метода перебора можно дать следующим образом. Для задачи *A* введем предикат P_A (x, *y)* : «*y*— решение задачи *A* при данных x». Аргументы *x* и *y* — это тексты в некотором алфавите, своем для каждой задачи. Однако опыт работы с универсальными машинами Тьюринга уже показал нам, что алфавит можно зафиксировать раз и навсегда для всех задач. В дальнейшем будем считать этот алфавит двоичным; тем самым тексты *x* и *y* при любом их двоичном кодировании становятся двоичными числами.
Пусть дан способ вычисления предиката P_A *(x, y).* Тогда метод перебора состоит в том, чтобы при заданном *n* и при *x* длины $l(x) \le n$ последовательно вычислять P_A *(x, y)* для всех 2^n значений *y* от 0 до 2^n— 1. (Если существует способ отбрасывать некоторые *y,* не вычисляя для них P_A *(x, y),* говорят, что в переборе есть *отсечения.)* Как только окажется, что P_A *(x, y)* истинно, выдается положительный ответ и это значение *y.* Если же на всех допустимых *y* P_A *(x, y)* = Л, то выдается отрицательный ответ. Очевидно, что если P_A *(x, y)* вычислим, то метод перебора — это алгоритм, или прогоамма. Основным ее блоком является программа $П_A$ *(x, y, n),* которая при заданных *x, y, n* вычисляет P_A *(x, y)* .
Общий решатель — это универсальная программа, которая по размерности *n* задачи *A* ее данным *x* и программе $П_A$ *(x, y, n)* вычисления соответствующего предиката P_A *(x, y)* находит решение *y* этой задачи или убеждается, что его не существует. При этом программа $П_A$ *(x, y, n)* не обязательно может всегда вычислить P_A *(x, y);* достаточно, чтобы она делала это в случаях, когда $l(x) \le n$ и $l(y) \le n$.
Несмотря на такую глобальную цель программы, построить некоторый общий решатель не так уж трудно. Нужно только обеспечить переход от выполнения действий общего решателя, не зависящих от конкретной поставленной задачи, к выполнению программы П и обратно, а также сообщить последней, с какими аргументами *x, y* и, может быть, *n* она должна работать и куда поместить ответ. Общая схема работы такой программы может иметь, например, следующий вид.
1. Присвоить *y* начальное значение 0.
2. При помощи программы П *(x, y, n)* вычислить предикат *P (x, y).*
3. Если *P (x, y)* = 1, то значение *y* найдено— конец работы с положительным ответом, иначе перейти к п. 4.
4. Увеличить *y* на 1.
5. Если $y < 2^n$, то перейти к п. 2, иначе искомого *y* не существует — конец работы с отрицательным ответом.

Эта программа перебирает все значения от 0 до 2^n— 1 и каждое проверяет, не является ли оно решением варианта данной задачи при данных *x.* Если такое *y* найдено, выдается положительный ответ — это значение *y,* и работа программы прекращается. В противном случае перебор *y* доводится до 2^n— 1, после чего дается отрицательный с твет. Время работы такого общего решателя имеет оценку — $O(2^n \times t(П, n))$, где *t* (П, *n)* — оценка времени работы программы П с аргументами длины, не превосходящей *n.* Остается неизвестным, существует ли более эффективный способ использования информации о задаче, содержащейся в этой программе.

Задачи перебора со степенной проверкой.

Среди задач можно выделить такие, для которых можно составить программы П *(x, y, n),* вычисляющие соответствующие предикаты *P (x, y)* за степенное, или полиномиальное время. Эта краткая формулировка означает, что количество действий такой программы имеет оценку сверху $t(П(x, y, n)) \le O((l(x) + l(y))^c)$, где *c*— некоторая константа. Однако необходимо произвести уточнение.

Для любого заданного *n* существует быстро работающая программа П *(x, y, n).* Такова, например, программа выборки из таблицы значений *P (x, y)* для всех пар аргументов *(x, y)* при $l(x) \le n$ и $l(y) \le n$. Работа такой программы состоит в чтении *x* и *y,* вычислении адреса соответствующего элемента таблицы, равного $A_0 + A_1 x 2^n + y$ и выборки значения *P* (x, *y)* по этому адресу. Количество действий чтения и определения адреса пропорционально *n*, а получение ответа на машине требует одного действия. Однако такую программу нельзя считать эффективной. Время ее ввода в машину зависит от *n* показательным образом (необходимо ввести 4^n элементов таблицы).

Уточнение понятия класса задач перебора со степенной проверкой можно произвести различными способами. Здесь будет использоваться следующее определение: пусть для задачи *A* существует программа $П_A(x, y, n)$ постоянной длины *l* (П), вычисляющая $P_A(x, y)$ за степенное время (можно разрешить длине *l* (П) расти, но не быстрее, чем степенным образом). Тогда рассматриваемая задача называется задачей перебора со степенней проверкой.

Класс задач перебора со степенной проверкой достаточно широк, как показывают следующие примеры.

Задача о покрытии.

Даны множество $A = \{a_1, \ldots, a_n\}$ из n элементов и система $W = \{V_1, \ldots, V_m\}$ его подмножеств. Найти $r < m$ подмножеств этой системы $V_{i_1}, \ldots, V_{i_r}$, покрывающих все A: $\bigcup_{s=1}^{r} V_{i_s} = A$. Для этой задачи x — матрица инцидентности $\| a_{ij} \|$

$$a_{ij} = \begin{cases} 1, & \text{если } a_i \in V_j; \\ 0, & \text{если } a_i \notin V_j, \end{cases} \quad (i = 1, 2, \ldots, \; j = 1, 2, \ldots, m).$$

Ответ y можно изобразить логическим массивом $y_1, y_2, \ldots, y_m$. Равенство $y_j = 1$ означает, что множество V_j входит в подсистему r множеств из системы, покрывающих все A; $y_j = 0$ — что V_j не входит в эту подсистему (случай, когда искомого покрытия не существует, можно изобразить, например, присвоив всем y, значение 0). При таких способах изображения $l(x) = mn$, $l(y) = m$.

Пусть множества $V_{i_1}, V_{i_2}, \ldots, V_{i_r}$ каким-либо способом выбраны, и $y_1, y_2, \ldots, y_m$ — соответствующие этому выбору значения элементов массива y. Если

$$\xi_i := \sum_{j=1}^{m} a_{ij} y_j = 0 \quad (1 \leqslant i \leqslant m),$$

то $a_i \notin \bigcup_{s=1}^{r} V_{i_s}$, в противном случае $a_i \in \bigcup_{s=1}^{r} V_{i_s}$. Таким образом, выбранная подсистема подмножеств покрывает A в том и только в том случае, когда $\xi_i \geqslant 1 \; (i = 1, \ldots, n)$.

Итак, задача о покрытии эквивалентна следующей целочисленной задаче линейного программирования: найти $y_1, \ldots, y_m$, принимающие значения 0 или 1, удовлетворяющие системе неравенств

$\sum_{j=1}^{m} a_{ij} y_j \geqslant 1 \; (j = 1, 2, \ldots, n)$, где $\| a_{ij} \|$ — заданная матрица из нулей и единиц, и равенству

$$\sum_{j=1}^{m} y_j = r.$$

Примером задачи о покрытии может служитьзадача минимизации ДНФ.

Задача о камнях.

Камни имеют веса $a_1, a_2, ..., a_n$. Можно ли выбрать камни с заданным суммарным весом

$$\sum_{s=1}^{r} a_{j_s} = b \ (r \leqslant n).$$

Аргумент x — это указатель n —числа камней в куче и массив их весов $(a_1, a_2, ..., a_n)$; y — указатель r— числа отобранных камней и массив их номеров I $(j_1, j_2, ...j_r)$. Для проверки достаточно выбрать из массива (a_i) элементы с номерами $j_1, ..., j_r$, сложить их и сравнить с b. Это можно сделать за $O\ (ln)$ действий, где l — число разрядов для изображения любого из чисел a_i или b .

Эта задача также эквивалентна целочисленной задаче линейного программирования — найти $y_1, y_ъ ..., y_n$, принимающие значения 0 или 1 и удовлетворяющие равенству

$$\sum_{j=1}^{n} a_j y_j = b.$$

Многомерная задача о камнях является естественным обобщением предыдущей задачи. Камни характеризуются матрицей параметров $\| a_{ij} \| (i = 1, 2, \ldots, m, \ j = 1, 2, \ldots, n)$. Требуется выбрать камни так, чтобы выполнялись условия

$$\sum_{s=1}^{r} a_{ij_s} = b_i \quad (i = 1, \ \ldots, \ m),$$

или, что то же самое, найти $y_1, y_2,...,y_n$, принимающие значения 0 или 1 и удовлетворяющие равенствам

$$\sum_{j=1}^{n} a_{ij} y_j = b_i \quad (i = 1, \ \ldots, \ m).$$

Размерность этой задачи не больше $O\ (mn \ \max\ (l\ (a_{ij}), l\ (b_i)))$ действий, а проверка ответа требует не более чем такого же по порядку числа действий.

Задачи перебора со степенной проверкой часто возникают при исследовании дискретных моделей технологических, организационных и экономических систем.

Степенная (полиномиальная) сводимость и универсальные задачи.

Некоторые задачи перебора со степенной проверкой и сами решаются за степенное время. Однако для многих часто рассматриваемых задач попытки найти способы их решения, требующие степенного времени, не привели к цели. Пока не удалось ни доказать, ни опровергнуть предположение о том, что любую задачу перебора со степенной проверкой можно решить за степенное время. Вместо этого С. А. Куком и Л. А. Левиным почти одновременно были введены понятия степенной сводимости и универсальных задач перебора, а также приведены примеры таких задач. Задача *A* называется *степенным образом (полиномиально) сводимой* к задаче *B,* если по алгоритму решения *B* можно построить алгоритм решения *A,* трудоемкость которого степенным образом зависит от размерности данных задачи *A* и трудоемкости алгоритма решения *B.* Если две задачи полиномиально сводимы друг к другу, то они называются *полиномиально эквивалентными.* В частности, все задачи, решаемые за степенное время, полиномиально эквивалентны.

Задача *A* перебора со степенной проверкой называется *универсальной задачей перебора,* если любая задача со степенной проверкой полиномиально сводима к *A.* Следовательно, если какую-нибудь универсальную задачу можно решить за степенное время, то и любую задачу перебора со степенной проверкой можно решить за степенное время.

Таким образом, поиск хорошего алгоритма решения любой универсальной задачи перебора имеет примерно одинаковые шансы на успех.

Позднее Р. М. Карп доказал универсальность целого ряда задач дискретной математики (универсальные задачи Карп называет полными).

Задача об общем решателе задач перебора (ОРЗП) со степенным временем работы—это естественный пример универсальной задачи перебора. Действительно, пусть *R* (П, *x, n)*— такая программа, причем число действий, которые она тратит на решение задачи размерности *n,* имеет оценку

$$t(R,\ \Pi,\ n) \sim O\left(n^{c_1} t(\Pi,\ n)^{c_2},\ l(\Pi)^{c_3}\right),$$

где *t* (П, *n)* — оценка числа действий программы П для вычисления предиката *P (x, y)* при размерности *n,* а *l* (П) — длина программы П. Когда она не зависит от *n,* и $t(\Pi, n) \sim O\left(n^{c_4}\right)$, время решения рассматриваемой задачи перебора со степенной оценкой имеет

порядок $O(n^{c_1 + c_2 c_4})$. Если же длина программы П зависит от n степенным образом, $l(\Pi) \sim O(n^{c_5})$, то оценка числа действий для решения этой задачи все равно степенная: $O(n^{c_1 + c_2 c_4 + c_3 c_5})$.

Этот результат можно считать тривиальным ввиду специфического характера задачи об общем решателе. Однако от нее можно перейти к другим универсальным задачам с более привычными комбинаторными формулировками.

Простые программы.

Это программы последовательных вычислений. Они удовлетворяют следующим условиям.

1. В них нет безусловных и условных передач управления, а все арифметические действия заменены последовательностями логических действий с двоичными разрядами изображений этих чисел в позиционной двоичной системе счисления. Таким образом, они состоят только из логических действий

$$Y_1 \vee Y_2 \to Y_3;\quad Y_1 \,\&\, Y_2 \to Y_3;\ Y_1 \to Y_3;\ \neg Y_1 \to Y_2.$$

2. Элементы поля переменных, которым присваиваются результаты этих операций, различны между собой.

Операнды логических действий простой программы, т. е. их аргументы или результаты — это либо данные, либо результаты предшествующих логических действий. Они могли бы еще быть константами, но каждое действие с постоянным операндом можно либо исключить из простой программы, либо упростить. Действительно, действия $0 \vee Y_1 \to Y_2$; $1 \,\&\, Y_1 \to Y_2$ заменяются пересылкой значения $Y_1 \to Y_2$. В остальных случаях ($1 \vee Y_1$, $0 \,\&\, Y_1$, а также действия только над константами) результат действия не зависит от данных, с которыми работает простая программа, и известен заранее. Поэтому такие действия можно исключить; в других действиях, использующих результат исключенных, сначала заменить соответствующие аргументы соответствующими константами, затем упростить или снова исключить эти действия и поступать таким образом, пока это потребуется (процесс обязательно закончится, так как в ходе него уменьшаются либо число действий программы, либо, по крайней мере, длина их записи). Аналогично можно исключить действия пересылки значений $Y_1 \to Y_2$.

Все переменные программы можно занумеровать так, что первые H номеров получают данные, с которыми работают задачи, а следующие — результаты действий в порядке их выполнения. Таким образом,

номера действий простой программы и ее результаты связаны между собой:

M': $u_{K'} \vee u_{L'} \to u_M \quad ; H;$

M'': $u_K \;\&\; u_{L''} \to u_M \quad ;$

M''': $\neg u_{K'''} \to u_{K'''} + H.$

Здесь K' и L' меньше, чем $M' + H$, K'' и L'' меньше $M'' + H$ и т. д.

Общий решатель для простых программ (ОРПП).

Пусть П *(x, y, n)* — простая программа вычисления предиката *P (x, y)* для задачи размер ности *n.* В дальнейшем *n* будет фиксировано, хотя и произвольным образом. В начало записи данных и (или) ответа можно приписать недостающее число нулей и считать, что *l (x) = l (y) = n.* Указания о произвольности и ограниченности *l (x)* и *l (y)* далее будут опускаться. В частности, выражения типа «для любого *x* ..» — надо понимать, как: «для любого *x,* удовлетворяющего условию *l (x*) = *n, ...*».

Программа, которая по заданной простой программе П *(x, y, n)* и аргументу *x* находит *y,* удовлетворяющий условию *P (x, y)* = 1, или убеждается, что такого *y* не существует (как указано ранее, рассматриваются только ответы *y* ограниченной длины *n),* называется общим решателем для простых программ. Если ее длина и время работы степенным образом зависят от *n* и длины простой программы П — это *общий решатель для простых программ со степенным временем работы.* Требовать еще степенной зависимости от времени работы программы П не нужно, так как последнее пропорционально ее длине.

Теорема 1. Задачи об ОРПП и ОРЗП полиномиально эквивнлентны.

Полиномиальная сводимость ОРЗП к ОРПП очевидна. Гораздо содержательнее тот факт, что и обратно по ОРПП со степенным временем работы строится аналогично работающий ОРЗП. Полное его доказательство требует существенного уточнения понятия программы для вычислительной машины и здесь не приводится, но основные идеи ниже будут указаны.

Прежде всего по заданной программе П *(x, y)* вычисления предиката *P (x, y)* и оценке ее трудоемкости строится простая программа П'*(x, y, n*), вычисляющая тот же предикат, для каждого *n* программа — своя. Время этого построения и длина получаемой простой программы степенным образом зависят от длины программы П *(x, y)* и оценки ее трудоемкости. Таким образом, если существует ОРПП со степенным временем работы, то аналогичным образом работающая ОРПП состоит из двух этапов.

1 Построение по заданным П *(x, y), n* и оценке времени работы программы П простой программы *П′ (x, y, n).*
2. Применение ОРПП для поиска ответа *y.*

Ациклический граф работы программы П *(x, y).* Пусть в программе П *(x, y) — M* действий Тогда можно построить ориентированный граф ее виртуальной работы. Вершины этого графа имеют две координаты *V (a, t),* где первая показывает, к какому действию относится эта вершина, а вторая — к какому моменту времени.

Если после выполнения в момент t' действия a' может оказаться, что в следующий момент $t' + 1$ будет выполняться действие a'', в графе G имеется ребро $(v(a', t'), v(a'', t' + 1))$. При этом мы не обращаем внимания на то, что в действительности команда a' в момент t' никогда не выполняется, но учитываем, что в последнем Т-м ряду вершин графа G, соответствующем последнему возможному моменту ее работы, может выполняться только действие конца работы программы П *(x, y)* (см. рис. 3).

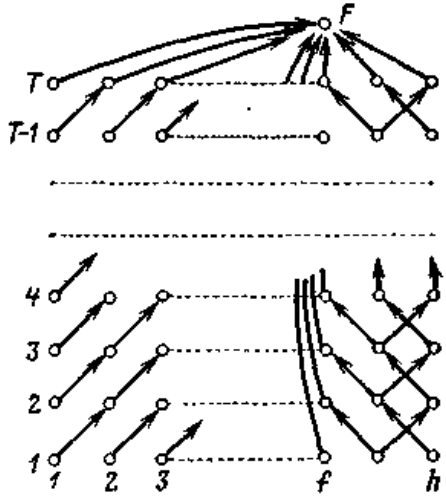


Рис. 3

Кроме того, вводим вершину *F* (общий конец) и соединяем ее ребрами со всеми вершинами, соответствующими концу *f* программы П *(x, y).* Так как все ребра графа *G* ведут вверх, последний является ациклическим. Число его вершин равно *MT,* т, е, степенным (с пока-

зателями степени 1) образом зависит от длины программы и времени ее работы, а число ребер не превосходит квадрата числа вершин [эту оценку легко уточнить до *MT (MT* — 1)/2].

Вытягивание графа в линию. Каждое арифметическое действие программы П *(x, y)* можно заменить последовательностью логических действий над одноразрядными переменными, составляющими операнды этого действия и промежуточными. При этом, так как набор таких действий для любой машины фиксирован, длины заменяющих их последовательностей ограничены, и в результате этой замены длина программы и время ее выполнения увеличиваются не более чем в *c* раз, где *c* — некоторая константа.

Если в программе П *(x, y)* имеется безусловный переход, то как и для действий с переходом к следующему в графе G из любой соответствующей вершины выходит только одно ребро. По два ребра выходит из вершин, соответствующих условным переходам, имеющим вид: если $z = 1$, то выполнять действие *A,* иначе выполнять *B* (обычно *B* — следующее по номеру действие).

Преобразование программы П (x, *y)* состоит в следующем. Прежде всего каждой промежуточной переменной u_k будет присвоено значение 0. Соответствующий участок программы является линейным и состоит из простейших действий $0 \to u_k$ $(k = M + 1)$. Затем каждое действие программы будет заменено вычислением логической формулы. Например, вместо действия $u_g > u_h \to u_k$ переменной u_k будет присвоено значение формулы $[w \& (u_g \vee u_h)] \vee [\neg w \& u_k]$.

После этого будут рассматриваться действия условного перехода. Следующие за ним ветви графа G будут «вытягиваться» в одну, состоящую сначала из действий первой ветви, а затем — из действий второй. Будут уточняться значения *w* в соответствующих формулах, а рассматриваемое действие условного перехода будет выброшено из программы.

Наконец, каждая формула снова будет заменена последовательностью действий, а все их результаты присвоены различным переменным. При этом мы получим простую программу, вычисляющую тот же предикат *P (x, y)* и нужно только оценить его длину.

Порядок ликвидации действий условного перехода. В ациклическом графе имеется хотя бы одно разветвление, после которого различные ветви этого графа только сходятся (см. рис.3). Назовем такое разветвление заключающим. На каждом из выходящих из *v′* разветвлений лежат вершины, соответствующие элементарным логическим действиям программы П *(x, y),* выполняемые подряд.

Выберем какое-либо заключающее разветвление и две ветви, сходящиеся в каком-либо месте графа G (так как в нем есть одна конечная

вершина), заменим идущими в заданном порядке действиями сначала одной ветви, а затем другой. Мы уже говорили, что каждое действие было заменено вычислением логической формулы. В ветви, начинающейся с действия *A,* эти формулы будут иметь вид

$$[z \,\&\, (u_g u_h)] \vee [\neg z \,\&\, u_k] \rightarrow u_k,$$

а в другой ветви

$$[\neg z \,\&\, (u_{g'} u_{h'})] \vee [z \,\&\, u_{k'}] \rightarrow u_{k'}.$$

При этом к длине программы прибавится не более чем длина одной из ветвей (если все соответствующие действия принадлежат к другим ветвям графа G), а число разветвлений уменьшится на единицу, и граф G ′ останется ациклическим.

Пусть в графе *G* еще остались разветвления. Опять берем заключающее, вытягиваем две выходящие из него ветви в одну, но преобразование формул производим чуть сложнее. Если в формуле, соответствующей действию программы П *(x, y)* условие *w* еще не заменялось каким-то образом, мы действуем, как и ранее при выбрасывании условного перехода в первый раз. В противном случае будем считать, что формула имеет вид

$$[Q \,\&\, (\ldots)] \vee [\neg Q \,\&\, u_{\tilde{k}}] \rightarrow u_{\tilde{k}},$$

где Q — логическое пересечение элементарных условий и их отрицаний. Заменим ее на $[\tilde{z}Q \,\&\, (\ldots)] \vee [\neg (\tilde{z}Q) \,\&\, u_{\tilde{k}}] \rightarrow u_{\tilde{k}}$ или

$$[\neg \tilde{z}Q \,\&\, (\ldots)] \vee [\neg (\tilde{z}Q) \,\&\, u_{\tilde{k}}] \rightarrow u_{\tilde{k}},$$

где $\tilde{z}$ — элементарное условие в ликвидируемом действии условного перехода.

Таким образом, новое $\tilde{Q}$, равное $\tilde{z}Q$ или $\neg \tilde{z}Q$, тоже является пересечением элементарных логических условий их отрицаний.

Длина полученной программы. Длина полностью спрямленной программы, состоящей только из вычисления формул, не больше суммы длин всех ветвей графа G, т. е. числа его дуг. Последнее, как указано ранее, не больше $(MT)^2$. В каждой формуле нужно вычислить соответствующее условие *Q,* а затем произвести не более пяти действий. Но *Q* — логическое пересечение элементарных условий и их отрицаний. Их число не больше, чем число всех условных переходов в исходной программе, а последнее не больше *M.* Таким образом, длина простой программы *П′ (x, y, n.),* вычисляющей предикат *P (x, y),* по порядку не больше M^3T^2 и степенным образом зависит от длины программы П *(x, y)* и размерности *n* (трудоемкость *T* зависит от *n* степенным образом).

Система логических уравнений.

Пусть задана простая программа П *(x, y, n)*. Найти *y* при заданном *x* так, чтобы выполнялось условие *P (x, y)* = 1, это значит решить приводимую далее систему логических уравнений, соответствующих командам логических действий программы П *(x, y, n)* относительно переменных, входящих в эту систему.

$u_м$ = const $_м$, когда в элемент *u [M]* переносится значение некоторого разряда аргумента *x;*

$u_M = u_K \vee u_L$ —когда в программе П *(x, y, n)* есть команда $u[M] := u[K] \vee u[L]$;

$u_M = u_K \,\&\, u_L$ —когда в этой программе имеется команда $u[M] := u[K] \,\&\, u[L]$;

$u_M = \neg u_K$ — когда имеется команда $u[M] := \neg u[K]$;

u_s = 1 — если элемент *u [S]* массива *u* должен получить значение *P (x, y).*

Значения $u_м$, соответствующие разрядам аргумента *y,* не фигурируют в левых частях этих уравнений. Зато в правых частях они встречаются. Когда эти переменные найдены, остальные просто вычисляются при помощи программы П *(x, y, n).*

Пусть эту систему уравнений можно решить (или убедиться, что решения нет), затратив $c\lambda^q$ действий, где λ — количество неизвестных (оно на *n* больше числа уравнений) системы, *c* и *q* — константы. Так как $\lambda \le n^p$ [число уравнений не больше количества команд программы П *(x, y, n)* и степенным образом зависит от *n],* число действий для решения этой системы уравнений не более cn^{pq}. Таким образом можно построить общий решатель для простых программ со степенным временем работы $Q(n^{pq})$. Отсюда следует, что решение описанной ранее системы логических уравнений является универсальной задачей перебора.

Эквивалентные системы неравенств.

Число переменных в рассматриваемой системе можно считать равным *s,* так как значение u_s= *P (x, y),* естественно, вычисляется последним. Переменные $u_м$ *(M* = 1, 2, ..., *s)* можно рассматривать как логические. Легко видеть, что уравнение $u_{M'} = u_{K'} \vee u_{L'}$ можно заменить тремя неравенствами

$u_{M'} > u_{K'}$;
$u_{M'} \geqslant u_{L'}$;
$u_{M'} \leqslant u_{K'} + u_{L'}$.

Аналогично уравнение $u_{M''} = u_{K''} \,\&\, u_{L''}$ эквивалентно неравенствам

$$u_{M''} \leqslant u_{K''};$$
$$u_{M''} \leqslant u_{L''};$$
$$u_{M''} \geqslant u_{K''} + u_{L''} - 1,$$

а уравнение $u_{M'''} = \neg u_{K'''}$ эквивалентно равенству $u_{M'''} + u_{K'''}$

Чтобы перейти к системе неравенств, определяющих задачу о покрытии, нужно добавить логические переменные v_M (M = 1, 2, ..., s), связанные с переменными u_M условиями $u_M + v_M = 1$. Тогда записанные ранее неравенства или равенства эквивалентны следующим:

$u_{M'} + v_{K'} \geqslant 1$;
$u_{M'} + v_{L'} \geqslant 1$;
$v_{M'} + u_{K'} + u_{L'} \geqslant 1$ (для уравнений $u_{M'} = u_{K'} \vee u_{L'}$);
$v_{M''} + u_{K''} \geqslant 1$,
$v_{M''} + u_{L''} \geqslant 1$;
$u_{M''} + v_{K'} + v_{L''} \geqslant 1$ (для уравнений $u_{M''} = u_{K''} \,\&\, u_{L''}$);
$u_{M'''} + u_{K'''} \geqslant 1$;
$v_{M'''} + u_{K'''} \geqslant 1$ (для уравнений $u_{M'''} = \neg u_{K'''}$).

Равенства $u_M + v_M = 1$ эквивалентны системе неравенств $u_M + v_M \geq 1$ (M =1, 2, ..., s) и одному равенству

$$\sum_{M=1}^{s} (u_M + v_M) = s.$$

Действительно, если хотя бы одно из выражений $u_M + v_M$ строго больше единицы, их сумма строго больше s.

Универсальность задачи о покрытии.

Приведенная ранее система неравенств и одного равенства, с одной стороны, эквивалентна универсальной задаче перебора — системе логических уравнений, а с другой — следующей задаче о покрытии:

множество A состоит из 4s +1 элемента α_M, β_M, γ_M, δ_M (M=1, 2, …, s) и ε_s;

система покрывающих подмножеств W — из 2s множеств u_M и v_M ($M = 1, \ldots, s$);

для каждого элемента множества A матрица инцидентности $\| a_{ij} \|$ определяет, каким подмножествам из системы W он принадлежит;

переменные u_M или v_M равны единице в том и только в том случае, когда множества u_M или соответственно v_M входят в покрывающую систему подмножеств;

число подмножеств в этой подсистеме равно s.

Таким образом, общая задача о покрытии является универсальной задачей перебора.

Универсальность многомерной задачи о камнях.

Пусть $y_1, y_2, ..., y_i$ — логические переменные, удовлетворяющие неравенству $\sum_{i=1}^{l} y_i \geqslant k$. Легко видеть, что $k \leq l$. При $k = l$ это неравенство эквивалентно равенству $\sum_{i=1}^{l} y_i = l$, которое выполняется, только если все $y_i = 1$. При $k < l$ можно ввести $l—k$ дополнительных переменных $z_1, ..., z_{l-k}$ и заменить это неравенство эквивалентным равенством

$$\sum_{i=1}^{l} y_i + \sum_{j=1}^{l-k} z_j = l.$$

Действительно, так как вторая сумма в этом равенстве не больше, чем $l — k$, первая не может быть меньше единицы.

Поэтому рассматриваемые ранее неравенства эквивалентны следующим равенствам, в которые введены дополнительные логические переменные w_M, z_M, q_M и r_M:

$$u_M + v_M + w_M = 2 \text{ вместо неравенств } u_M + v_K \geqslant 1;$$
$$u_M + v_L + z_M = 2 \text{ вместо } u_M + v_l \geqslant 1;$$
$$v_M + u_K + u_L + q_M + r_M = 3 \text{ вместо } v_M + u_K + u_L \geqslant 1;$$
$$v_M + u_K + w_M = 2 \text{ вместо } v_M + u_K \geqslant 1;$$
$$v_M + u_l + z_M = 2 \text{ вместо } v_M + u_L \geqslant 1;$$
$$u_M + v_K + v_L + q_M + r_M = 3 \text{ вместо } u_M + v_K + v_l \geqslant 1.$$

Остальные неравенства были написаны вместо соответствующих равенств, которые можно теперь восстановить:

$$u_M + v_M = 1 \text{ вместо } u_M + v_M \geqslant 1;$$
$$u_M + u_K = 1 \text{ вместо } u_M + u_K \geqslant 1;$$
$$v_M + v_K = 1 \text{ вместо } v_M + v_K \geqslant 1.$$

Равенство

$$\sum_{M=1}^{s} (u_M + v_M) = s$$

можно исключить, так как оно следует из условий

$u_M + v_M = 1$ $(M = 1, 2, \ldots, s)$.

Эта система равенств определяет многомерную задачу о камнях, которая, таким образом, тоже является универсальной задачей перебора.

Универсальность одномерной задачи о камнях следует из того, что эта задача полиномиально эквивалентна многомерной задаче о камнях. Для доказательства эквивалентности построим одномерную задачу, имеющую одинаковые решения с данной многомерной задачей с длиной данных x', степенным образом зависящей от длины данных x для исходной задачи.

Пусть многомерная задача о камнях определяется системой равенств

$$\sum_{j=1}^{n} a_{ij} y_j = b_i \quad (i = 1, 2, \ldots, m),$$

где все a_{ij} и b_i — целые числа. Эквивалентная ей одномерная задача о камнях имеет вид:

$$\sum_{j=1}^{n} (a_{1j} + c_1 a_{2j} + c_1 c_2 a_{3j} + \ldots + c_1 c_2 \ldots c_{m-1} a_{mj}) y_j = \\ = b_1 + c_1 b_2 + \ldots + c_1 \ldots c_{m-1} b_m;$$

при этом

$$c_i = 4 \sum_{j=1}^{n} |a_{ij}| \qquad (i = 1, 2, \ldots, m).$$

Так как в каждом из уравнений для многомерной задачи о камнях есть хотя бы один ненулевой коэффициент a_{ij}, а все эти коэффициенты — целые, значения c_i не менее четырех.

Очевидно, что решение приведенной ранее одномерной задачи о камнях является также и решением рассматриваемой многомерной задачи. Остается только доказать обратное. Прежде всего несколько предварительных замечаний.

По индукции можно доказать неравенства

$$c_1 + c_1 c_2 + \ldots + c_1 c_2 \ldots c_i < 2 c_1 c_2 \ldots c_i \quad (i = 1, 2, \ldots, m).$$

Если $i = 1$, то это неравенство справедливо, так как $c_i > 0$. Пусть оно выполняется при $i = i'$. Если $i = i' + 1$, то, так как $c_i = c_{i'+1} \geqslant 4$,

$$c_1 + c_1 c_2 + \ldots + c_1 c_2 \ldots c_i = c_1 + c_1 c_2 + \ldots + c_1 c_2 \ldots c_{i'} + \\ + c_1 c_2 \ldots c_{i'} c_{i'+1} < 2 c_1 \ldots c_{i'} + c_1 \ldots c_{i'} c_{i'+1} = \\ = c_1 \ldots c_{i'} (2 + c_{i'+1}) < 2 c_1 \ldots c_{i'} c_{i'+1} = 2 c_1 \ldots c_i,$$

что и требовалось доказать.

Пусть все y_j ($j = 1, 2, ..., n$) — логические. Тогда

$$\left|\sum_{j=1}^{n} a_{ij}y_j\right| \leqslant \sum_{j=1}^{n} |a_{ij}||y_j| \leqslant$$

$$\leqslant \sum_{j=1}^{n} |a_{ij}| = \frac{c_i}{4} \qquad (i = 1, \ldots, m).$$

Значит, когда хотя бы одно из значений больше, чем $c_i/4$, решений исходной многомерной задачи о камнях не существует. В этом случае она эквивалентна, например, одномерной задаче о камнях

$$\sum_{j=1}^{n} y = n + 1.$$

Пусть теперь логические переменные $y_1, y_2, ..., y_n$ являются решением рассматриваемой одномерной задачи о камнях. Тогда

$$\sum_{j=1}^{n} (a_{1j} + c_1a_{2j} + c_1c_2a_{3j} + \ldots + c_1c_2\ldots c_{m-1}a_{mj})\, y_j -$$

$$- (b_1 + c_1b_2 + \ldots + c_1\ldots c_{m-1}b_m) =$$

$$= \sum_{i=1}^{m} \left(\sum_{j=1}^{n} a_{ij}y_j - b_i\right) c_1c_2\ldots c_{i-1} = 0.$$

Уже было доказано, что $\left|\sum_{j=1}^{n} a_{ij}y_j\right| < \frac{c_i}{4}$; значит, и $\sum_{j=1}^{n} a_{ij}y_j =$ $|b_i| < \frac{c_i}{4}$ $(i = 1, \ldots, m)$. Поэтому, если для $i > l$ $= b_i$, в частности, если $l = m$ и значений $i > l$ не существует, то выполняются условия

$$\left|\sum_{i=l}^{m} \left(\sum_{j=1}^{n} a_{ij}y_j - b_i\right) c_1\ldots c_{i-1}\right| =$$

$$= \left|\left(\sum_{j=1}^{n} a_{ij}y_j - b_l\right) c_1\ldots c_{l-1}\right| =$$

$$= \left|\sum_{i=1}^{l-1} \left(\sum_{j=1}^{n} a_{ij}y_l - b_l\right) c_1\ldots c_{i-1}\right| <$$

$$< \sum_{i=1}^{l-1} c_1\ldots c_{i-1}\left(\frac{c_i}{4} + \frac{c_l}{4}\right) = \frac{1}{2}\sum_{i=1}^{l-1} c_1\ldots c_l < c_1c_2\ldots c_{i-1}.$$

Отсюда $\left|\sum_{j=1}^{n} a_{ij} y_j - b_i\right| < 1$. Так как эта разность — целое число, она равна нулю. Таким образом, последовательно доказывается выполнение *m*-го, *(m*— 1)-го, ..., 1-го уравнений многомерной задачи о камнях, т. е. эквивалентность рассматриваемых задач.

Размерности рассматриваемых задач.

Пусть $\rho = \max_{\substack{i=1,\ \dots,\ m \\ j=1,\ \dots,\ n}} (|a_{ij}|, |b_i|)$. Тогда размерность исходной многомерной задачи о камнях удовлетворяет условию

$$r \geqslant \max (\lg \rho,\ (n+1)\, m).$$

Размерность *R* эквивалентной ей одномерной задачи о камнях равна длине записи всех ее коэффициентов и правой части. Все они по абсолютной величине не превосходят $c_1 \dots c_m < (4\rho)^m$. Значит, длина записи любого коэффициента не больше *2m + m* lg ρ, а длина записи всех условий задачи

$$R \leqslant (n+1)\,(2+\lg \rho)\, m \leqslant r^2 + 2r = O(q^2).$$

Время *q* перехода от многомерной задачи о камнях к эквивалентной ей одномерной степенным образом зависит от размерности *R* первой задачи. Если одномерная задача решается за степенное время O (R^c), то многомерную тоже можно решить за степенное время $q + O(r^{2c})$. Значит, одномерная задача о камнях универсальна.

1.3. Алгоритмы эвристического поиска

В рассмотренных примерах поиска решений число состояний невелико, поэтому перебор всех возможных состояний не вызвал затруднений. Однако при значительном числе состояний время поиска возрастает экспоненциально, и в этом случае могут помочь алгоритмы эвристического поиска, которые обладают высокой вероятностью правильного выбора решения. Рассмотрим некоторые из этих алгоритмов.

1.3.1. Алгоритм наискорейшего спуска по дереву решений

Пример построения более узкого дерева рассмотрим на примере задачи о коммивояжере. Торговец должен побывать в каждом из 5 городов, обозначенных на карте (рис.4).

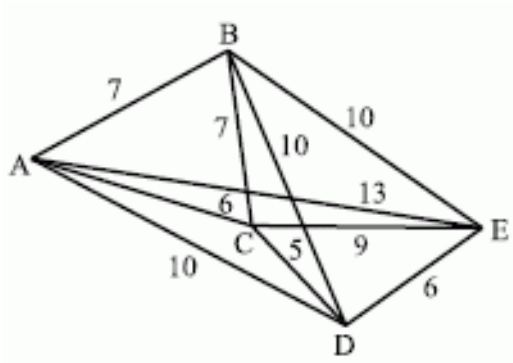


Рис.4.

Задача состоит в том, чтобы, начиная с города А, найти минимальный путь, проходящий через все остальные города только один раз и приводящий обратно в А. Идея метода исключительно проста - из каждого города идем в ближайший, где мы еще не были. Решение задачи показано на рис. 5.

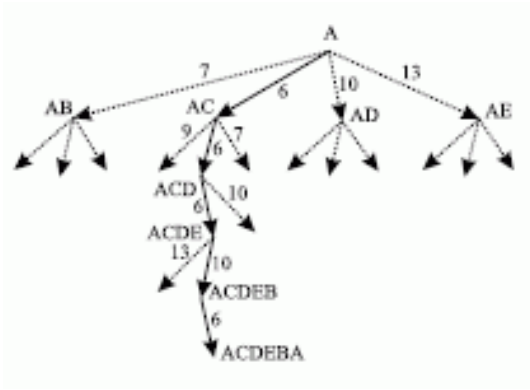


Рис.5.

Такой алгоритм поиска решения получил название алгоритма наискорейшего спуска (в некоторых случаях - наискорейшего подъема).

1.3.2. Алгоритм оценочных (штрафных) функций

Умело подобранные оценочные функции (в некоторых источниках - штрафные функции) могут значительно сократить полный перебор и привести к решению достаточно быстро в сложных задачах. В нашей задаче о людоедах и миссионерах в качестве самой простой целевой функции при выборе очередного состояния можно взять число людоедов и миссионеров, находящихся "не на месте" по сравнению с их расположением в описании целевого состояния. Например, значение этой функции f=x+y для исходного состояния f_0=6, а значение для целевого состояния f_1=0.

Эвристические процедуры поиска на графе стремятся к тому, чтобы минимизировать некоторую комбинацию стоимости пути к цели и стоимости поиска. Для задачи о людоедах введем оценочную функцию:

```
f(n) = d(n) + w(n)
```

где d(n) - глубина вершины n на дереве поиска и w(n) - число находящихся не на нужном месте миссионеров и людоедов. Эвристика заключается в выборе минимального значения f(n). Определяющим в эвристических процедурах является выбор оценочной функции. Рассмотрим вопрос о сравнительных характеристиках оценочных целевых функций на примере функций для игры в "8" ("пятнашки"). Игра в "8" заключается в нахождении минимального числа перестановок при переходе из исходного состояния в конечное (терминальное, целевое).

2	8	3
1	6	4
7	*	5

1	2	3
8	*	4
7	6	5

Рассмотрим две оценочные функции:

$h_1(n) \ \& = Q(n)$
$h_2(n) \ \& = P(n) + 3S(n),$

где Q(n) - число фишек не на месте; P(n) - сумма расстояний каждой фишки от места в ее целевой вершине; S(n) - учет последовательности нецентральных фишек (штраф +2 если за фишкой стоит не та, которая должна быть в правильной последовательности; штраф +1 за фишку в центре; штраф 0 в остальных случаях).

Сравнение этих оценочных функций приведено в таблица 1.

Таблица 1. Сравнение оценочных функций

Оценочная функция h	**Стоимость (длина) пути L**	**Число вершин, открытых при нахождении пути N**	**Трудоемкость вычислений, необходимых для подсчета h S**	**Примечания**
h_1 S_0 S_1	5 >18	13 100- 8!(=40320)	8	Поиск в ширину
h_2 S_0 S_1	5 18	11 43	8*2+8+1+1	Поиск в глубину

На основе сравнения этих двух оценочных функций можно сделать выводы.

- Основу алгоритма поиска составляет выбор пути с минимальной оценочной функцией.
- Поиск в ширину, который дает функция h_1, гарантирует, что какой-либо путь к цели будет найден. При поиске в ширину вершины раскрываются в том же порядке, в котором они порождаются.
- Поиск в глубину управляется эвристической компонентой 3S(n) в функции h_2 и при удачном выборе оценочной функции позволяет найти решение по кратчайшему пути (по минимальному числу раскрываемых вершин). Поиск в глубину

тем и характеризуется, что в нем первой раскрывается та вершина, которая была построена самой последней.

- Эффективность поиска возрастает, если при небольших глубинах он направляется в основном в глубь эвристической компонентой, а при возрастании глубины он больше похож на поиск вширь, чтобы гарантировать, что какой-либо путь к цели будет найден. Эффективность поиска можно определить как E=K/L*N*S, где K и S (трудоемкость) - зависят от оценочной функции, L - длина пути, N - число вершин, открытых при нахождении пути. Если договориться, что для оптимального пути E=1, то $K=L^0*N^0*S^0$.

1.3.3. Алгоритм минимакса

В 1945 году Оскар Моргенштерн и Джон фон Нейман предложили метод минимакса, нашедший широкое применение в теории игр. Предположим, что противник использует оценочную функцию (ОФ), совпадающую с нашей ОФ. Выбор хода с нашей стороны определяется максимальным значением ОФ для текущей позиции. Противник стремится сделать ход, который минимизирует ОФ. Поэтому этот метод и получил название минимакса. На рис. приведен пример анализа дерева ходов с помощью метода минимакса (выбранный путь решения отмечен жирной линией).

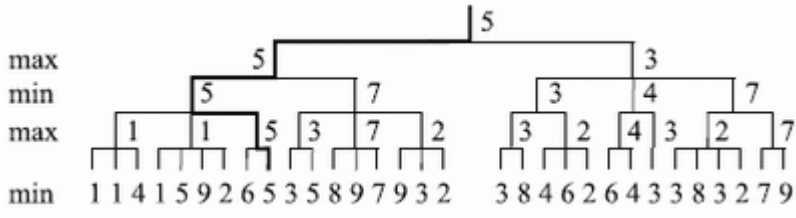


Рис.6. Дерево ходов

Развивая метод минимакса, назначим вероятности для выполняемых действий в задаче о миссионерах и людоедах:

```
P([2 : 0]R) = 0; 8; P([1 : 1]R) = 0; 5;
P([0 : 2]R) = 0; 9;
P([1 : 0]R) = 0; 3; P([0 : 1]R) = 0; 3:
```

Интуитивно понятно, что посылать одного людоеда или одного миссионера менее эффективно, чем двух человек, особенно на начальных этапах. На каждом уровне мы будем выбирать состояние по критерию P_i. Даже такой простой подход позволит нам избежать части тупиковых состояний в процессе поиска и сократить время по сравнению с полным перебором. Кстати, этот подход достаточно распространен в экспертных продукционных системах.

1.3.4. Альфа-бета-процедура

Теоретически, это эквивалентная минимаксу процедура, с помощью которой всегда получается такой же результат, но заметно быстрее, так как целые части дерева исключаются без проведения анализа. В основе этой процедуры лежит идея Дж. Маккарти об использовании двух переменных, обозначенных α и β (1961 год).

Основная идея метода состоит в сравнении наилучших оценок, полученных для полностью изученных ветвей, с наилучшими предполагаемыми оценками для оставшихся. Можно показать, что при определенных условиях некоторые вычисления являются лишними. Рассмотрим идею отсечения на примере рис. 7. Предположим, позиция A полностью проанализирована и найдено значение ее оценки α. Допустим, что один ход из позиции Y приводит к позиции Z, оценка которой по методу минимакса равна z. Предположим, что $z \leq \alpha$. После анализа узла Z, когда справедливо соотношение $y \leq z \leq \alpha \leq s$, ветви дерева, выходящие из узла Y, могут быть отброшены (альфа-отсечение).

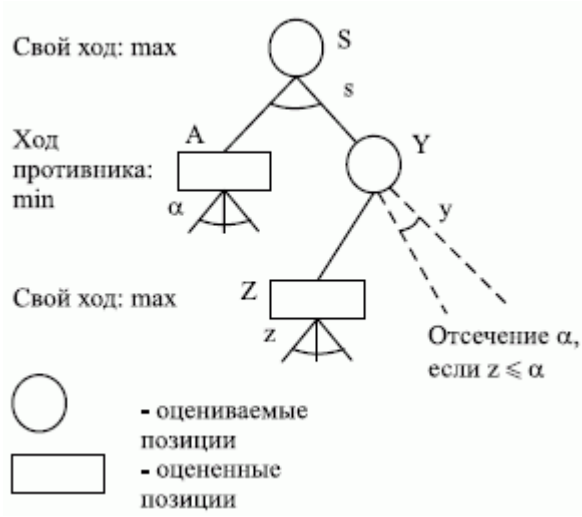


Рис.7. - отсечение

Если мы захотим опуститься до узла Z, лежащего на уровне произвольной глубины, принадлежащей той же стороне, что и уровень S, то необходимо учитывать минимальное значение оценки β, получаемой на ходах противника.

Отсечение типа β можно выполнить всякий раз, когда оценка позиции, возникающая после хода противника, превышает значение β. Алгоритм поиска строится так, что оценки своих ходов и ходов противника сравниваются при анализе дерева с величинами α и β соответственно. В начале вычислений этим величинам присваиваются значения $+\infty$ и $-\infty$, а затем, по мере продвижения к корню дерева, находится оценка начальной позиции и наилучший ход для одного из противников.

Правила вычисления α и β в процессе поиска рекомендуются следующие:

1. у MAX вершины значение α равно наибольшему в данный момент значению среди окончательных возвращенных значений для ее дочерних вершин;
2. у MIN вершины значение β равно наименьшему в данный момент значению среди окончательных возвращенных значений для ее дочерних вершин.

Правила прекращения поиска:

3. можно не проводить поиска на поддереве, растущем из всякой MIN вершины, у которой значение β не превышает значения α всех ее родительских MAX вершин;
4. можно не проводить поиска на поддереве, растущем из всякой MAX вершины, у которой значение α не меньше значения β всех ее родительских MIN вершин.

На рис. 8 показаны $\alpha - \beta$ отсечения для конкретного примера.

Таким образом, $\alpha - \beta$ -алгоритм дает тот же результат, что и метод минимакса, но выполняется быстрее.

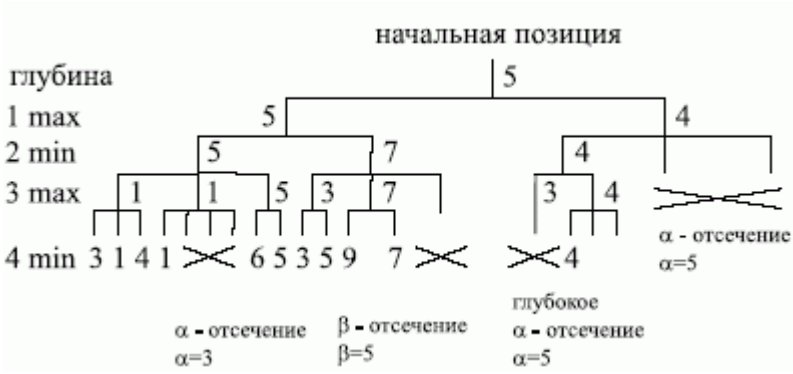


Рис.8. a-b отсечение для конкретного примера

Использование алгоритмов эвристического поиска для поиска на графе И, ИЛИ выигрышной стратегии в более сложных задачах и играх

(шашки, шахматы) не реален. По некоторым оценкам игровое дерево игры в шашки содержит 10^{40} вершин, в шахматах 10^{120} вершин. Если при игре в шашки для одной вершины требуется 1/3 наносекунды, то всего игрового времени потребуется 10^{21} веков. В таких случаях вводятся искусственные условия остановки, основанные на таких факторах, как наибольшая допустимая глубина вершин в дереве поиска или ограничения на время и объем памяти.

Многие из рассмотренных выше идей были использованы А. Ньюэллом, Дж. Шоу и Г. Саймоном в их программе GPS. Процесс работы GPS в общем воспроизводит методы решения задач, применяемые человеком: выдвигаются подцели, приближающие к решению; применяется эвристический метод (один, другой и т. д.), пока не будет получено решение. Попытки прекращаются, если получить решение не удается.

Программа STRIPS (STanford Research Institut Problem Solver) вырабатывает соответствующий порядок действий робота в зависимости от поставленной цели. Программа способна обучаться на опыте решения предыдущих задач. Большая часть игровых программ также обучается в процессе работы. Например, знаменитая шашечная программа Самюэля, выигравшая в 1974 году у чемпиона мира, "заучивала наизусть" выигранные партии и обобщала их для извлечения пользы из прошлого опыта. Программа HACHER Зуссмана, управляющая поведением робота, обучалась также и на ошибках.

1.4. Метод ветвей и границ

Исследование проблем перебора показывает, что среди разрешимых задач могут быть хорошо и плохо реализуемые. Однако подобно тому, как в неразрешимой задаче возможны разрешимые частные случаи, так и в «плохой» задаче возможны «хорошие» частные случаи. Как правило, это объясняется тем, что при некоторых конкретных значениях исходных данных задачи удается полный перебор заменить перебором с отсечениями; иначе говоря, найти такие «хорошо» вычислимые условия, которые позволяют отбрасывать некоторые подмножества вариантов, удовлетворяющие этим условиям (т. е. не выполнять для них перебор). Наиболее типичным вычислительным методом сокращения перебора является метод ветвей и границ, к рассмотрению которого мы переходим. Этот метод нельзя назвать алгоритмом, поскольку его основные блоки (и прежде всего,

вычисление условий отсечения) зависят от конкретной задачи; скорее, это вычислительная схема. Эффективность же метода может зависеть от конкретных данных задачи, и в «плохих» случаях привести к тому же полному перебору.

Ветвление. Пусть имеется конечное множество M ситуаций или вариантов и функция F, принимающая различные значения в зависимости от выбранного варианта. Требуется среди вариантов найти оптимальный, т. е. такой, на котором функция F принимает минимальное значение (или максимальное в зависимости от условия задачи).

Метод ветвей и границ отыскания оптимального варианта состоит из ветвления и отсечений. Рассмотрим сначала ветвление.

Принимаем какой-либо принцип разбиения множества M на подмножества M_i такие, что $\bigcup_i M_i = M$, $M_i \cap M_j = 0$, $i \neq j$

Затем, пользуясь этим же принципом, разбиваем множества на части и т. д. После некоторого шага разбиения каждое множество содержит по одному варианту.

На каждом шаге вариант, оптимальный для всего множества M, принадлежит одному из M_i и является для него оптимальным. Поэтому его достаточно искать среди оптимальных вариантов для подмножеств M_i, составляющих множество M. Этим самым решение задачи для всего множества M сводится к решению задач для составляющих его множеств M_i и последующему отысканию оптимальных среди найденных для них решений.

Процесс разбиения множества M на подмножества становится наглядным, если его изобразить с помощью ориентированного графа следующим образом. Каждому множеству M_i, полученному при последовательном разбиении множества M на части, ставится в соответствие вершина графа.

Обозначим вершины графа так же, как и множества, которым они соответствуют, т. е. M_i. От вершины M_i к вершине M_j проводим направленное ребро, если множество M_j получено непосредственным разбиением на части множества M_i. Полученный граф является деревом, поскольку в каждую его вершину входит единственное ребро и граф имеет одну начальную вершину. [1]

(В более общем случае может быть рассмотрено разбиение множества на пересекающиеся подмножества. Кроме того, в некоторых задачах производится разбиение одного и того же множества на части несколько раз различными (независимыми) способами В обоих случаях граф, изображающий процесс разбиения, не является деревом.)

Построение дерева с помощью разбиения множества вариантов на подмножества называется *ветвлением.*

Построенное указанным ранее способом дерево имеет число конечных вершин, равное числу элементов (вариантов) в множестве *M:* для каждого варианта одна конечная вершина. Такое дерево назовем деревом полного перебора.

Пример 1. Построить дерево путей из v_1 в v_6 в графе рис. 9 (на рисунке вершины обозначены индексами *i* вместо v_i).

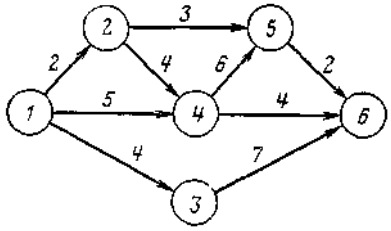


Рис. 9

Решение. Таких путей в рассматриваемом графе шесть:

$$L_1=(v_1,\ v_3,\ v_6);\quad L_2=(v_1,\ v_4,\ v_6);\quad L_3=(v_1,\ v_2,\ v_4,\ v_6);$$
$$L_4=(v_1,\ v_2,\ v_4,\ v_5,\ v_6);\ L_5=(v_1,\ v_4,\ v_5,\ v_6);\ L_6=(v_1,\ v_2,\ v_5,\ v_6),$$

Ветвление производим по принципу: множество M_i путей, проходящих через вершину v_i, разбиваем на непересекающиеся подмножества M'_j путей, содержащих ребра (v_i, v_j). На рис. 10 изображено дерево всех путей (полного перебора).

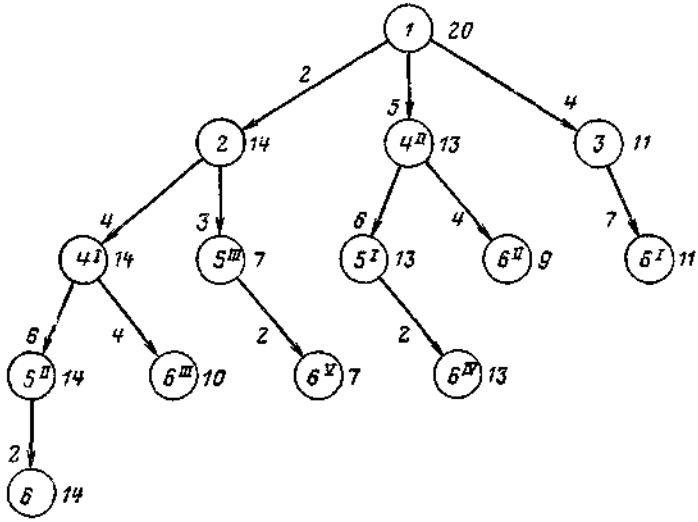


Рис.10.

В вершинах римскими цифрами указаны номера путей, образующих соответствующие множества.

Пусть маршрут — это путь в дереве от его корня к какой-нибудь конечной вершине. Функция F определена на конечных вершинах дерева полного перебора. Каждой конечной вершине такого дерева соответствует один и только один маршрут с концом в этой вершине. Поэтому функция F — это функция от маршрута.

Оценочные функции. Оценки. Разбиение множества на подмножества и независимое их рассмотрение приобретает смысл лишь в случае, если на некоторых этапах построения дерева полного перебора удается установить, что в каком-то подмножестве нет варианта, оптимального для всего множества. Дальнейшее ветвление в этой вершине не происходит. От дерева полного перебора отсекаются ветви с корнем в ней (отсекаются вершины и ребра следующие за ней). Исключение множеств вариантов из рассмотрения производится с помощью оценочных функций. Оценочная функция — это функция $m(M_i) = m_i$ заданная на вершинах дерева полного перебора, возможно, исключая корень, и равная в его конечных вершинах соответствующим значениям функции F, а в остальных вершинах M_t дающая нижнюю или верхнюю границу значений функции F для вариантов, входящих в множество M_i.

Оценочная функция, дающая нижнюю границу, в процессе разбиения множества на части не убывает, а дающая верхнюю границу — не возрастает. Действительно, пусть $m(M_i) = m_i$ — оценочная функция, дающая нижнюю границу. Это значит, что все значения вариантов из множества M_i не меньше числа m_i. Тогда для любого подмножества M_i нижняя граница значений вошедших в него вариантов не может быть меньше m_i . Поэтому если для некоторой части M_i не находится значение оценочной функции, большее m_i, считаем его равным m_i. Следует различать оценочные функции, определенные на любых подмножествах вариантов, и оценочные функции, определенные лишь на некоторых таких подмножествах. В первом случае при построении дерева можно использовать для ветвления произвольный принцип разбиения множеств вариантов на части. Во втором случае при ветвлении следует делить множества вариантов лишь на такие части, на которых определена рассматриваемая оценочная функция. Поэтому дерево полного перебора зависит, вообще говоря, от выбора оценочной функции.

Рассмотрим несколько примеров оценочных функций.

Пример 2. Пусть каждому ребру (M_k, M_l) дерева полного перебора поставлено в соответствие некоторое число $d_{kl} \geqslant 0$. Требуется среди маршрутов L найти такой, для которого величина

$$F(L) = \sum_{(M_k,\ M_l) \in L} d_{kl}$$

принимает минимальное значение.

Функция $m_i = \sum_{1}^{i} d_{kl}$, где суммирование производится лишь по ребрам, входящим в путь от корня до вершины M_i, является оценочной функцией, дающей нижнюю границу.

Пример 3. Если в дереве полного перебора примера 1 все конечные вершины имеют ранг n, то оценочной функцией, дающей нижнюю границу, является и функция

$$m_i = \sum_{1}^{i} d_{kl} + \sum_{i+1}^{n} \min d_{kl}, \quad \textbf{(1)}$$

где минимум ищется среди значений d_{kl} для ребер M_i-ветви одного ранга, знак $\sum_{i+1}^{n}$ означает суммирование по рангам ребер этой ветви (отсчитываемым от M_i).

Аналогично функция

$$n_i = \sum_{1}^{i} d_{kl} + \sum_{i+1}^{n} \max d_{kl} \quad \textbf{(2)}$$

является оценочной функцией, дающей верхнюю границу.

Пример 4. Если конечные вершины дерева полного перебора имеют неодинаковые ранги, то в примере 3 оценочные функции m_i и n_i принимают вид:

$$m_i = \sum_{1}^{i} d_{kl} + \sum_{i+1}^{s} \min d_{kl};\ n_i = \sum_{1}^{i} d_{kl} + \sum_{i+1}^{t} \max d_{kl}, \quad \textbf{(3)}$$

где знак $\sum_{i+1}^{s}$ означает суммирование по рангам ребер M_i-ветви, в которых представлены все пути, выходящие из M_i, а знак $\sum_{i+1}^{t}$ — суммирование по рангам ребер, в которых имеется хотя бы один путь, выходящий из M_i.

В примерах 2—4 для построения оценочной функции были использованы конкретные свойства изучаемых в задаче объектов (неотрицательность слагаемых d_{kl} и оценка их значений по рангам). Вообще основным принципом получения оценочной функции является индивидуальное изучение задачи.

Проиллюстрируем это следующим примером.

Прлмер 5. Рассмотрим задачу об отыскании одно о радиоактивного шарика среди n шариков, одинаковых в остальных отношениях. При этом используется прибор, позволяющий устанавливать, находится ли искомый шарик в рассматриваемой группе шариков; если да, прибор показывает единицу, если нет — нуль. Оценочной функцией любого подмножества шариков будем считать показание прибора при проверке этого подмножества. Такая оценочная функция определена на любом подмножестве шариков и тем самым на любом подмножестве вариантов размещения радиоактивного шарика. Всякое подмножество, для которого значение оценочной функции равно нулю, исключается из дальнейшего рассмотрения. Поэтому при каждой проверке исключается одно из подмножеств: либо проверяемое, либо его дополнение до множества шариков, оставшихся после исключений при предыдущих проверках.

Обозначим $k = k\ (n)$ минимальное число проб (вопросов), достаточных для выделения одного радиоактивного шарика из n. Можно доказать, что $k\ (n)$ не убывает при возрастании n. Оценим $k\ (n)$. Пусть мы спрашиваем о множестве M_1, содержащем n_1 шарик, $n_1 < n$. Если значение оценочной функции $m\ (M_1) = 1$, то на выделение радиоактивного шарика из n_1 шариков требуется $k\ (n_1)$ вопросов, при $m\ (M_1) = 0$ требуется $k\ (n — n_1)$ вопросов да один вопрос (о подмножестве M_1) уже задан. Поэтому

$$k(n) = \min_{n_1 < n} \max \{1 + k(n_1);\ 1 + k(n - n_1)\}. \quad (4)$$

Из (4) следует, что минимальное число проб получается, если каждый раз задавать вопрос о половине рассматриваемых шариков, т. е. когда

$$n_1 = \left[\frac{n-1}{2}\right] + 1,\ n_2 = n - n_1 = [n/2].$$

Методом индукции для *k (n)* получаем оценки

$$s - 1 \leqslant k(n) \leqslant s, \quad (5)$$

где $2^{s-1} < n \leqslant 2^s$. Отсюда

$$[\lg_2 n] \leqslant k(n) \leqslant [\lg_2 (n-1)] + 1. \quad (6)$$

Действительно, непосредственно убеждаемся в том, что (5), (6) верно для $n = 2$ и $n = 3$. Далее из предположения, что (5), (6) верно при $2^{s-3} < n \leqslant 2^{s-2}$ и учитывая значения n_1 и n_2, получаем, что эти оценки верны и при $2^{s-1} < n \leqslant 2^s$.

Из (6) следует, что минимальное число *k* проб, достаточное для выделения из *n* шариков одного радиоактивного, меньше числа проб при полном переборе. Это уменьшение получается с помощью оценочной функции: как было указано ранее, подмножества шариков, на которых она равна нулю, каждый раз отсекаются.

Значение оценочной функции для данного множества M_i назовем его оценкой и обозначим оц$M_i = m_i$. Она может не быть точной границей значений вариантов из M_i. Чем ближе оценка к точной границе, тем эффективнее применение ветвей и границ, ибо число отсекаемых вершин дерева полного перебора зависит, в частности, от силы оценки. Действительно, пусть в задаче отыскивается минимум и на некоторых подмножествах определены две оценочные функции, m_i и m'_i, причем m'_i — более точная, т. е. $m'_i \geq m_i$. И пусть найдено значение F_o для некоторого варианта. Если для какой-то вершины M_{i_o} имеем $m(M_{i_o}) \geq F_0$, то и $m'(M_{i_o}) \geqslant F_0$ и множество M_{i_o} не следует далее рассматривать, так как оно не содержит оптимального варианта. Но при $m(M_{i_o}) < F_0$ может оказаться $m'(M_{i_o}) \geq F_0$. Тогда при использовании оценочной функции m_i множество M_{i_o} нужно изучать дальше, а при использовании m'_i этого делать не следует. К сожалению, обычно, чем точнее оценка, тем больших по объему вычислений требует ее отыскание.

Так как точность оценки зависит, в частности, от того, насколько целесообразен принятый принцип разбиения на части рассматриваемого множества вариантов, то при выборе способа разбиения из нескольких возможных следует отдать предпочтение тому из них, при котором оценки полученных подмножеств более точные. Не менее важно, насколько сильно на этих подмножествах

различаются между собой значения оценочной функции. Лучшим является тот из способов разбиения (и такая оценочная функция), при котором разность между оценками подмножеств наибольшая.

Отсечение вариантов (ветвей). Можно указать следующие основные принципы отсечения ветвей.

а) Отсечение по сравнению с уже найденным значением функции *F*. Пусть ищется минимальное значение *F* и получены оценки снизу оцM_i для части вершин дерева и значение $F = F_0$ для некоторого варианта. Тогда в вершинах, где оц$M_i \geq F_0$, ветвление прекращается. Количество отсечений по этому способу тем больше, чем меньше F_0.

б) Отсечение по сравнению двух оценок. Его можно производить, когда для M_i строятся оценка снизу (ноц M_i) и оценка сверху (воцM_i). Если при этом для некоторых M_i и M_j оказывается, что ноц$M_i \geq$воцM_j, то при отыскании минимума F ветвление из вершины M_i прекращается.

в) Ветвление в данной вершине прекращается, если известно, что соответствующее ей подмножество не содержит оптимального варианта или известен оптимальный среди принадлежащих ему вариантов.

Порядок продолжения ветвления может быть выбран различными способами. Укажем основные из них.

1. Ветвление по минимальной нижней границе (при отыскании минимума *F).* Сначала строятся (все или некоторые) ребра, выходящие из корня. Затем в каждый момент ветвление производится в вершине с минимальной из уже найденных оценок снизу. Для всех вершин, в которых производится ветвление, можно строить все ребра, выходящие из них в дереве полного перебора, либо только часть этих ребер.

2. Развитие дерева по ветвям (последовательное построение ветвей). Скачала строим один маршрут, включая его конечную вершину. При этом получаем некоторое значение F_0, которым можно пользоваться при отсечении множества вариантов. Пусть M_k — первая от конца этого маршрута вершина, такая, что еще не построено выходящее из нее ребро (M_k, M_i) и при этом оц$M_i < F_0$, т. е. множество M_i не отсекается (отыскивается минимум). Строим ребро (M_k, M_i) и достраиваем какой-нибудь из проходящих через нее маршрутов до конца либо до вершины, в которой происходит отсечение. Далее строим ребро (M_j, M_i) в первой от конца этого маршрута (от конца построенной его части) вершине M_j, такой, что выходящее из нее ребро (M_j, M_i) еще не построено и при этом множество M_i не отсекается и т. д. При таком способе продолжения ветвления в построенном дереве обычно больше вершин, чем при ветвлении по минимальной нижней

границе, но запоминать приходится данные о меньшем количестве построенных вершин.

Никаких точных оценок числа отсечений при различных способах продолжения ветвления указать нельзя, все соображения здесь носят эвристический характер.

Пример 6. Используя оценку типа (3), найти путь максимальной длины из p_1 в p_6 в графе рис.9.

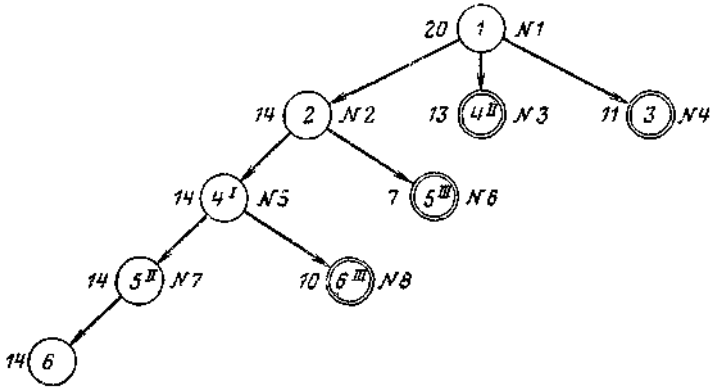


Рис.11.

На рис. 11 приведено решение при ветвлении по максимальной верхней границе, на рис.12, *а* и *б* — при развитии дерева по ветвям.

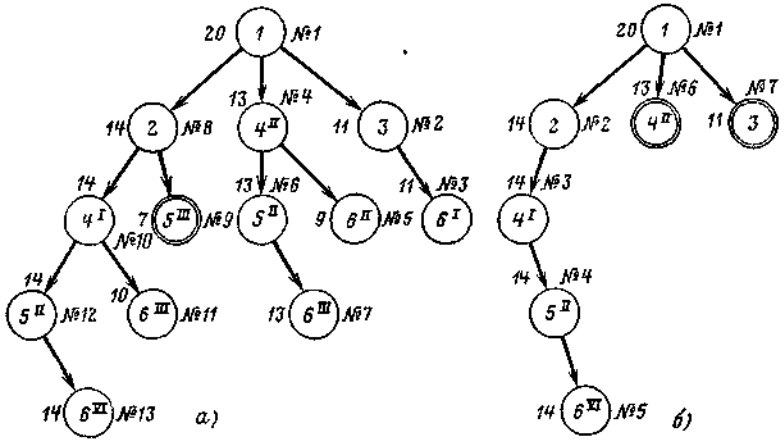


Рис.12.

Количество построенных при этом вершин дерева зависит от очередности развития ветвей.

На рисунках слева от вершин проставлены оценки, найденные по формулам типа (3), справа — номера вершин по порядку, появляющихся при построении дерева.

3. Еще один способ продолжения ветвления сочетает принципы рассмотренных ранее двух способов. Здесь задаем некоторое число $\delta > 0$. Пусть M_i— вершина, в которую мы попали при очередном шаге ветвлений. Если оцM_i превосходит минимальную из ранее найденных оценок не более чем на δr_i, где r_i — ранг рассматриваемой вершины, то ветвление продолжаем из этой вершины. В противном случае производим ветвление в вершине с минимальной нижней границей.

Отметим, наконец, что могут возникнуть дополнительные отсечения, если отыскивается не минимум функции F, а лишь выясняется, существует ли значение F, удовлетворяющее условию $F(M_i) \leqslant C$.

Задача об очередности выполнения работ. Постановка задачи. Пусть сеть комплекса работ задана рис.13, где каждому ребру соответствует работа.

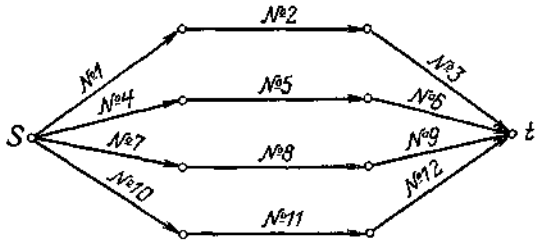


Рис.13.

Заданы постоянные продолжительности d_i всех работ сети (табл. 1).

Таблица 1

Номер работы i	1	2	3	4	5	6	7	8	9	10	11	12
Продолжительность d_i	10	5	3	12	7	10	28	4	6	23	7	3

Имеется нескладируемый ресурс в количестве 1 (например, башенный кран), который необходим при выполнении работ №2, 5, 8, 11. Для остальных работ он не нужен; потребность в других видах ресурса не учитывается (считается, что их всегда достаточно). Требуется указать такой порядок выполнения работ № 2, 5, 8, 11 (впредь будем их

называть ресурсными), чтобы задержка выполнения комплекса работ по сравнению с критическим временем $T_{кр} = 38$ была минимальной. Обозначим: t^p_i — ранний срок начала i-й работы (т. е. момент, раньше которого она не может начаться), $t^н_i$ — время начала i-й работы, t^o_i — время окончания i-й работы; $\tau_i = d_{i-1}$ Учитывая вид сети, для i-й ресурсной работы имеем $t^p_i = d_{i-1}$.

Ветвление. Всего имеется 24 варианта последовательности выполнения ресурсных работ (4! =24). Разобьем их на подмножества вариантов, в которых первой выполняется одна и та же ресурсная работа. Ее будем считать фиксированной. Это варианты с одинаковым началом. Таких подмножеств четыре (по шесть вариантов в каждом). Далее полученные подмножества разбиваем на части, в которых фиксирован порядок выполнения уже двух ресурсных работ и т. д.

Отсечение. При отыскании оценок снизу будем опираться на следующий факт: пусть задан некоторый момент времени f и календарный план выполнения ресурсных работ из некоторого их множества R найден с помощью формул

$$t^н_j = t + \sum_{\tau_i < \tau_j} d_i, \quad i, j \in R;$$

$$t^o_j = t^н_j + d_j, \qquad j \in R;$$

если в этом плане для всех i выполнено неравенство $t^н_i \geq t^p_i$, то он минимизирует величину

$$\max_{i \in R} (t^н_i + d_i + \tau_i - t).$$

При решении задачи время i принимается равным сроку окончания последней из ресурсных работ, фиксированных в данной группе вариантов. Вычисления производятся с помощью таблиц такого типа, как табл. 2: $t^н_i$ — срок начала i-й ресурсной работы при соответствующем таблице порядке выполнения ресурсных работ; T_i — длина пути (т. е. суммарная продолжительность) от начальной вершины к конечной, проходящего через i-ю ресурсную работу.

Таблица 2

i	t^p_i	d_i	τ_i	$t^н_i$	$T_i = t^н_i + d_i + \tau_i$	
2	10	5	3	10	18	
5	12	7	10	15	32	
8	28	4	6	22	32	$T_{макс} = 36$
11	23	7	3	26	36	

В табл. 2 приведены значения T для подмножества вариантов, в которых зафиксировано положение одной ресурсной работы — работы № 2. Имеем $t_2^н = 10 = t_2^р$; $t_5^н = t_2^н + d_2 = 10 + 5 = 15, t_8^н = t_5 + d_5 = 15 + 7 = 22$;

$t^н_{11}=t_8 + d_8 =22 + 4 = 26$, причем $t^н_8 = 22 < 28 = t^р_8$. Из таблицы получаем нижнюю границу $T = \max \{ T_2; T_5; T_8; T_{11}\} = 36$ продолжительности для всех вариантов рассматриваемого подмножества.

В первой строке табл. 2 выписаны данные для ресурсных работ, положение которых в рассматриваемой вершине уже зафиксировано. Для них время начала $t^н_i$ допустимое, т. е. $t_i^н \geqslant t_i^р$. Остальные ресурсные работы выписываются в порядке убывания τ_i. Для каждой из них $t^н_i$ равно сумме времени начала и продолжительности записанной перед ней (выше нее) ресурсной работы.

Заполнив таблицу, находим T — $\max T_i$. Если для ресурсных работ, положение которых в рассматриваемой вершине не зафиксировано, оказалось $t_i^н \geqslant t_i^р$, то найденное T — минимальное время выполнения комплекса работ для множества вариантов, соответствующих этой вершине, причем минимум достигается при выполнении ресурсных работ в порядке, указанном таблицей. Если же окажется, что для какой-либо ресурсной работы, положение которой не зафиксировано, $t_i^н < t_i^р$, то найденное время T является нижней границей для этого подмножества вариантов. Действительно, T получено без учета того, что работы нельзя начинать ранее их раннего срока начала.

При фиксации положения одной ресурсной работы № 8 получаем табл. 3.

Таблица 3

i	$t_i^р$	d_i	τ_i	t_i	T_i	
8	28	4	6	28	38	
5	12	7	10	32	49	
2	10	5	3	39	47	$T_{min} = 52$
11	23	7	3	44	52	

Поскольку здесь все $t_i^н \geqslant t_i^р$, число $T = \max T_i = 52$ равно минимальной продолжительности комплекса работ для множества вариантов, в которых первой из ресурсных работ выполняется работа № 8. Это время достигается при порядке выполнения ресурсных работ: №8, 5, 2, 11. Независимо от результатов дальнейших вычислений соответствующую ветвь графика развивать не следует.

Одновременно с подсчетом оценок строим дерево, например, развивая ветвь с наименьшей нижней границей (рис.14).

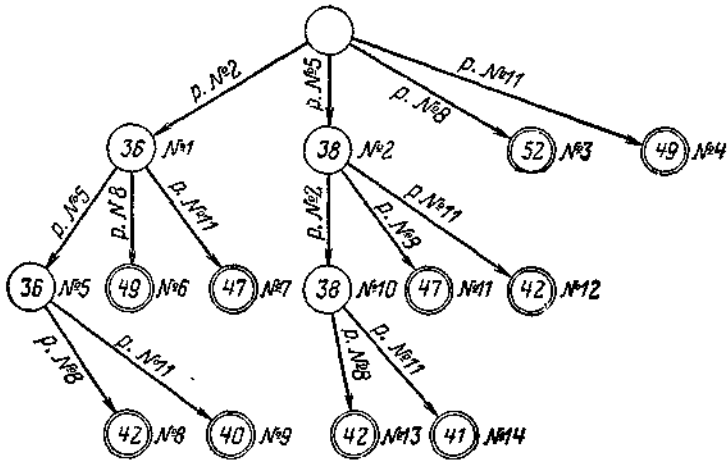


Рис.14.

Последовательность рассмотрения вершин указана номерами возле них на рисунке.

В данном примере числа 42 и 40 в вершинах № 8 и 9 являются значениями соответствующих вариантов. В вершинах № 3, 4, 6, 7, 11, 12 ветвление производить не следует по двум причинам: так как их оценки (на рисунке записаны в кружках) — минимумы для соответствующих групп вариантов и так как каждая из них больше уже полученного времени $t = 40$. Любой одной из этих причин достаточно для прекращения ветвления.

Неперебoрные задачи дискретной математики. Как уже говорилось, существуют «хорошие» неперебoрные задачи дискретной математики, которые всегда можно решить за время, степенным образом зависящее от размерности. Среди них *задача о максимальном потоке,* которой посвящены многочисленные исследования. Пусть дан ориентированный граф G, и отмечены две его вершины p' и p'', называемые *полюсами.* Функция $f(i, j)$, заданная на ребрах $(i, j) \in G$, называется *потоком,* если она удовлетворяет условиям

$$\sum_{(i,\ g) \in G} f(i,\ g) - \sum_{(g,\ j) \in G} f(g,\ i) = 0,\ g \neq p',\ p'',$$

где $a(i, j)$ заданы (они называются *пропускными способностями* соответствующих ребер).

Аналогичное выражение $\sum_{(i,\ p')} f(i,\ p') - \sum_{(p',\ j)} f(p',\ j)$ для полюса p' не обязательно равно нулю. Оно называется *мощностью* потока. Поток с максимальной мощностью называется *максимальным,* и его построение является целью рассматриваемой задачи (обычно он определяется неоднозначно). К задаче о максимальном потоке сводятся многие прикладные задачи дискретной математики.

Другой пример не универсальных задач — это задачи построения календарных планов ранних и поздних сроков для сетевых моделей. Классическая сетевая модель процесса производства, проектирования или исследований состоит из событий $i = 1, 2, ..., n$ и условий логически временного характера, связывающих их сроки $T[k] \geqslant T[j] + t_{jk}$. Событие k может быть концом некоторой работы — подпроцесса рассматриваемого процесса, который в данной модели считается элементарным, j — ее началом, t_{jk} — длительностью; или же j — конец некоторой работы, а i — начало другой, которую можно выполнять только после окончания первой (в этом случае обычно $t_{jk}=0$). Пусть событие 1 (начало процесса) имеет срок $T[1]=0$. Ранние сроки $T[i]$ $(i = 1,\ 2,\ \ldots,\ n)$— это минимальные сроки остальных событий, выбранные так, чтобы удовлетворялись все сетевые ограничения — неравенства приведенного ранее типа из определяющего модель списка $\{j_\alpha, k_\alpha\}_1^M$.

Общая задача линейного программирования — среди систем значений n переменных $\{x_1, x_2, \ldots, x_n\}$ удовлетворяющих N линейным условиям— неравенствам и уравнениям:

$$\sum_{j=1}^{n} a_{ij}x_j \leqslant b_j, \qquad i = 1,\ 2,\ \ldots,\ M;$$

$$\sum_{j=1}^{n} a_{ij}x_j = b_j, \quad i = M+1,\ \ldots,\ N,$$

найти такую, что линейная форма $\sum_{j=1}^{n} c_j x_j$ примет максимальное значение.

Л. Г. Хачияну удалось найти алгоритм решения общей задачи линейного программирования со степенной трудоемкостью. Таким образом, эта задача не универсальна.

Отсюда следует, что не является универсальной *транспортная задача.* В ней требуется найти поставки $x_{ij} \geq 0$ $(i = 1,\ 2,\ \ldots,\ m,\ j = 1,\ 2,\ \ldots,\ n)$ продукта i-го поставщика j-му потребителю так, чтобы запросы всех потребителей

были удовлетворены: $\sum_{i=1}^{m} x_{ij} = b_j \quad (j = 1, 2, \ldots, n)$, каждый поставщик отправил бы не больше продукта, чем у него имеется $\sum_{j=1}^{n} x_{ij} \leqslant a_i \ (i = 1, 2, \ldots, m)$, а транспортные расходы $\sum_{i=1}^{m} \sum_{j=1}^{n} c_{ij} x_{ij}$ были минимальны.

Неизвестно, является ли универсальной задача об изоморфизме графов; однако доказана универсальность более общей задачи изоморфного вложения графа: установить, изоморфен ли граф G_1 какому-либо подграфу графа G_2.

1.5. Методы поиска решений на основе исчисления предикатов

Семантика исчисления предикатов обеспечивает основу для формализации логического вывода. Возможность логически выводить новые правильные выражения из набора истинных утверждений очень важна. Логически выведенные утверждения корректны, и они совместимы со всеми предыдущими интерпретациями первоначального набора выражений. Обсудим вышесказанное неформально и затем введем необходимую формализацию.

В исчислении высказываний основным объектом является переменное высказывание (предикат), истинность или ложность которого зависит от значений входящих в него переменных. Так, истинность предиката " x был физиком" зависит от значения переменной x. Если x - П. Капица, то предикат истинен, если x - М. Лермонтов, то он ложен. На языке исчисления предикатов утверждение $\forall x(P(x) \supset Q(x))$ читается так: "для любого x если P(x), то имеет место и Q(x) ". Иногда его записывают и так: $\forall x(P(x) \rightarrow Q(x))$. Выделенное подмножество тождественно истинных формул (или правильно построенных формул - ППФ), истинность которых не зависит от истинности входящих в них высказываний, называется аксиомами.

В исчислении предикатов имеется множество правил вывода. В качестве примера приведем классическое правило отделения, или modus ponens :

```
(A, A -> B) / B
```

которое читается так "если истинна формула А и истинно, что из А следует В, то истинна и формула В ". Формулы, находящиеся над чертой, называются посылками вывода, а под чертой - заключением. Это правило вывода формализует основной закон дедуктивных систем: из истинных посылок всегда следуют истинные заключения. Аксиомы и правила вывода исчисления предикатов первого порядка задают основу формальной дедуктивной системы, в которой происходит формализация схемы рассуждений в логическом программировании. Можно упомянуть и другие правила вывода.

Modus tollendo tollens : Если из А следует В и В ложно, то и А ложно.

Modus ponendo tollens : Если А и В не могут одновременно быть истинными и А истинно, то В ложно.

Modus tollendo ponens : Если либо А, либо В является истинным и А не истинно, то В истинно.

Решаемая задача представляется в виде утверждений (аксиом) f_1, F_2... F_n исчисления предикатов первого порядка. Цель задачи В также записывается в виде утверждения, справедливость которого следует установить или опровергнуть на основании аксиом и правил вывода формальной системы. Тогда решение задачи (достижение цели задачи) сводится к выяснению логического следования (выводимости) целевой формулы В из заданного множества формул (аксиом) f_1, F_2... F_n. Такое выяснение равносильно доказательству общезначимости (тождественно-истинности) формулы

f_1& F_2&... & F_n -> B

или невыполнимости (тождественно-ложности) формулы

$$f_1 \vee F_2 \vee \cdots \vee F_n \vee \neg B$$

Из практических соображений удобнее использовать доказательство от противного, то есть доказывать невыполнимость формулы. **На доказательстве от противного основано и ведущее правило вывода, используемое в логическом программировании, - принцип резолюции**. Робинсон открыл более сильное правило вывода, чем modus ponens, которое он назвал **принципом резолюции (или правилом резолюции)**. При использовании принципа резолюции формулы исчисления предикатов с помощью несложных преобразований приводятся к так называемой дизъюнктивной форме, то есть представляются в виде набора дизъюнктов. При этом под дизъюнктом понимается дизъюнкция литералов, каждый из которых является либо предикатом, либо отрицанием предиката.

Приведем пример дизъюнкта:

$$\forall x(P(x, c_1) \supset Q(x, c_2)).$$

Пусть P - предикат уважать, c_1 - Ключевский, Q - предикат знать, c_2 - история. Теперь данный дизъюнкт отражает факт "каждый, кто знает историю, уважает Ключевского".

Приведем еще один пример дизъюнкта:

$$\forall x(P(x, c_1)P(x, c_2)).$$

Пусть P - предикат знать, c_1 - физика, c_2 - история. Данный дизъюнкт отражает запрос "кто знает физику и историю одновременно".

Таким образом, условия решаемых задач (факты) и целевые утверждения задач (запросы) можно выразить в дизъюнктивной форме логики предикатов первого порядка. В дизъюнктах кванторы всеобщности $\forall, \exists$, обычно опускаются, а связки $\supset, \neg, \wedge$ заменяются на $\longrightarrow$ импликацию.

Вернемся к принципу резолюции. Главная идея этого правила вывода заключается в проверке того, содержит ли множество дизъюнктов R пустой (ложный) дизъюнкт. Обычно резолюция применяется с прямым или обратным методом рассуждения. Прямой метод из посылок A,

A → B выводит заключение B (правило modus ponens). Основной недостаток прямого метода состоит в его не направленности: повторное применение метода приводит к резкому росту промежуточных заключений, не связанных с целевым заключением. Обратный вывод является направленным: из желаемого заключения B и тех же посылок он выводит новое подцелевое заключение A. Каждый шаг вывода в этом случае связан всегда с первоначально поставленной целью. Существенный недостаток метода резолюции заключается в формировании на каждом шаге вывода множества резольвент - новых дизъюнктов, большинство из которых оказывается лишними. В связи с этим разработаны различные модификации принципа резолюции, использующие более эффективные стратегии поиска и различного рода ограничения на вид исходных дизъюнктов. **В этом смысле наиболее удачной и популярной является система ПРОЛОГ, которая использует специальные виды дизъюнктов, называемых дизъюнктами Хорна.**

Процесс доказательства методом резолюции (от обратного) состоит из следующих этапов:

1. Предложения или аксиомы приводятся к дизъюнктивной нормальной форме.
2. К набору аксиом добавляется отрицание доказываемого утверждения в дизъюнктивной форме.
3. Выполняется совместное разрешение этих дизъюнктов, в результате чего получаются новые основанные на них дизъюнктивные выражения (резольвенты).
4. Генерируется пустое выражение, означающее противоречие.
5. Подстановки, использованные для получения пустого выражения, свидетельствуют о том, что отрицание отрицания истинно.

Рассмотрим примеры применения методов поиска решений на основе исчисления предикатов. Пример "интересная жизнь". Итак, заданы утверждения 1-4 в левом столбце таблица 1 Требуется ответить на вопрос: "Существует ли человек, живущий интересной жизнью?" В виде предикатов эти утверждения записаны во втором столбце таблицы. Предполагается, что $\forall X(smart(X) = \neg stupid(X))$ и $\forall Y(wealthy(Y) = \neg poor(Y))$.
В третьем столбце таблицы записаны дизъюнкты.

Таблица1. Интересная жизнь

Утверждения и заключение	**Предикаты**	**Предложения(дизъюнкты)**
1. Все небедные и умные люди счастливы	$\forall X(\neg poor(X) \wedge sn$ $happy(X))$	$poor(X) \wedge \neg smart(X)$ $happy(X)$
2. Человек, читающий книги, - неглуп	$\forall Y(read(Y) \rightarrow sm$	$\neg read(Y) smart(Y)$
3. Джон умеет читать и является состоятельным человеком	$read(John) \wedge \neg poo$	$3a\ read(John)$ $3b\ \neg poor(John)$
4. Счастливые люди живут интересной жизнью	$\forall Z(happy(Z) \rightarrow ex$	$\neg happy(Z) \vee exciting(Z$
5. Заключение: Существует ли человек, живущий интересн	$\exists W(exciting(W))$	`exciting(W)`

ой жизнью?		
6. Отрицание заключения	$\neg\exists W(exciting(W)$	$\neg exciting(W)$

Отрицание заключения имеет вид (строка 6):

$$\neg\exists W(exciting(W))$$

Одно из возможных доказательств (их более одного) дает следующую последовательность резольвент:

1. $\neg happy(Z)$ резольвента 6 и 4
2. $poor(X) \wedge \neg smart(X)$ резольвента 7 и 1
3. $poor(Y) \wedge \neg read(Y)$ резольвента 8 и 2
4. $\neg read(John)$ резольвента 9 и 3b
5. NIL резольвента 10 и 3a

Символ NIL означает, что база данных выражений содержит противоречие и поэтому наше предположение, что не существует человек, живущий интересной жизнью, неверно.

В методе резолюции порядок комбинации дизъюнктивных выражений не устанавливался. Значит, для больших задач будет наблюдаться экспоненциальный рост числа возможных комбинаций. Поэтому в процедурах резолюции большое значение имеют также эвристики поиска и различные стратегии. Одна из самых простых и понятных стратегий - стратегия предпочтения единичного выражения, которая гарантирует, что резольвента будет меньше, чем наибольшее родительское выражение. Ведь в итоге мы должны получить выражение, не содержащее литералов вообще.

Среди других стратегий (поиск в ширину (breadth-first), стратегия "множества поддержки", стратегия линейной входной формы)

стратегия "множества поддержки" показывает отличные результаты при поиске в больших пространствах дизъюнктивных выражений. Суть стратегии такова. Для некоторого набора исходных дизъюнктивных выражений S можно указать подмножество T, называемое множеством поддержки. Для реализации этой стратегии необходимо, чтобы одна из резольвент в каждом опровержении имела предка из множества поддержки. Можно доказать, что если S - невыполнимый набор дизъюнктов, а S-T - выполнимый, то стратегия множества поддержки является полной в смысле опровержения. С другими стратегиями для поиска методом резолюции в больших пространствах дизъюнктивных выражений читатель может познакомиться в специальной литературе.

Исследования, связанные с доказательством теорем и разработкой алгоритмов опровержения резолюции, привели к развитию языка логического программирования PROLOG (Programming in Logic). PROLOG основан на теории предикатов первого порядка. Логическая программа - это набор спецификаций в рамках формальной логики. **Несмотря на то, что в настоящее время удельный вес языков LISP и PROLOG снизился и при решении задач ИИ все больше используются C, C++ и Java, однако многие задачи и разработка новых методов решения задач ИИ продолжают опираться на языки LISP и PROLOG.** Рассмотрим одну из таких задач - задачу планирования последовательности действий и ее решение на основе теории предикатов.

Задачи планирования последовательности действий

Многие результаты в области ИИ достигнуты при решении " задач для робота ". Одной из таких простых в постановке и интуитивно понятных **задач является задача планирования последовательности действий, или задача построения планов.**

В наших рассуждениях будут использованы примеры традиционной робототехники (современная робототехника во многом основывается на реактивном управлении, а не на планировании). Пункты плана определяют атомарные действия для робота. Однако при описании плана нет необходимости опускаться до микроуровня и говорить о датчиках, шаговых двигателях и т. п. Рассмотрим ряд предикатов, необходимых для работы планировщика из мира блоков. Имеется некоторый робот, являющийся подвижной рукой, способной брать и

перемещать кубики. Рука робота может выполнять следующие задания (U, V, W, X, Y, Z - переменные).

goto(X,Y,Z) перейти в местоположение X, Y, Z

pickup(W) взять блок W и держать его

putdown(W) опустить блок W в некоторой точке

stack(U,V) поместить блок U на верхнюю грань блока V

unstack(U,V) убрать блок U с верхней грани блока V

Состояния мира описываются следующим множеством предикатов и отношений между ними.

on(X,Y) блок X находится на верхней грани блока Y

clear(X) верхняя грань блока X пуста

gripping(X) захват робота удерживает блок X

gripping() захват робота пуст

ontable(W) блок W находится на столе

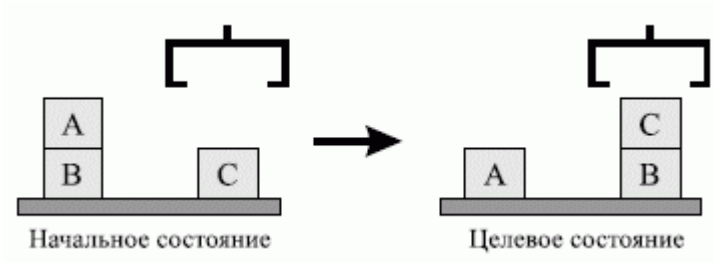


Рис. 1. Начальное и целевое состояния задачи из мира кубиков

Предметная область из мира кубиков представлена на рис. 1 в виде начального и целевого состояния для решения задачи планирования.

Требуется построить последовательность действий робота, ведущую (при ее реализации) к достижению целевого состояния.

Состояния мира кубиков представим в виде предикатов. Начальное состояние можно описать следующим образом:

```
start = [handempty, ontable(b),
         ontable(c), on(a,b), clear(c),
         clear(a)]
```

где: handempty означает, что рука робота Робби пуста.

Целевое состояние записывается так:

```
goal = [handempty, ontable(a),
        ontable(b), on(c,b), clear(a),
        clear(c)]
```

Теперь запишем правила, воздействующие на состояния и приводящие к новым состояниям.

$$(\forall X)(pickup(X) \rightarrow (gripping(X) \leftarrow (gripping() \wedge clear(X) \wedge ontable(X))))$$

$$(\forall X)(putdown(X) \rightarrow ((gripping() \wedge ontable(X) \wedge clear(X)) \leftarrow gripping(X)))$$

$$(\forall X)(\forall Y)(stack(X,Y) \rightarrow ((on(X,Y) \wedge gripping() \wedge clear(X)) \leftarrow (clear(Y) \wedge gripping(X))))$$

$$(\forall X)(\forall Y)(unstack(X,Y) \rightarrow ((clear(Y) \wedge gripping(X)) \leftarrow (on(X,Y) \wedge clear(X) \wedge gripping())))$$

Прежде чем использовать эти правила, необходимо упомянуть о проблеме границ. При выполнении некоторого действия могут изменяться другие предикаты и для этого могут использоваться аксиомы границ - правила, определяющие инвариантные предикаты. Одно из решений этой проблемы предложено в системе STRIPS.

В начале 1970-х годов в Стэнфордском исследовательском институте (Stanford Research Institute Planning System) была создана система STRIPS для управления роботом. В STRIPS четыре оператора pickup, putdown, stack, unstack описываются тройками элементов. Первый элемент тройки - множество предусловий (П), которым удовлетворяет мир до применения оператора. Второй элемент тройки - список дополнений (Д), которые являются результатом применения оператора. Третий элемент тройки - список вычеркиваний (В), состоящий из выражений, которые удаляются из описания состояния после применения оператора.

Ведя рассуждения для рассматриваемого примера от начального состояния, мы приходим к поиску в пространстве состояний. Требуемая последовательность действий (план достижения цели) будет следующей:

```
unstack(A,B), putdown(A), pickup(C), stack(C,B)
```

Для больших графов (сотни состояний) поиск следует проводить с использованием оценочных функций. Более подробно о работах по планированию, в том числе публикации по адаптивному планированию, можно прочитать в литературе .

В качестве заключения по данному разделу следует сказать, что планирование достижения цели можно рассматривать как поиск в пространстве состояний. Для нахождения пути из начального состояния к целевому (плана последовательности действий робота) могут применяться методы поиска в пространстве состояний с использованием исчисления предикатов.

Поиск решений в системах продукций

Поиск решений в системах продукций наталкивается на проблемы выбора правил из конфликтного множества. Различные варианты решения этой проблемы рассмотрим на примере ЭСО CLIPS. Правила

в ЭС, кроме фактора уверенности эксперта, имеют приоритет выполнения (salience). Конфликтное множество (КМ) - это список всех правил, имеющих удовлетворенные условия при некотором, текущем состоянии списка фактов и объектов и которые еще не были выполнены. Как отмечалось ранее, конфликтное множество это простейшая база целей. Когда активизируется новое правило с определенным приоритетом, оно помещается в список правил КМ ниже всех правил с большим приоритетом и выше всех правил с меньшим приоритетом. Правила с высшим приоритетом выполняются в первую очередь. Среди правил с одинаковым приоритетом используется определенная стратегия.

CLIPS поддерживает семь стратегий разрешения конфликтов.

Стратегия глубины (depth strategy) является стратегией по умолчанию (default strategy) в CLIPS. Только что активизированное правило помещается поверх всех правил с таким же приоритетом. Это позволяет реализовать поиск в глубину.

Стратегия ширины (breadth strategy) - только что активизированное правило помещается ниже всех правил с таким же приоритетом. Это, в свою очередь, реализует поиск в ширину.

LEX стратегия - активация правила, выполненная более новыми образцами (фактами), располагается перед активацией, осуществленной более поздними образцами. Например, как это указано в таблица 2 ниже.

MEA стратегия - сортировка образцов не производится, а осуществляется только упорядочение правил по первым образцам, как это показано в столбце 3 таблица 2.

Таблица 2. Результаты применения LEX и MEA стратегий

Исходный набор правил	**Правила, отсортированные LEX**	**Правила, отсортированные MEA**
rule-6: f-1,f-4	rule-6: f-4,f-1	rule-2: f-3,f-1
rule-5: f-1,f-2,f-3	rule-5: f-3,f-2,f-1	rule-3: f-2,f-1
rule-1: f-1,f-2,f-3	rule-1: f-3,f-2,f-1	rule-6: f-1,f-4
rule-2: f-3,f-1	rule-2: f-3,f-1	rule-5: f-1,f-2,f-3

rule-4: f-1,f-2	rule-4: f-2,f-1	rule-1: f-1,f-2,f-3
rule-3: f-2,f-1	rule-3: f-2,f-1	rule-4: f-1,f-2

Стратегия упрощения (simplicity strategy) - среди всех правил с одинаковым приоритетом только что активизированное правило располагается выше всех правил с равной или большей определенностью (specificity). Определенность правила задается количеством сопоставлений в левой части правил плюс количество вызовов функций. Логические функции не увеличивают определенность правила.

Стратегия усложнения (complexity strategy) - среди всех правил с одинаковым приоритетом только что активизированное правило располагается выше всех правил с равной или большей определенностью.

Случайная стратегия (random strategy) - каждой активации назначается случайное число, которое используется для определения местоположения среди активаций с определенным приоритетом.

Подход на основе стратегий поиска решений в продукционных ЭС известен достаточно давно. Весьма популярная в начале 90-х годов ЭСО GURU (ИНТЕР-ЭКСПЕРТ) также использовала подобные механизмы управления стратегиями поиска. Возможность смены стратегии в ходе решения задачи программным образом и накопление опыта, какие стратегии дают лучшие результаты для определенных классов задач, позволяет получить эффективные механизмы поиска решений в СПЗ на основе продукций.

Следует отметить, что существуют различные методы поиска решений в семантических сетях, например, метод обхода семантической сети - мультипарсинг. Данный метод оригинален тем, что позволяет параллельно "вести" по графу несколько маркеров и, тем самым, распараллеливать процесс поиска информации в семантической сети, что увеличивает скорость поиска. Эти методы используются, как правило, при представлении текста в виде объектно-ориентированной семантической сети и в данной лекции не рассматриваются.

Поиск в сетях фреймов, основанный на прецедентах вывод (Case-based Reasoning - CBR), правдоподобные рассуждения (plausible reasoning),

методы поиска на основе нечеткой логики и другие методы поиска решений ИИ в данной работе не рассматриваются.

1.6. Эвристический алгоритм определения физических закономерностей по результатам наблюдений.

1.6.1. Введение. Постановка задачи

Многие проблемы, связанные с исследованием явлений окружающей нас действительности, сводятся к задаче определения неизвестных функциональных зависимостей между характеризующими эти явления величинами. Такие зависимости представляют собой физические закономерности (законы природы), описывающие происходящие в окружающей нас действительности процессы, если мы изучаем природные явления, или математические модели в случае исследования различного рода естественных или искусственно созданных систем.

Интересующие нас функциональные зависимости могут быть определены в результате анализа данных, полученных при проведении специальных экспериментов, или в процессе наблюдения изучаемых явлений в ходе их естественного развития.

Существует **два подхода** к решению задачи исследования зависимостей. **При первом, характерном для регрессионного анализа, переменные разделяются на две группы : результирующие (прогнозируемые) и объясняющие (предикторные).** Затем при соответствующих допущениях определяются зависимости первых от вторых. Неудобство такого подхода обусловлено тем, что для подобного разделения часто нет достаточных оснований. Особенно тогда, когда исследуются слабо изученные объекты. В этом случае к числу прогнозируемых переменных случайно могут быть отнесены такие, которые на самом деле не зависят от остальных. Полученные при этом результаты анализа не будут иметь никакой ценности: они не могут быть интерпретированы, так как за ними нет никакой существующей зависимости.

Более привлекательным в свете сказанного представляется подход, принятый в факторном анализе, когда все переменные считаются равноценными. Но **модель классического факторного анализа, основанная на представлении величин в виде линейной комбинации факторов с соответствующими факторными нагрузками**, предоставляет исследователю значительно меньше возможностей, чем более гибкие нелинейные модели регрессионного анализа.

В настоящей работе предпринята попытка использовать достоинства упомянутых подходов и отказаться от некоторых допущений, принятых при решении задачи исследования зависимостей: разделения переменных на объясняющие и результирующие, а также предположений о том, что число характеризующих исследуемое явление параметров, число зависимостей, связывающих между собой наблюдаемые величины, и степень алгебраического полинома,определяющего вид функции регрессии, известны.

Прежде чем приступить к формулировке задачи сделаем одно замечание. **Зависимости между параметрами, характеризующими явления окружающей нас действительности, могут иметь вероятностный или детерминированный характер**. В случае вероятностной зависимости при осуществлении события, заключающегося в том, что часть параметров принимает какое-то определенное значение, величины остальных параметров могут быть указаны лишь с некоторой вероятностью. **При детерминированной - заданием части параметров значения остальных определяются однозначно.**

В приведенной ниже формулировке задачи исследуемые зависимости между параметрами будут считаться детерминированными. Отметим, что **такие зависимости характерны для многих фундаментальных законов природы. Со случайными связями параметров мы часто сталкиваемся только потому, что при исследованиях в силу различных обстоятельств не учитываем факторов, влияющих на изучаемое явление и имеющих случайный характер. Измерение этих факторов и включение их в состав вектора наблюдаемых величин сводит задачу к случаю детерминированных зависимостей.**

Введем в рассмотрение два вектора **: вектор наблюдаемых** в **экспериментах величин**, представляющих собой измеряемые параметры исследуемого явления

$\mathbf{Y_k} = | y_1, y_2, \ldots, y_k |^T$,
и **вектор характеризующих исследуемое явление параметров**
$\mathbf{X_n} = | x_1, x_2, \ldots, x_n |^T$.

Состав вектора $\mathbf{Y_k}$ определяется структурой измерительного комплекса, который фиксирует в общем случае с ошибками реализовавшиеся в экспериментах параметры исследуемого явления, то есть компоненты вектора $\mathbf{X_n}$.

Приступая к изучению явления, мы, как правило, не знаем какими параметрами это явление характеризуется. Нам обычно не известна размерность **n** вектора параметров $\mathbf{X_n}$. Часто не бывает ясна даже физическая сущность всех величин, входящих в его состав. Поэтому вектор наблюдаемых параметров $\mathbf{Y_k}$ по своему составу, как правило, отличается от вектора параметров $\mathbf{X_n}$.

Эти отличия могут быть сведены к **двум случаям**. **В первом** среди составляющих вектора наблюдаемых параметров $\mathbf{Y_k}$ могут присутствовать такие, которые не имеют аналога среди компонент вектора $\mathbf{X_n}$. Они не являются измеренными значениями параметров исследуемого явления. В состав вектора $\mathbf{Y_k}$ они оказываются включенными случайно из-за недостаточности знаний об изучаемом явлении и выступают в роли мешающих факторов (помех), увеличивая размерность задачи и повышая случайные ошибки результатов определения зависимостей между параметрами $\mathbf{X_n}$.

Во втором - в составе вектора $\mathbf{Y_k}$ могут отсутствовать компоненты, соответствующие некоторым составляющим вектора $\mathbf{X_n}$. Так случается, когда часть определяющих физическое явление параметров также из-за недостаточности знаний об изучаемом явлении не наблюдается. В этой ситуации закономерность, связывающая между собой параметры изучаемого явления, найдена быть не может. Чтобы ее можно было определить, необходимо расширить возможности системы наблюдения исследуемого явления, включив в ее состав дополнительные измерительные устройства. Такие и столько, чтобы все характеризующие явление параметры были бы измерены. Эта

задача может быть решена, по-видимому, только методом проб и ошибок с привлечением экспертных оценок состава вектора $\mathbf{X_n}$

Задачу определения неизвестных функциональных зависимостей между параметрами изучаемого явления сформулируем следующим образом **: по результатам измерений характеризующих изучаемое явление параметров, представленных величинами**

$$y^{*}_{ij} = y_{ij} + \Delta y_{ij}$$

$$(i=1,2,..,N;\ j=1,2,..,k),$$

где N - число наблюдений (экспериментов), k - число измеряемых в каждом наблюдении величин, а Δy_{ij}- ошибки измерения, построить такую неявно заданную векторнозначную функцию

$$f(x_1, x_2, .., x_n) = \begin{cases} f_1(x_1, x_2, .., x_n) = 0, \\ f_2(x_1, x_2, .., x_n) = 0, \\ \cdot \quad\quad \cdot\ \cdot \\ \cdot \quad\quad \cdot\ \cdot \\ \cdot \quad\quad \cdot\ \cdot \\ f_L(x_1, x_2, .., x_n) = 0, \end{cases} \qquad (1)$$

которая наилучшим в определенном смысле образом восстанавливала бы неизвестные зависимости между параметрами $x_1, x_2, \ldots, x_n$ исследуемого явления.

В приведенной формулировке наблюдаемые параметры не разделяются на объясняющие и результирующие. Интересующая нас зависимость (1) представлена в виде неявно заданной векторнозначной функции составляющих вектора $\mathbf{X_n}$. Такое представление позволяет учесть при анализе все зависимости, которыми, возможно, связаны между собой различные наблюдаемые величины, входящие в состав $\mathbf{X_n}$. Число этих зависимостей может равняться нулю, что соответствует случаю отсутствия связей между параметрами, единице, двум и т.д. Обозначим его через **L**. Число **L** в рассматриваемой задаче будем считать неизвестным.

1.6.2. Основные обозначения, предварительные преобразования

Предположим, что вид исследуемых функциональных зависимостей (1) известен. Неизвестными является только значения векторов параметров **C** .

$$f_l(x_1, x_2, .., x_n) = f_l(x_1, x_2, .., x_n, \bar{c}_l) = 0, \qquad (2)$$

$$\bar{c}_l = | c_{1l}, c_{2l}, .., c_{pl} |^T,$$

$$(l = 1, 2, .., L).$$

Многие гладкие функции могут быть представлены в виде алгебраического полинома относительно переменных:

$\mathbf{X_n} = | \mathbf{x_1}, \mathbf{x_2}, .., \mathbf{x_n} |^T$:

$$f_l(x_1, x_2, .., x_n, \bar{c}_l) = c_{0l} + \sum_{p=1}^{n} c_{pl} \cdot x_p + \sum_{p_1=1}^{n} \sum_{p_2=1}^{n} c_{p_1 p_2, l} \cdot x_{p_1} \cdot x_{p_2} +$$

$$+ .... + \sum_{p_1=1}^{n} ... \sum_{p_m=1}^{n} c_{p_1 ... p_m, l} \cdot x_{p_1} \cdot ... \cdot x_{p_m} = \qquad (3)$$

$$= \sum_{q=1}^{P} c_{q,l} d_q(x_1, x_2, .., x_n),$$

содержащего свободный член, линейные, квадратичные и т.д. слагаемые. Число **P** членов полиномов (3) зависит от числа **n** параметров функций (1) и степени разложения этих функций в ряд. В дальнейшем будем считать, что все степени $\mathbf{m_l}$ **(l = 1,..,L)** разложения функций (2) равны их максимальному значению **m : m = Max ($\mathbf{m_1}$,$\mathbf{m_2}$,..,$\mathbf{m_L}$).** Представление функций отклика (1) в виде полинома (3), линейного относительно неизвестных параметров $\mathbf{c_q}$ **(q = 1,2,..,P)** и нелинейного относительно аргументов $\mathbf{x_p}$ **(p = 1,2,..,n)**, удобно для решения задачи исследования зависимостей между параметрами, характеризующими природные явления. **Оно позволяет использовать для решения задачи хорошо разработанный аппарат регрессионного и конфлюэнтного анализа.** Удобство представления (3) объясняется также тем, что многие физические законы

записываются с помощью соотношений, представляющих собой его частные случаи. Это свойство является следствием того, что функциональные зависимости, выражающие физические факты, не зависящие от единиц измерения, обладают специальной структурой. Особенности структуры соотношений, с помощью которых записываются физические закономерности, аналитически определяются формулой размерности, утверждающей , что размерности физических величин имеют вид степенных полиномов, а также вытекающей из этой формулы **П**-теоремы теории подобия.

Введем в рассмотрение матрицу **"D"** размерности **P * P :**

$$D = \begin{Vmatrix} 1, x_{11}, \ldots, x_{1n}, x_{11}^2, x_{11} \cdot x_{12}, \ldots, x_{1n}^2, x_{11}^3, \ldots, x_{1n}^m \\ 1, x_{21}, \ldots, x_{2n}, x_{21}^2, x_{21} \cdot x_{22}, \ldots, x_{2n}^2, x_{21}^3, \ldots, x_{2n}^m \\ \cdot \quad \cdot \quad \cdot \quad \cdot \quad \cdot \quad \cdot \quad \cdot \quad \cdot \quad \cdot \quad \cdot \quad \cdot \\ \cdot \quad \cdot \quad \cdot \quad \cdot \quad \cdot \quad \cdot \quad \cdot \quad \cdot \quad \cdot \quad \cdot \quad \cdot \\ 1, x_{p1}, \ldots, x_{pn}, x_{p1}^2, x_{p1} \cdot x_{p2}, \ldots, x_{pn}^2, x_{p1}^3, \ldots, x_{pn}^m \end{Vmatrix} \quad (4)$$

$$m = \text{Max} (m_1, m_2, .., m_L).$$

Матрица **D**, аналогичная матрице плана в теории планирования экспериментов [6], обладает рядом полезных для нас свойств. Одно из них позволяет сформулировать критерий существования связи между параметрами исследуемого явления. Сформулируем его в виде **Утверждения 1**, справедливость которого легко может быть доказана.

1.6.3. Утверждение 1. Критерий существования зависимости между параметрами исследуемого явления

Утверждение 1. Если наблюдаемые параметры

$x_1, x_2, .., x_n$ связаны между собой **L** функциональными

зависимостями, каждая из которых представлена

соотношением (3), ранг матрицы **D** равен **(P - L).**

Из сказанного следует, что в качестве критерия, позволяющего определить существование **L** функциональных связей между составляющими вектора **x** , может быть использовано условие отличия от нуля по крайней мере одного определителя матрицы **(P - L)**-го порядка, полученной из матрицы **D** удалением **L** столбцов и **L** строк , при условии равенства нулю всех определителей более высокого порядка. Действие ***Утверждения 1.*** распространяется на случай систем с обратной связью (замкнутых систем), на ситуации, когда часть параметров являются константами или представляют собой результат измерения разными средствами одной и той же величины.

Сформулированный критерий позволяет устанавливать наличие связей между измеряемыми параметрами исследуемого явления $y_1,...,y_k$. Эти свзи при условии измерения всех характеризующих явление параметров будут представлять собой интересующие нас зависимости между параметрами $x_1,...,x_n$ исследуемого явления. Критерий позволяет также выделить линейно зависимые столбцы матрицы **D**, если связи между параметрами существуют. Для этого нужно при наличии одной связи, последовательно удаляя столбцы матрицы **D** , определять ранг полученных таким образом матриц размерности **(P - 1) * (P - 1)**. Если зависимостей несколько, то нужно удалять группы столбцов : пары, тройки и т.д.

1.6.4. Параметры зависимостей, оцениваемые в результате анализа

Параметрами зависимостей, описывающих исследуемое явление, которые должны быть определены в конечном итоге, ***являются коэффициенты разложения (3)***

Если зависимости между параметрами, характеризующими исследуемое явление, существуют и степень разложения функций (2) выбрана достаточной, то коэффициенты разложения (3) удовлетворяют системе **P** линейных однородных уравнений :

$$\sum_{q=1}^{P} c_q \cdot d_{iq}\,(x_1, x_2, \ldots, x_n) = 0, \quad (i = 1,2,..,P). \qquad (5)$$

В векторном виде система (5) может быть записана следующим образом :

$$D \cdot \bar{C} = 0, \qquad (6)$$

где $\mathbf{C} = |\, c_1, c_2, .., c_p \,|^T$, а **D** - квадратная **P** * **P** матрица (4).

Система однородных линейных уравнений (6) имеет отличное от тривиального $c_q = 0$ (q = 1,.., **P**) решение, если определитель **detD** равен нулю, то есть когда между столбцами матрицы **D** существует линейная зависимость.

Если параметры $x_1, x_2, .., x_n$ связаны между собой только одной зависимостью **(L=1)**, ранг матрицы **D** равен **P-1** и равен рангу квадратных **(P - 1) * (P - 1)** матриц $D_\varphi^{(r)}$ **(** $\varphi = q_1^{(r)}, q_2^{(r)}, .., q_r^{(r)}$ **)**, полученных из **D** удалением любой (номер ее обозначим через **s , s = 1,2,..,P**) строки и одного из **r** линейно зависимых столбцов **d** . Здесь $q_1^{(\varphi)}, q_2^{(\varphi)}, .., q_r^{(\varphi)}$ **)** - номера линейейно зависимых столбцов матрицы **D** , а **r** - их число.

Рассмотрим системы неоднородных линейных уравнений:

$$D_\varphi^{(z)} \cdot \bar{c}_\varphi^{(z)} = -\bar{d}_\varphi^{(z)}, \qquad (7)$$

$$(\varphi = q_1^{(z)}, q_2^{(z)}, .., q_z^{(z)};\ 2 \le r \le P),$$

где $\bar{c}_\varphi^{(z)} = |\, c_1, c_2, \ldots, c_{\varphi-1}, c_{\varphi+1}, \ldots, c_P \,|^T$,

$\bar{d}_\varphi^{(z)} = |\, d_{1\varphi}, d_{2\varphi}, \ldots, d_{s-1,\varphi}, d_{s+1,\varphi}, \ldots, d_{P\varphi} \,|$.

Все **r** систем (7) имеют единственное решение, так как матрицы **D** и их расширенные матрицы имеют один и тот же ранг.

1.6.5. Утверждение 2

Утверждение 2. Решения всех систем линейных

неоднородных уравнений (7), соответствующих различным

значениям $\varphi = q_1^{(r)}, q_2^{(r)},..., q_r^{(r)}$, тождественны. Все они

определяют одну и ту же зависимость между

исследуемого физического явления.

Справедливость утверждения легко доказывается, а само оно без труда обобщается на случай, когда параметры связаны между собой несколькими зависимостями. В этом случае ранг матрицы **D** равен **P-L** и равен рангу квадратных **(P-L)*(P-L)** матриц , полученных из **D** удаением любых **L** строк и **L** столбцов , каждый из которых относится к своей группе линейно зависимых столбцов матрицы **D.**

Из сказанного следует, что если зависимости между хактеризующими явление параметрами существуют и степень разложения функций (3) выбрана достаточной , параметры описывающих исследуемое явление зависимостей являются решением одной из систем (любой) неоднородных линейных уравнений (7). Все эти решения тождественны и определяют одни и те же зависимости между параметрами $x_1, x_2, ..., x_n$ исследуемого физического явления.

Неоднородные уравнения (7) могут быть получены из исходной системы (6) в процессе проверки существования связи между параметрами исследуемого явления с помощью критерия, приведенного в предыдущем разделе.

1.6.6. Структура системы обработки информации. Алгоритм определения зависимостей между параметрами исследуемого явления.

Результаты двух предыдущих разделов позволяют определить состав, структуру и алгоритмы системы обработки информации, с помощью которой могут быть установлены зависимости между параметрами изучаемого физического явления по результатам наблюдений.

Представим результаты измерений параметров исследуемого явления в виде прямоугольной **N * k** таблицы (матрицы) **A (y^*)** измеренных величин.

$$A(y^*) = \begin{Vmatrix} y^*_{11}, & y^*_{12}, & \dots, & y^*_{1k} \\ y^*_{21}, & y^*_{22}, & \dots, & y^*_{2k} \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ y^*_{N1}, & y^*_{N2}, & \dots, & y^*_{Nk} \end{Vmatrix} . \qquad (8)$$

Каждый элемент этой матрицы y_{ij}^* (i =1,..., **N**; j =1,..., **k**) представляет собой измеренное в общем случае с ошибкой значение реализовавшегося в **i**-том эксперименте **j**-того наблюдаемого параметра $y_{i,j}$.

Один из вариантов структурной схемы системы определения зависимостей на основании результатов экспериментов приведен на ***Рисунке 1***. В состав системы входят: комплекс наблюдения исследуемого явления, база данных для накопления и хранения результатов измерений, база знаний для размещения результатов анализа и ряд алгоритмов анализа данных. В том числе алгоритм проверки существования и определения числа зависимостей между наблюдаемыми параметрами и алгоритм оценки параметров связей между наблюдаемыми параметрами y_1 ,.., y_k .

Проверка существования зависимостей между параметрами сводится к проверкам статистическох гипотез о равенстве нулю значений определителей $\mathbf{d\,e\,t\,D_{p-l}\,(\,y\,)\quad(\,l=0\,,1\,,...,\,L\,,\,L+1\,)}$ матриц

$$\mathbf{D_{p-l}\,(\,y\,)\;=\;matrix\,(\;d_{p-l,i,j}\,(\,y_{i1},\,y_{i2}\,,\,...\,,\,y_{ik})\,)}$$
$$\mathbf{(\,i=1\,,...\,,\,p\,;\,j=1\,,...\,,\,p\,;\,l=0\,,1\,,..\,,\,L\,,\,L=1\,)}$$

на основании информации, представленной в виде таблицы $\mathbf{A\,(\,y^{*})}$ (8) , или гипотез о принадлежности к одной генеральной совокупности значений определителей : соответствующих исходным матрицам

$$\mathbf{D_{p-l}\,(\,y^{*}\,)\;=\;matrix\,(\,d_{p-l,i,j}\;(\,y^{*}_{i1},\,y^{*}_{i2}\,,\,...\,,\,y^{*}_{ik})\,)}$$
$$\mathbf{(\,i=1\,,...\,,\,p\,;\,j=1\,,...\,,\,p\,;\,l=0\,,1\,,..\,,\,L\,,\,L=1\,)}$$

и матрицам, полученным из них путем принудительного разрушения причинно-следственных связей между элементами их столбцов. В процессе этих проверок определяется также число существующих зависимостей **L** между наблюдаемыми величинами и формируются неоднородные уравнения (7), которым удовлетворяют интересующие нас параметры исследуемых зависимостей.

Кроме упомянутых, в состав системы целесообразно включить комплекс алгоритмов анализа возможностей снижения размерности задачи на двух уровнях: на уровне матрицы наблюдений **A** и на уровне матрицы **D**. Эти алгоритмы призваны устранить столбцы матрицы **A**, не участвующие в формировании линейно зависимых столбцов матрицы **D**, и матрицы **D**, не входящие в число таких столбцов, и снизить тем самым размерность задачи. На завершающей стадии анализа, если в поле зрения исследователя оказалось несколько зависимостей, целесообразно провести декомпозицию ситуации , выделив в виде отдельной задачи анализ каждой мз зависимостей.

Если степень аппроксимирующего полинома в процессе анализа будет выбрана большей, чем это необходимо для представления исследуемого соотношения, среди решений появяться "вторичные" завмсимости. Например, при действительной связи между параметрами $\mathbf{x_1 = x_2 + x_3}$ и степени полинома равной двум среди решений, кроме основного, появится "вторичное" $\mathbf{x_1^2 = (\,x_2 + x_3\,)^2}$. Такие решения должны быть идентифицированы с основными и удалены, а степень полинома соответствующим образом скорректирована.

Обратим внимание на следующее обстоятельство: невыполнение условий $\mathbf{detD_{p-1}(y) = 0}$, проверка которых была осуществлена по информации, представленной в виде матрицы (8), еще не означает, что параметры $\mathbf{x_1, ..., x_n}$ не связаны функционально между собой. Такое событие может наступить и при наличии зависимости, если степень аппроксимирующего полинома (3) недостаточна для представления исследуемого соотношения, а также в случае, когда не все параметры исследуемого явления наблюдаются.

Таблица 1.

Плотность распределения detD.

G(i)\N№	1	2	3	4
G (1)	112	135	246	174
. . .	.	.	.	.
G (40)	0	9	3	2
G (41)	3	10	9	7
G (42)	5	8	4	7
G (43)	8	14	8	9
G (44)	9	18	6	11
G (45)	17	6	13	12
G (46)	12	10	12	9
G (47)	20	16	14	16
G (48)	26	20	13	21
G (49)	35	32	30	40
G (50)	146	125	36	68
G (51)	166	136	32	79
G (52)	42	39	20	28
G (53)	22	23	11	21
G (54)	16	17	10	16
G (55)	15	12	7	13
G (56)	16	10	6	14
G (57)	12	13	8	5
G (58)	9	10	10	4
G (59)	8	14	4	9
G (60)	7	10	9	6
. . .	.	.	.	.
G(100)	114	107	268	196

В этой ситуации для нахождения решения необходимо сначала повышать степень полинома , а если это не приводит к успеху,

увеличивать число наблюдаемых параметров изучаемого явления , используя для принятия решения о включении в состав измерительного комплекса дополнительных измерителей экспертные оценки состава вектора параметров, характеризующих изучаемое явление

Для разработки алгоритмов проверки упомянутых выше гипотез необходимо знать условные законы распределения определителей матриц $\mathbf{D}_{p\text{-}l}(\mathbf{y}^*)$.

Вид этих законов зависит от ряда факторов: числа наблюдаемых параметров, степени разложения функции отклика (3), числа экспериментов, уровня и законов распределения ошибок измерения, распределения наблюдаемых параметров. Исследование их в общем случае представляет собой достаточно сложную задачу.

В таблице 1 и на рисунке 2 приведены результаты такого исследования для частного случая: полученные методом Монте-Карло при числе циклов моделирования 1000 гистограммы распределения определителя матрицы **D** размерности (15x15), что соответствуют числу наблюдаемых параметров равному четырем и второй степени аппроксимирующего полинома, количеству экспериментов **N** равному 15, аддитивных ошибках измерения уровня 5% от наблюдаемых величин и равномерном распределении измеряемых параметров.

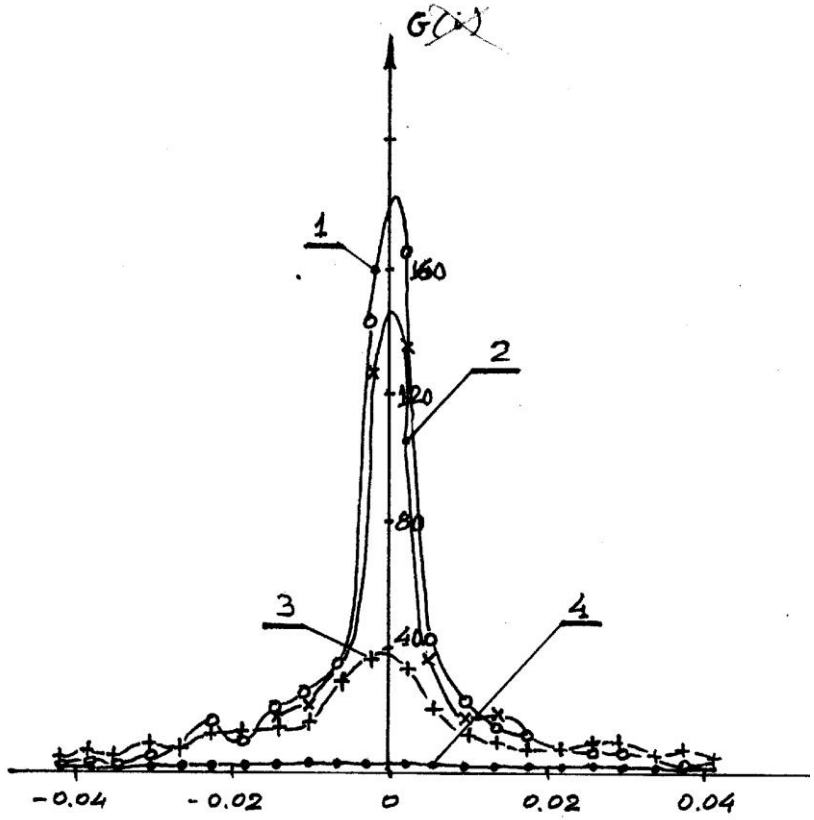


Рис.2. Плотность распределения detD.

Кривая **N 1** на **рис. 2** и колонка **N 1 таблицы 1** соответствуют случаю, когда функциональная зависимость между элементами столбцов матрицы **A** и, следовательно, линейная зависимость между элементами столбцов матрицы **D** существует. Кривые **N 2, 3, 4** и соответствующие им колонки **N 2, 3, 4** таблицы относятся к ситуациям, когда такие связи принудительно разрушены.

Результаты моделирования показывают, что графики плотности распределения значений определителя detD имеют колоколообразную

форму с сильно вытянутыми в обеих направлениях вдоль оси **X** ветвями. Разрушение связей между элементами матриц **A** и **D** заметно изменяют вид законов распределения определителя матрицы **D**. Такое изменение более существенно при разрушении связей между элементами матрицы **A** (кривая **N 4** существенно отличается от кривой **N 1**). Кривые **N 2** и **3** , соответствующие случаю разрыва связей между элементами столбцов матрицы **D**, отличаются от кривой **N 1** в меньшей степени.

Приведенные результаты моделирования позволяют сделать вывод о возможности создания основанного на предложенных принципах работоспособного алгоритма определения физических закономерностей по результатам наблюдений.

1.6.7 .Пример: "Открытие" второго закона Ньютона.

Рассмотрим задачу определения по результатам наблюдений зависимости между ускорением тела, его массой и действующей на тело силой (закона Ньютона). Вектор наблюдаемых параметров **Y**, характеризующих изучаемое явление, будем считать состоящим из четырех элементов: $y_1 = m$ и $y_2 = a$ - соответственно массы и ускорения тела, y_3 - случайно попавшей в поле зрения исследователя величины, не имеющей отношения к анализируемой ситуации (помехи), и $y_4 = F$ - действующей на тело силы.

Вектор **X** характеризующих исследуемое явление параметров в рассматриваемом случае содержит три координаты: $x_1 = m$, $x_2 = a$, $x_3 = F$. Соотношения между составляющими векторов y и x определяются, таким образом, следующими зависимостями **:**

$$y_1 = x_1 , \; y_2 = x_2 , \; y_4 = x_3 .$$

Степень аппроксимирующего полинома зададим равной двум. Число столбцов матрицы **A** в нашем случае будет равно четырем, матрицы **D** - пятнадцати. Параметрам y_1 - y_3 при моделировании будем придавать случайные независимые между собой значения. Величина y_4 , естественно, всегда будет равна произведению y_1 на y_2 . Вносимые в процессе измерения ошибки будем менять в пределах от нуля до пяти процентов от измеряемых величин.

Результаты оценки по 100 реализациям значений вторых начальных моментов определителя матрицы **D** при различных уровнях ошибок измерения (первая строка), а также этих моментов после замены столбцов матриц **A** (второй блок таблицы) и **D** (третий блок) столбцами с элементами, распределенными в соответствии с теми же законами, но независимыми от элементов столбцов исходных матриц, приведены в **таблице 2.**

Таблица 2.

Значения вторых начальных моментов
распределения определителя матрицы D.

Ош. изм.	0.00001%	1%	2%	3%	5%
ГdetD	7.22E-17	2.08E-01	3.55E-01	1.97E-01	3.42E+00
ГdetDa 1	9.40E+01	4.43E+02	1.55E+03	5.94E+02	1.89E+03
ГdetDa 2	4.31E+05	1.08E+06	2.58E+06	4.21E+06	7.26E+06
ГdetDa 3	1.64E-16	1.64E+00	1.66E+00	1.06E+00	2.83E+00
ГdetDa 4	5.01E+06	1.10E+06	3.46E+06	9.90E+05	6.12E+06
ГdetDd 0	7.22E-17	2.08E-01	3.55E-01	1.97E-01	3.42E+00
ГdetDd 1	2.97E-15	9.33E+00	6.17E+00	1.66E+01	6.14E+01
ГdetDd 2	7.24E-15	4.03E+00	5.83E+00	2.81E+00	2.83E+02
ГdetDd 3	3.08E-15	3.93E-01	1.95E+00	1.01E+00	2.75E+00
ГdetDd 4	1.48E+02	5.72E+02	2.94E+02	9.53E+01	3.51E+02
ГdetDd 5	3.64E-15	5.89E+00	9.63E+00	2.58E+00	3.51E+01
ГdetDd 6	2.47E+01	9.56E+02	8.20E+02	2.08E+02	8.68E+02
ГdetDd 7	5.01E-16	3.53E-01	1.05E+00	8.56E-01	1.41E+01
ГdetDd 8	4.57E-14	6.95E+00	1.20E+01	9.24E+00	1.11E+02
ГdetDd 9	4.69E-15	1.92E+00	5.37E+00	5.72E+00	4.71E+01
ГdetDd10	1.65E-15	2.66E-01	1.08E+00	2.29E+00	4.51E+01
ГdetDd11	1.07E-14	1.52E+01	8.31E+01	2.67E+01	1.04E+02
ГdetDd12	2.46E-16	3.02E-02	1.55E-01	7.09E-02	3.51E+00
ГdetDd13	8.21E-15	1.72E+00	3.56E+00	2.51E+00	3.38E+01
ГdetDd14	1.17E-13	2.24E+01	3.29E+01	2.40E+01	1.68E+02

Приведенные числовые результаты показывают, что четыре наблюдаемых параметра y_1 , y_2 , y_3 и y_4 связаны между собой только одной зависимостью (ранг матрицы **D**, состоящей из 15 строк и 15 столбцов, равен 14) и что среди столбцов матрицы **D** есть только два линейно зависимых (**r = 2**) **:** с номерами **4** и **6**. Им соответствуют две системы линейных неоднородных уравнений (7), которые имеют решения :

$$\bar{c}_1^{(2)} = \begin{cases} 1 & \text{при } q = 4, \\ 0 & \text{при } q \neq 4, \end{cases}$$

$$\bar{c}_2^{(2)} = \begin{cases} 1 & \text{при } q = 6, \\ 0 & \text{при } q \neq 6, \end{cases}$$

$$(q = 0, 1, \ldots, 14),$$

определяющие одну и ту же зависимость между элементами столбцов матрицы **D :**

$\mathbf{d_4 = d_6}$

что соответствует зависимости между характеризующими явление параметрами $\mathbf{x_i}$ **(i = 1, 2, 3) :**

F = m a. $\quad (\mathbf{x_3 = x_1 x_2})$ **.**

Заметим, что размерность задачи до решения систем уравнений (до этапа решения задачи регрессионного анализа) могла быть снижена: столбец $\mathbf{a_3}$ матрицы **A** и все столбцы матрицы **D**, кроме $\mathbf{d_4}$ и $\mathbf{d_6}$, можно было удалить и тем самым упростить задачу.

Несколько расширим задачу, включив в число наблюдаемых величин параметры, характеризующие движение тела массы **m** под действием силы **F :** его положение и скорость в два момента времени $\mathbf{t_0}$ и **t .**

Вектор наблюдаемых параметров при этом кроме $y_1 - y_4$ будет включать еще следующие элементы **:**
$y_5 = X_0$, $y_6 = V_0$ - начальные значения положения и скорости тела в момент времени t_0 , $y_7 = X$, $y_8 = V$ - значения положения и скорости в момент t , $y_9 = t_0$, $y_{10} = t$.

Число столбцов матрицы **A** теперь будет равно 10-и, а матрицы **D** при степени аппроксимирующего полинома равной трем - 286-и. Элементы столбцов матрицы **D** в рассматриваемом случае оказываются связанными между собой ткими тремя соотношениями:

Число столбцов матрицы **A** теперь будет равно 10-и, а матрицы **D** при степени аппроксимирующего полинома равной трем - 286-и. Элементы столбцов матрицы **D** в рассматриваемом случае оказываются связанными между собой ткими тремя соотношениями:

$$\begin{aligned} F &= m \cdot a , \\ V &= V_o + a \cdot (t - t_o), \\ X &= X_o + V_o \cdot (t - t_o) + 0.5 \cdot a \cdot (t - t_o)^2 . \end{aligned} \qquad (9)$$

Кроме этих между элементами столбцов матрицы D существуют еще "вторичные" зависимости. Все они тождественны соотношениям (9).

1.6.8. Пример:"Доказательство" теоремы Пифагора

Теорема Пифагора в геометрических терминах формулируется следующим образом:

Площадь квадрата, построенного на гипотенузе прямоугольного треугольника, равна сумме площадей квадратов, построенных на его катетах.

Эту теорему можно сформулировать, используя алгебраическую форму записи :

$$a^2 + b^2 = c^2,$$

где a,b – катеты прямоугольного треугольника, а c – его гипотенуза.

Впервые с доказательством теоремы Пифагора мы сталкиваемся в средней школе . В статье “Теорема Пифагора”(Материал из Википедии — свободной энциклопедии) сказано, что на данный момент в научной литературе зафиксировано 367 доказательств этой теоремы. Известны доказательства Евклида, Леонардо да Винчи, самого Пифагора. В них используются самые разнообразные подходы : геометрический (через подобные треугольники , с помощью метода площадей), основанный на анализе бесконечно малых , векторный и т.д. и т.д.

Справедливость выражения (1) может быть доказана также с помощью метода, основанного на анализе результатов наблюдений (измерений) параметров , характеризующих изучаемое явление, и выявлении скрытых в этих результатах зависимостей между параметрами (См.разд 1-9).

Наблюдаемыми параметрами в нашем случае являются: длина катетов a и b и гипотенузы с прямоугольного треугольника , а интересующая нас скрытая в этих данных зависимость - соотношение между величинами a, b и c.

Интересующие нас зависимости могут быть достаточно просто установлены тогда, когда они представляют собой полином степени p относительно своих параметров. Используемую формулу представления зависимости между параметрами исследуемой зависимости в виде полинома можно посмотреть в разделе 2. Число членов этого полинома K зависит от числа параметров функции M и степени полинома p .

Решение задачи прямоугольного треугольника предусматривает выполнение следующих трёх этапов.

1. 1. Формирования матрицы наблюдений A[i , j] = | a [i,j] | (i = 1, , M ; j = 1, .. , N) , где N – число экспериментов. Элементы этой матрицы a[i,j] представляют собой измеренные в j– том эксперименте i – того параметра.
2. 2. Формирования матрицы D[i,j] (i = 0, 1,.., K ; j = 1,.., N) . Элементы матрицы d[i,j] определяются на основании элементов матрицы A[i , j] в соответствии с формулой представления зависимости между параметрами исследуемой зависимости.

3. 3. Проверка существования линейной зависимости между столбцами матрицы D[i,j] ; решение системы линейных алгебраических уравнений и определение зависимости между параметрами.

Матрица A[i,j] (i = 1, , M ; j = 1, .. , N) может быть заполнена данными, полученными в результате измерения длинны катетов a[j],b[j] и гипотенузы c[j] (j = 1, .. , N) N разных прямоугольных треугольников. В нашем эксперименте будем придавать величинам a[j] и b[j] независимые положительные случайные значения, а c[j] определять в соответствии с формулой :

c[j] = + Sqrt (a[j]2 + b[j]2), где Sqrt – квадратный корень.

В состав матрицы A[i,j], кроме столбцов , содержащих значения a, b и c, включим ещё один столбец, элементы которого независимы от a и b и выступают в роли помехи.

Матрица результатов измерений A[i,j] , таким образом, имеет следующий вид : первые два столбца - это измеренные значения катетов прямоугольных треугольников

a[j,1] = a[j], a[j,2] = b[j], ;

третий столбец a[j,3] - независимые от a[j,1] и a[j,2] случайные помехи, а четвёртый a[j,4] – их гипотенузы (j=1,..,N).

Число строк матрицы результатов измерений A[i,j] (число измерений N) должно быть выбрано равным числу столбцов матрицы D[i,j] K. В нашем случае при числе столбцов матрицы A[i,j] M=4 и степени полинома p=2 число столбцов матрицы D[i,j] K = 15. Соответственно N = 15.

Результаты проверки на основании критерия существования линейной зависимости между параметрами исследуемого явления приведены в **Таблице 1**.

В нулевой строке таблицы приведено значение определителя матрицы D[i,j]. Оно не равно нулю (хотя линейная зависимость между её

столбцами есть) из-за искусственно введённых ошибок измерения параметров a, b и c.

В строках от первой до пятнадцатой приведены значения определителей этой матрицы после искусственного разрыва причинно-следственных связей между её столбцами, который осуществлялся заменой значений элементов i-того столбца (i=1.2….15) независимыми случайными значениями с таким же, как и у исходного столбца, законом распределения.

Приведенные результаты показывают, что между столбцами матрицы D[i,j] существует только одна линейная зависимость. и в этой зависимости участвуют столбцы № 6,10 и 15 (Выделены розовым цветом). Этим столбцам соответствуют члены разложения соответственно : a[1]2, a[2]2 и a[4]2.

Таблица 1

0	DetD	-3.609 E - 02
1	DetD 1	1.302 E - 03
2	DetD 2	1.625 E - 01
3	DetD 3	9.755 E - 01
4	DetD 4	6.433 E - 02
5	DetD 5	2.344 E - 02
6	DetD 6	5.135 E +16
7	DetD 7	2.128 E - 01
8	DetD 8	3.266 E - 01
9	DetD 9	2.214 E - 02
10	DetD 10	5.091 E +20
11	DetD 11	3.135 E - 01
12	DetD 12	2.093 E - 05
13	DetD 13	1.651 E - 02
14	DetD 14	5.948 E - 01
15	DetD 15	5.684 E +19

Для определения зависимости между параметрами a, b и c необходимо решить систему N – 1 линейных алгебраических уравнений с матрицей размера (N–1)*(N–1), полученной из N*N матрицы D[i,j] вычеркиванием последнего столбца и последней строки и вектором свободных членов, полученным из последнего столбца матрицы D[i,j] удалением последнего элемента. Решение этой системы представлено в таблице 2.

Таблица 2

Coef[1]	-2.3029258500 E-09
Coef[2]	1.1266365618 E-09
Coef[3]	-1.8540049496 E-09
Coef[4]	1.2841994135 E-10
Coef[5]	1.9426806830 E-09
Coef[6]	1.0000000004 E+00
Coef[7]	8.5447027232 E-10
Coef[8]	-5.4154014606 E-11
Coef[9]	-8.4128259914 E-10
Coef[10]	1.0000000013 E+00
Coef[11]	-1.8426504766 E-10
Coef[12]	-1.2960299500 E-09
Coef[13]	-2.9010405189 E-12
Coef[14]	1.8464874074 E-10

Результаты решения показывают, что коэффициенты разложения Coef[6] и Coef[10] при a[1]2 и a[2]2 равны 1, остальные – нулю. Что соответствует зависимости :

$a[1]^2 + a[2]^2 = a[4]^2$

Или, что то же самое:

$a^2 + b^2 = c^2$

1.6.9.Заключение

В заключение отметим **основные особенности предложенного эвристического алгоритма определения физических закономерностей по результатам наблюдений и основанной на нем технологии анализа информации:**

1. Зависимости между параметрами, определяющими исследуемое физическое явление, могут быть определены только в том случае, когда все определяющие явление параметры измерены. Если это условие не выполнено, физическая закономерность установлена быть не может.

2. В случае, когда все определяющие явление параметры измерены, для определения зависимости между параметрами даже при отсутствии ошибок измерения требуется определенный объем наблюдений. Минимальное число экспериментов зависит от сложности исследуемой зависимости: числа параметров, определяющих класс явлений, и степени функциональной зависимости между ними.

3. Предложенный эвристический алгоритм позволяет организовать процедуру поиска решения, включающую проверку по предложенному критерию существования зависимости между параметрами и определение их числа, управление процессом наблюдения исследуемого явления (изменение состава вектора наблюдаемых параметров и числа экспериментов), оценку параметров функций, описывающих исследуемые физические закономерности. Процедура поиска реализует метод проб и ошибок и носит случайный характер.

4. Алгоритм определения параметров физических закономерностей по результатам наблюдений и основанная на нем технология анализа информации подвержены влиянию помех. Результаты моделирования показывают, что при уровне ошибок измерения 5% от значения измеряемых величин картина становится значительно менее выразительной, чем при уровне ошибок в 1,2 и 3%. Для повышения помехозащищенности процедуры необходимо принимать специальные меры: проводить исследования законов распределения участвующих в анализе величин и с учетом результатов такого анализа разрабатывать алгоритмы проверки гипотез и оценки параметров определяемых зависимостей, а также увеличивать число наблюдений, стараться снижать ошибки измерения и добиваться всеми возможными способами совпадения состава и структуры векторов наблюдаемых параметров и параметров, характеризующих изучаемое явление.

1.7. NP-полнота

1.7.1. Введение

Большинство задач, интересных с практической точки зрения, имеют полиномиальные (работающие за полиномиальное время) алгоритмы решения. То есть время работы алгоритма на входе длины n составляет не более $O(n^k)$ для некоторой константы k (не зависящей от длины входа). Разумеется, не каждая задача имеет алгоритм решения, удовлетворяющий этому свойству. Некоторые задачи вообще не могут

быть решены никаким алгоритмом. Классический пример такой задачи — «проблема остановки» (выяснить останавливается ли данная программа на данном входе). Кроме того, бывают задачи, для которых существует решающий их алгоритм, но любой такой алгоритм работает «долго» — время его работы не есть $O(n^k)$ ни для какого фиксированного числа k.

Если мы хотим провести пусть грубую, но формальную границу между «практическими» алгоритмами и алгоритмами, представляющими лишь теоретический интерес, то класс алгоритмов, работающих за полиномиальное время, является разумным первым приближением. Мы рассмотрим, класс задач, называемых *NP-полными*. Для этих задач не найдены полиномиальные алгоритмы, однако и не доказано, что таких алгоритмов не существует. Изучение NP-полных задач связано с так называемым вопросом «P = NP». Этот вопрос был поставлен в 1971 году и является одной из наиболее сложных проблем теории вычислений.

Зачем программисту знать о NP-полных задачах? Если для некоторой задачи удается доказать ее NP-полноту, есть основания считать ее практически неразрешимой. В этом случае лучше потратить время на построение приближенного алгоритма, чем продолжать искать быстрый алгоритм, решающий ее точно.

1.7.2. Полиномиальное время

Абстрактные задачи

Как уже упоминалось, понятие полиномиально разрешимой задачи принято считать уточнением идеи «практически разрешимой» задачи. Чем объясняется такое соглашение? Во-первых, используемые на практике полиномиальные алгоритмы обычно действительно работают довольно быстро. Второй аргумент в пользу рассмотрения класса полиномиальных алгоритмов — тот факт, что объем этого класса практически не зависит от выбора конкретной модели вычислений. Например, класс задач, которые могут быть решены за полиномиальное время на последовательной машине с произвольным доступом (RAM), совпадает с классом задач, полиномиально разрешимых на машинах Тьюринга. Класс будет тем же и для модели параллельных вычислений, если, конечно, число процессоров

ограничено полиномом от длины входа. В-третьих, класс полиномиально разрешимых задач обладает естественными свойствами замкнутости. Например, композиция двух алгоритмов также работает полиномиальное время. Это объясняется тем, что сумма, произведение и композиция многочленов есть многочлен.

Введем следующую абстрактную модель вычислительной задачи. Будем называть *абстрактной задачей* — произвольное бинарное отношение Q между элементами двух множеств — множества условий I и множества решений S. Например, в задаче поиска кратчайшего пути между двумя заданными вершинами неориентированного графа $G = (V, E)$ условием (элементом I) является тройка, состоящая из графа и двух вершин, а решением (элементом S) — последовательность вершин, составляющих требуемый путь в графе.

В теории NP-полноты рассматриваются только *задачи разрешения* — задачи, в которых требуется дать ответ «да» или «нет» на некоторый вопрос. Формально её можно рассматривать как функцию, отображающую множество условий I во множество $\{0, 1\}$. Например, с задачей поиска кратчайшего пути в графе связана задача разрешения: «по заданному графу $G = (V, E)$, паре вершин $u, v \in V$ и натуральному числу k определить, существует ли в G путь между вершинами u и v, длина которого не превосходит k».

Часто встречаются *задачи оптимизации* — задачи, в которых требуется максимизировать или минимизировать значение некоторой величины. Прежде чем ставить вопрос о NP-полноте таких задач, их следует преобразовать в задачи разрешения. Так, например, в задаче о поиске кратчайшего пути в графе мы перешли от задачи оптимизации к задаче разрешения, добавив ограничение на длину пути. Если после преобразования получается NP-полная задача, то тем самым установлена и трудность исходной задачи.

Представление данных

Прежде чем подавать на вход алгоритма исходные данные (то есть элемент множества I), надо договориться о том, как они представляются в «понятном для компьютера виде». Будем считать, что исходные данные закодированы последовательностью битов. Формально говоря, *представлением* элементов некоторого множества S называется отображение e из S во множество битовых строк.

Например, целые числа 0, 1, 2, 3, … обычно представляются битовыми строками 0, 1, 10, 11, 100, …

Фиксировав представление данных, мы превращаем абстрактную задачу в *строковую*, для которой входными данными является битовая строка, представляющая исходные данные абстрактной задачи. Будем говорить, что алгоритм *решает* строковую задачу за время $O(T(n))$, если на входных данных (битовой строке) длины n алгоритм работает время $O(T(n))$. Строковая задача называется *полиномиальной*, если существует константа k и алгоритм, решающий эту задачу за время $O(n^k)$. *Сложностной класс P* — множество всех строковых задач разрешения, которые могут быть решены за полиномиальное время, т. е. за время $O(n^k)$, где k не зависит от входа.

Желая определить понятие полиномиальной абстрактной задачи, мы сталкиваемся с тем, что возможны различные представления данных. Для каждого представления e множества I входов абстрактной задачи Q мы получаем свою строковую задачу. Однако на практике (если исключить такие «дорогие» способы представления, как система счисления с основанием 1) естественные способы представления оказываются обычно эквивалентными, поскольку они могут быть полиномиально преобразованы друг в друга. Будем говорить, что функция $f: \{0,1\}^* \to \{0,1\}^*$ *вычислима за полиномиальное время*, если существует полиномиальный алгоритм A, который для любого $x \in \{0,1\}^*$ выдает результат $f(x)$.

Рассмотрим множество I условий произвольной абстрактной задачи разрешения. Два представления e_1 и e_2 этого множества называются *полиномиально связанными*, если существуют две вычислимые за полиномиальное время функции f_{12} и f_{21}, для которых $f_{12}(e_1(i)) = e_2(i)$ и $f_{21}(e_2(i)) = e_1(i)$ для всякого $i \in I$. В этом случае не имеет значения, какое из двух полиномиально связанных представлений выбрать.

Формальные языки

Для задач разрешения удобно использовать терминологию теории формальных языков. *Алфавитом* Σ называется любой конечный набор различных символов. *Языком* L над алфавитом Σ называется произвольное множество строк символов из алфавита Σ (такие строки называются *словами* в алфавите Σ). Например, можно рассмотреть $\Sigma = \{0, 1\}$ и язык $L = \{10, 11, 101, 111, 1011, \ldots\}$, состоящий из

двоичных записей простых чисел. Будем обозначать символом ε *пустое слово*, не содержащее символов, а символом ∅ — *пустой язык*, не содержащий слов.

Имеется несколько стандартных операций над языками. Операции *объединения* и *пересечения* языков определяются как обычные операции объединения и пересечения множеств. *Конкатенацией*, или *соединением*, двух языков L_1 и L_2 называется язык $L = \{x_1x_2 \mid x_1 \in L_1, x_2 \in L_2\}$ *Замыканием* языка L называется язык $L^* = \{\varepsilon\} \in L \in L^2 \in L^3 \in \ldots$, где L^k — язык, полученный k-кратной конкатенацией L с самим собой. Операция замыкания называется также **-операцией Клини*.

Теперь можно сказать, что строковая задача разрешения является языком над алфавитом Σ. Например, задаче разрешения о существовании в графе пути длины не превосходящей k, соответствует язык $PATH = \{\langle G, u, v, k\rangle \mid G = (V, E)$ — неориентированный граф, $u, v \in V; k \geq 0$ — целое число, и в графе G существует путь из u в v, длина которого не превосходит k.$\}$

Говорят, что алгоритм A *допускает слово* $x \in \{0,1\}^*$, если на входе x алгоритм выдает результат 1; если же A выдает 0, говорят, что он *отвергает слово* x. Заметим, что алгоритм может не останавливаться на входе x, или дать ответ отличный от 0, 1. В этом случае он и не допускает, и не отвергает слово x. Алгоритм A *допускает язык* L, если алгоритм допускает те и только те слова, которые принадлежат языку L. Алгоритм A, допускающий некоторый язык L, не обязан отвергать всякое слово $x \notin L$. Если алгоритм A допускает все слова из L, а все остальные отвергает, то говорят, что A *распознает язык* L. Язык L *допускается за полиномиальное время*, если имеется алгоритм A, который допускает данный язык, причем всякое слово $x \in L$ допускается алгоритмом за время $O(n^k)$, где n — длина слова x, а k — некоторое не зависящее от x число. Язык L *распознается за полиномиальное время*, если некоторый алгоритм A распознает данный язык, причем время работы алгоритма на каждом слове длины n не больше $O(n^k)$.

Рассмотренный ранее язык PATH, допускается за полиномиальное время. Нетрудно построить алгоритм, который методом поиска в ширину за полиномиальное время находит кратчайший путь между

вершинами u и v в графе G, а затем сравнивает длину найденного пути с данным в условии числом k. Если длина пути не превосходит k, алгоритм выдает 1 и останавливается. В противном случае алгоритм зацикливается, не выдавая никакого ответа. Ясно, что такой алгоритм допускает, но не распознает язык PATH. Однако легко исправить описанный алгоритм таким образом, чтобы слова, не принадлежащие языку, отвергались. Такой алгоритм допускает и распознает язык PATH. Стоит отметить, что некоторые языки (например, для множества всех программ, заканчивающих свою работу) имеют допускающий алгоритм, но не имеют распознающего. Определение класса задач P мы уже имеем. Дадим определение класса NP. Неформально *сложностной класс* можно определить как семейство языков, для которых распознающие алгоритмы имеют заданную меру сложности, например заданное время работы. В теории существует множество сложностных классов. Один из них (P) мы уже рассматривали. Есть также не менее важный класс PSPACE — состоящий из задач, решаемых алгоритмами с использованием памяти полиномиального размера. Или класс EXP, состоящий из задач, которые решаются за время $O(2^{n^k})$. Переформулируем определение класса P: P = {$L \in \{0,1\}^*$ | существует алгоритм A, распознающий L за полиномиальное время.} На самом деле в данной ситуации нет разницы между языками допускаемыми и распознаваемыми за полиномиальное время, о чем свидетельствует следующая теорема.

Теорема

P = {L | L допускается за полиномиальное время}.

Доказательство. Если язык распознается некоторым алгоритмом, то он и допускается тем же алгоритмом. Остается доказать, что если язык L допускается полиномиальным алгоритмом A, то он и распознается некоторым (возможно, другим) полиномиальным алгоритмом A'. Пусть алгоритм A допускает язык L за время $O(n^k)$. Это значит, что существует константа c, для которой A допускает любое слово длины n из L, сделав не более $T = cn^k$ шагов. С другой стороны слова не из L алгоритм не допускает (ни за какое время). Новый алгоритм A' моделирует работу алгоритма A и считает число шагов этого алгоритма, сравнивая его с известной границей T. Если за время T алгоритм A допускает слово x, алгоритм A' также допускает это слово и выдает 1. Если же A не допускает x за указанное время, то алгоритм A' прекращает моделирование и отвергает слово (выдает 0). Замедление

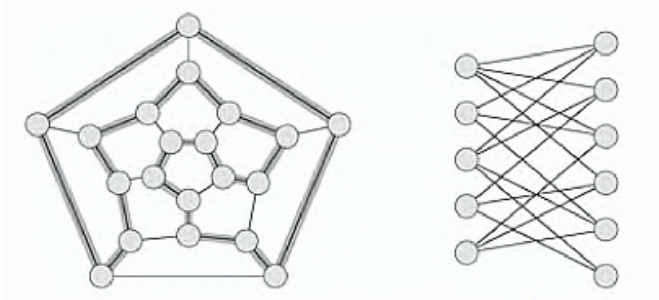
работы за счет моделирования и подсчета шагов не так уж велико и оставляет время работы полиномиальным.

1.7.3. Проверка принадлежности языку и класс NP

Выше мы рассматривали задачу разрешения PATH. Эта задача оказалась полиномиальной (и решается с помощью алгоритма поиска в ширину). Допустим, однако, что мы ничего не знаем про поиск в ширину. Тогда задача PATH будет для нас трудной: видя граф G и вершины u и v и зная число k, мы не будем знать, есть ли искомый путь, пока нам его не покажут. Но если он нам станет известен, то мы можем легко проверить, что этот путь — искомый. Именно такая ситуация имеет место для задач из класс NP.

Гамильтонов цикл

Задача поиска гамильтонова цикла в графе изучается уже более ста лет. *Гамильтоновым циклом* в неориентированном графе $G = (V, E)$ называется простой цикл, который проходит через все вершины графа. Графы, в которых есть гамильтонов цикл, называют *гамильтоновыми*. *Задача о гамильтоновом цикле* состоит в выяснении, имеет ли заданный граф G гамильтонов цикл. Формально говоря, *HAM-CYCLE* = {<G> | G — гамильтонов граф}. Задача о гамильтоновом цикле является NP-полной, и поэтому можно предполагать, что полиномиального алгоритма для нее вообще не существует. На рисунке ниже приведены примеры двух графов, левый из которых является гамильтоновым, правый же не обладает аналогичным свойством.



Проверка принадлежности языку

Определить, является ли граф гамильтоновым, за полиномиальное время, скорее всего, невозможно, однако доказательство существования гамильтонова цикла в графе (состоящее в предъявлении гамильтонова пути) можно проверить за полиномиальное время. Назовем *проверяющим алгоритмом* алгоритм A с двумя аргументами; первый аргумент — входная строка, а второй — *сертификат*. A *допускает* вход x, если существует сертификат y, для которого $A(x, y) = 1$. *Язык, проверяемый алгоритмом* A: $L = \{x \in \{0, 1\}^* : \exists\ y \in \{0,1\}^*, A(x, y) = 1\}$. Другими словами, алгоритм A проверяет язык L, если для любой строки $x \in L$ найдется сертификат y, с помощью которого A может проверить принадлежность x к языку L, а для $x \notin L$ такого сертификата нет. Например, в задаче HAM-CYCLE сертификатом была последовательность вершин, образующая гамильтонов цикл.

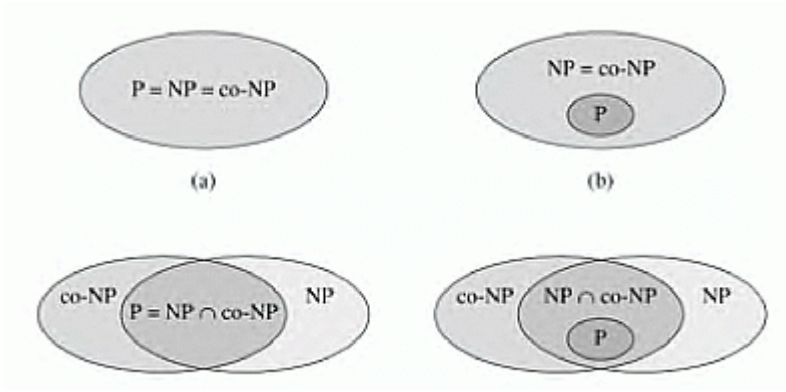
Сложностной класс NP

Сложностной класс NP — класс языков, для которых существуют проверяющие алгоритмы, работающие полиномиальное время, причем длина сертификата также ограничена некоторым полиномом.

Более формально. Язык L принадлежит классу NP, если существует такой полиномиальный алгоритм A с двумя аргументами и такой многочлен $p(x)$ с целыми коэффициентами, что $L = \{x \in \{0, 1\}^* : \exists\ y,$ для которого $|y| \le p(|x|)$ и $A(x, y) = 1\}$. В этом случае говорят, что A проверяет L за полиномиальное время.

Ранее уже была рассмотрена задача из класса NP — это задача HAM-CYCLE. Кроме того, всякая задача из P принадлежит также NP. Действительно, если есть полиномиальный алгоритм, распознающий язык, то легко построить проверяющий алгоритм для того же языка — проверяющий алгоритм может просто игнорировать свой второй аргумент (сертификат). Таким образом, $P \in NP$. В данное время неизвестно совпадают ли классы P и NP, но большинство специалистов полагает, что нет. Наиболее серьезная причина полагать, что $P \neq NP$ — существование NP полных задач (они будут рассмотрены далее). Кроме проблемы $P = NP$, остаются открытыми и многие другие вопросы о классе NP. Так, несмотря на огромные усилия

исследователей, не известно, замкнут ли класс NP относительно дополнения. Определив сложностной класс co-NP, как множество языков L, для которых $\neg L \in$ NP, можно переформулировать вопрос о замкнутости класса NP относительно дополнения следующим образом: равны ли классы NP и co-NP? Поскольку класс P замкнут относительно перехода к дополнению, $P \subset NP \cap co\text{-}NP$. В то же время остается неизвестным, верно ли равенство $P = NP \cap co\text{-}NP$. Приведенная ниже иллюстрация, отображает четыре возможных ситуации.

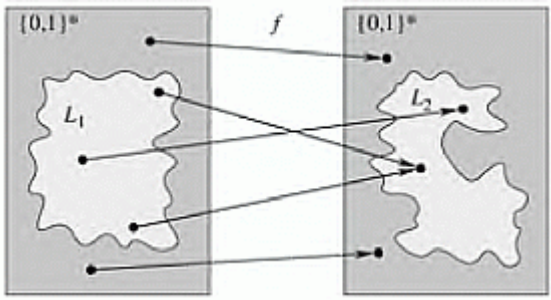


К сожалению, о соотношениях классов P и NP почти ничего не известно. Но уже понятие NP полноты является важным средством классификации задач; как станет ясно далее, оно сводит вопрос о сложности данной задачи к общему (пусть и не решенному) вопросу о соотношении классов P и NP.

1.7.4. NP-полнота и сводимость

Наиболее убедительным аргументом в пользу того, что классы P и NP различны, является существование так называемых NP-полных задач. Этот класс обладает тем важным свойством, что если какая-нибудь NP-полная задача разрешима за полиномиальное время, то и все задачи из класса NP разрешимы за полиномиальное время, то есть P = NP. В частности, задача HAM-CYCLE является NP-полной. Таким образом, научившись решать ее за полиномиальное время, мы получим полиномиальные алгоритмы для всех задач класса NP. Неформально говоря, NP-полные языки являются самыми «трудными» в классе NP. При этом трудность языков можно сравнивать, сводя один язык к

другому. Метод сведения является основным при доказательстве NP-полноты многих задач.



Сводимость

Неформально, задача Q сводится к задаче Q', если задачу Q можно решить для любого входа, считая известным решение задачи Q' для какого-то другого входа. Например, задача решения линейного уравнения сводится к задаче решения квадратного уравнения.

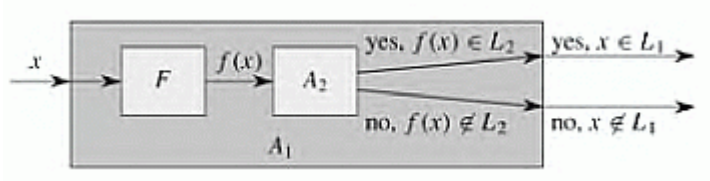
Теперь дадим формальное определение. Говорят, что язык L_1 *сводится за полиномиальное время* к языку L_2 (запись: $L_1 \leq_P L_2$), если существует такая функция f: $\{0,1\}^* \rightarrow \{0, 1\}^*$, вычислимая за полиномиальное время, что для любого $x \in \{0, 1\}^*$: $x \in L_1$ равносильно $f(x) \in L_2$. Функцию f называют *сводящей функцией*, а полиномиальный алгоритм F, вычисляющий f, — *сводящим алгоритмом*. Рисунок, приведенный выше, иллюстрирует данное определение сводимости. Запись $L_1 \leq_P L_2$ можно интерпретировать так: сложность языка L_1 не более чем полиномиально превосходит сложность языка L_2.

Лемма

Если язык $L_1 \in \{0, 1\}^*$ сводится за полиномиальное время к языку $L_2 \in \{0, 1\}^*$ и $L_2 \in P$, то $L_1 \in P$.

Доказательство. Пусть A_2 — полиномиальный алгоритм, распознающий язык L_2, а F — полиномиальный алгоритм, сводящий язык L_1 к языку L_2. Построим алгоритм A_1, который будет за полиномиальное время разрешать язык L_1, согласно нижеприведенной

иллюстрации, а именно: получив на вход $x \in \{0, 1\}^*$, алгоритм A_1 (с помощью алгоритма F) получает $f(x)$ и с помощью алгоритма A_2 проверяет, принадлежит ли слово $f(x)$ языку L_2. Результат работы алгоритма A_2 на основе $f(x)$ и выдается алгоритмом A_1 в качестве ответа. Определение полиномиальной сводимости гарантирует, что алгоритм A_1 дает правильный ответ; он полиномиален, поскольку полиномиальны алгоритмы F и A_2.



NP-полнота

Понятие сводимости позволяет формализовать сравнение языков относительно их трудности. Запись $L_1 \leq_P L_2$ можно интерпретировать так: сложность языка L_1 не более чем полиномиально превосходит сложность языка L_2. Наиболее трудны в этом смысле NP-полные задачи. Язык $L \in \{0, 1\}^*$ называется *NP-полным*, если $L \in$ NP и $L' \leq_P L$ для любого $L' \in$ NP. Класс NP-полных языков будем обозначать *NPC*. Языки, которые обладают вторым свойством, но не обязательно отвечают первому, называют *NP-трудными*. Сформулируем *основное свойство NPC языков* в виде следующей несложной теоремы:

Теорема

Если некоторая NP-полная задача разрешима за полиномиальное время, то P = NP. Другими словами, если в классе NP существует задача, не разрешимая за полиномиальное время, то все NP-полные задачи таковы.

Доказательство. Пусть L — NP-полный язык, который одновременно оказался разрешимым за полиномиальное время. Тогда для любого языка $L' \in$ NP из определения NP-полного языка имеем $L' \leq_P L$. Следовательно, $L' \in$ P, и теорема доказана.

Таким образом, гипотеза P ≠ NP означает, что NP-полные задачи не могут быть решены за полиномиальное время. Видимо, это действительно так, а потому доказательство NP-полноты некоторой задачи является существенным аргументом в пользу ее практической неразрешимости.

Итак, какой же практический смысл имеет изучение теории сложности и классификация задач с точки зрения NP-полноты? Ответ очевиден — зачастую гораздо разумнее и эффективнее найти доказательство того, что рассматриваемая задача принадлежит к классу NP-полных, и в соответствие с этим заняться поиском достаточно точных приближенных алгоритмов, нежели безрезультатно тратить время на отыскание полиномиальных алгоритмов ее решения. Ясно, что именно NP-полные задачи играют здесь центральную роль — дело в том, что полиномиальное время является, хоть и первым, но достаточно хорошим приближением понятия «практической разрешимости задачи».

1.8. Приближенные алгоритмы решения сложных задач

Это вводнный раздел в приближенные алгоритмы. В ней мы рассмотрим лишь самые простые и известные алгоритмы для приближенного решения NP-трудных задач.

На данный момент не известно полиномиальных алгоритмов для точного решения NP-трудных задач. Более того, специалисты по теории сложности склоняются к варианту, что таких алгоритмов не существует. Однако, NP-трудные задачи часто встречаются в жизни. Одним из подходов борьбы с NP-трудными задачами на практике является разработка приближенных алгоритмов.

Рассмотрим следующие задачи.

1) Задача о вершинном покрытии (Vertex Cover).
Задача заключается в том, чтобы выбрать в неориентированном графе G=(V, E) минимальное (по количеству вершин) множество вершин S так, чтобы оно покрывало все рёбра графа. То есть так, чтобы у

каждого из рёбер графа хотя бы один из концов принадлежал S.

Построим простой алгоритм решения этой задачи, ошибающийся не более чем в два раза. Это означает, что если есть оптимальное решение — множество вершин T, то |S| <= 2|T|.
Алгоритм:
Выбираем случайное ребро графа e=(u, v). Добавляем в решение S обе вершины u и v. Удаляем из множества рёбер все рёбра, инцидентные вершинам u или v.
Повторяем этот шаг пока рёбра не закончатся.
Очевидно, что построенное множество покрывает все рёбра графа.
Докажем, что наш алгоритм ошибается не более, чем в 2 раза.
Все рассматриваемые алгоритмом рёбра e не имеют общих вершин. Следовательно, из каждого из этих рёбер в оптимальное решение T должна входить хотя бы одна вершина. Это значит, что 2|T|>=|S|.
Интересно, что не известно полиномиальных приближенных алгоритмов для Vertex Cover значительно улучшающих ошибку. То есть, не известно полиномиального алгоритма, который ошибается не более, чем в 1.999 раза. Таким образом, мы получили простой алгоритм с небольшой ошибкой.

2) Задача о коммивояжёре (Travelling salesman problem).
Вероятно, самая знаменитая NP-трудная задача. Задача заключается в том, чтобы пройти все вершины неориентированного графа ровно по одному разу и вернуться в исходную вершину. Кроме того, для каждого ребра указан его вес. И нам необходимо найти такой обход графа с минимальной суммой весов пройденных рёбер.

Интересно то, что для данной задачи не известно никакого полиномиального приближенного алгоритма, ошибающегося в C раз. Более того, доказано, что в предположении P != NP, такого алгоритма не существует. Доказательство этого простого факта и 2-приближенный алгоритм для метрического пространства можно найти в огромном количестве книг. К примеру, в книге Кормен, Т., Лейзерсон, Ч., Ривест, Р. Алгоритмы: построение и анализ /Introduction to Algorithms.
Мы же остановимся на более интересном алгоритме. 3/2 приближенный алгоритм для задачи коммивояжёра в метрическом пространстве. Метрическое пространство в названии задачи означает, что для любых трёх вершин графа выполняется неравенство треугольника:
w(a, b) <= w(a, c) + w(c, b).

Алгоритм начинается также, как в 2-приближении. Мы строим минимальное остовное дерево T исходного графа G. Выберем в T все вершины нечётной степени. На найденных вершинах нечётной степени построим минимальное полное паросочетание и добавим все полученные рёбра в дерево T. В пополненном рёбрами дереве мы можем найти эйлеров цикл (так как степени всех вершин уже стали чётными). Теперь просто удаляем из полученного цикла все повторения вершин и получаем ответ.
Несложно показать, что полученный алгоритм является 3/2 приближением. Для этого достаточно показать, что вес минимального паросочетания вершин нечётной степени из T не превосходит половины оптимального ответа.
Это лучший известный приближенный алгоритм задачи коммивояжёра в метрическом пространстве.

3) Покрытие множествами (Set Cover)
Задача покрытия множествами является обобщением задачи о вершинном покрытии. Дано множество U. Задано B — множество подмножеств U. Необходимо выбрать минимальный набор множеств из B так, чтобы покрыть всё множество U.

Приближенный алгоритм решения данной задачи является жадным алгоритмом. То есть на каждом шаге алгоритм выбирает «наиболее выгодное в данный момент» множество. То есть на каждом шаге алгоритм должен выбирать то множество, которое покроет наибольшее количество ещё не покрытых элементов множества U.
Легко показать, что ошибка данного алгоритма O(log |U|). Также можно построить пример с логарифмической ошибкой. Доказана теорема, говорящая о том, что в предположении P != NP, значительно улучшить ошибку полиномиальным алгоритмом не удастся.

1.9. Приближенные алгоритмы для NP-полных задач

1.9.1. Понятие NP-полной задачи

Задача поиска (search problem) задаётся алгоритмом C, который получает на вход условие I и кандидата на решение S и имеет время работы, ограниченное полиномом от | I |. S называется решением (solution), если и только если C(S, I)=true.
NP — класс всех задач поиска.

P — класс задач поиска, решение для которых может быть быстро найдено (за полиномиальное время).

Большинство исследователей считают, что $P \neq NP$.

Задача поиска называется NP-полной (NP-complete), если к ней сводятся все задачи поиска.

В предположении $P \neq NP$ не существует полиномиальных алгоритмов для NP-полных задач.

1.9.2. Понятие приближенного алгоритма

Многие задачи, представляющие практический интерес - NP-полные. Для них маловероятно найти точный алгоритм с полиномиальным временем работы.
При небольшом объеме входных данных может подойти алгоритм, время работы которого выражается показательной функцией.
Иногда удается выделить важные частные случаи, разрешимые в течение полиномиального времени.
Можно найти в течение полиномиального времени решение, близкое к оптимальному.

Алгоритм, возвращающий решения, близкие к оптимальным, называется **приближенным алгоритмом.**

1.9.3. Методы решения NP-полных задач

Приближенные и эвристические методы - применение эвристик для выбора элементов решения.
Псевдополиномиальные алгоритмы - подкласс динамического программирования.
Метод локальных улучшений - поиск более оптимального решения в окрестности некоторого текущего решения.
Метод ветвей и границ - отбрасывание заведомо неоптимальных решений целыми классами в соответствии с некоторой оценкой.

Метод случайного поиска - представление выбора последовательностью случайных выборов.

1.9.4. Оценка качества приближенных алгоритмов

Говорят, что алгоритм обладает коэффициентом аппроксимации ρ(*n*), если

$$\max\left(\frac{C}{C^*}, \frac{C^*}{C}\right) \le \rho(n)$$

n – размер входных данных
C – стоимость решения
C* - стоимость оптимального решения

ρ(*n*)- приближенный алгоритм - алгоритм, в котором достигается коэффициент аппроксимации ρ(*n*).
Рассматриваемые примеры:
2-приближенный алгоритм решения задачи о вершинном покрытии.
2-приближенный и 3/2-приближенный алгоритмы решения задачи о коммивояжере.
(Ln|n|+1)-приближенный алгоритм решения задачи о покрытии множества.
Схема аппроксимации - приближенный алгоритм, входные данные которого включают в себя такое ε *>0,* что для любого фиксированного ε эта схема является (1 + ε)- приближенным алгоритмом.
Схема аппроксимации с полиномиальным временем выполнения - схема аппроксимации, работа которой при любом фиксированном ε *>0* завершатся в течение времени, выраженного полиномиальной функцией от размера п входных данных.
Схема аппроксимации с полностью полиномиальным временем работы - схема аппроксимации, время работы которой выражается полиномом от 1/ ε и размера входных данных задачи *n*.
Рассматриваемый пример: алгоритм решения задачи о сумме подмножества - (1 + ε)-приближенный с временем работы O(Polynom(4n*ln(t)/ε))

1.9.5. Задача о вершинном покрытии

Определения и постановка

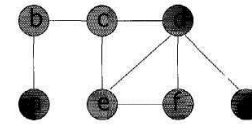
Определение: *Вершинное покрытие* неориентированного графа G =(V, E) - такое подмножество $V' \subseteq V$, что если (u, v)- ребро графа G, то либо $u \in V'$, либо $v \in V'$ (могут выполняться и оба соотношения).
Размер вершинного покрытия - количество содержащихся в нем вершин.

Задача: найти для заданного неориентированного графа вершинное покрытие минимального размера.

Алгоритм

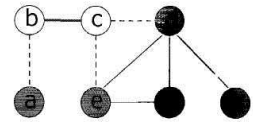
2-приближенный алгоритм

```
Approx_Vertex_Cover(G)
1. C ← Ø
2. E′ ← E[G]
3. While E′ ≠ Ø
4.      do  (u, v) – произвольное
                     ребро из E′
5.           C ← C ∪ {u, v}
6.      Удаляем из множества E′ все
        ребра, инцидентные вершинам
        u или v
7. Return C
```



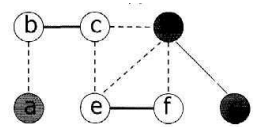
2-приближенный алгоритм

```
Approx_Vertex_Cover(G)
1. C ← Ø
2. E′ ← E[G]
3. While E′ ≠ Ø
4.      do  (u, v) – произвольное
                     ребро из E′
5.            C ← C ∪{u, v}
6.        Удаляем из множества E′ все
          ребра, инцидентные вершинам
          u или v
7. Return C
```



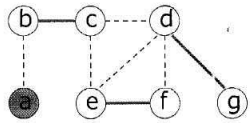
2-приближенный алгоритм

```
Approx_Vertex_Cover(G)
1. C ← Ø
2. E′ ← E[G]
3. While E′ ≠ Ø
4.      do  (u, v) – произвольное
                     ребро из E′
5.            C ← C ∪{u, v}
6.        Удаляем из множества E′ все
          ребра, инцидентные вершинам
          u или v
7. Return C
```



2-приближенный алгоритм

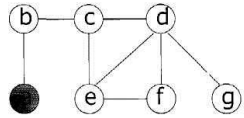
```
Approx_Vertex_Cover(G)
1. C ← Ø
2. E′ ← E[G]
3. While E′ ≠ Ø
4.      do  (u, v) – произвольное
                     ребро из E′
5.            C ← C ∪{u, v}
6.       Удаляем из множества E′ все
         ребра, инцидентные вершинам
         u или v
7. Return C
```



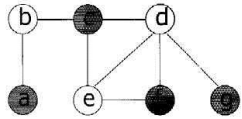
2-приближенный алгоритм

```
Approx_Vertex_Cover(G)
1. C ← Ø
2. E′ ← E[G]
3. While E′ ≠ Ø
4.      do  (u, v) – произвольное
                     ребро из E′
5.            C ← C ∪{u, v}
6.       Удаляем из множества E′ все
         ребра, инцидентные вершинам
         u или v
7. Return C
```

Результат:



Оптимальное решение:



Теорема о точности алгоритма

Теорема:
Approx_Vertex_Cover является 2-приближенным алгоритмом с полиномиальным временем работы.
Доказательство

время работы
O(V+E) при использовании списков смежных вершин.
корректность

множество вершин C, возвращаемое алгоритмом -вершинное покрытие, т.к. алгоритм не выходит из цикла, пока каждое ребро E[G] не будет покрыто некоторой вершиной из множества C.

точность
A - множество выбранных ребер
C* - оптимальное покрытие
C - возвращаемое алгоритмом покрытие
C* содержит хотя бы одну конечную точку каждого ребра из A. Никакие два ребра из A не покрываются одной и той же вершиной из C*. Следовательно

$$|C^*| \geq |A|$$

Каждый раз выбирается ребро, ни одна из конечных точек которого не вошла в C. Поэтому

$$|C| = 2|A|$$

Получаем

$$|C| = 2|A| \leq 2|C^*|$$

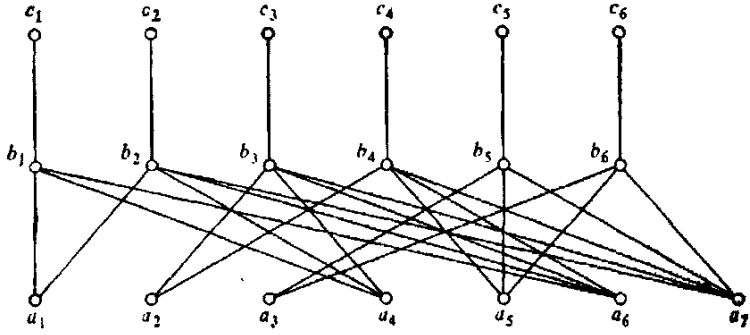
$$\rho(n) = C/C^* = 2$$

Другие алгоритмы

Существует множество других решений этой задачи, однако значение всех коэффициентов аппроксимации не меньше 2 - o(1).
Использование эвристики выбора вершин с максимальной кратностью приводит к алгоритму, не имеющему фиксированной границы для получаемой относительной ошибки.

Выбор вершин с максимальной степенью

Контрпример для эвристики выбора вершин с максимальной степенью. Коэффициент приближения равен L (n)/n+1, где L(n) - число вершин типа a.



n	6	10	30	100	1000	...
$L(n)/n$ (%)	117	160	267	380	600	...

1.9.7. Задача о коммивояжере

Определения и постановка

Дано:

полный неориентированный граф G=(V,E)

каждому ребру (u,v) ∈ E сопоставляется целочисленная стоимость c(u,v)≥0

c(u,v) удовлетворяет неравенству треугольника, т.е. для всех u,v,w∈V

$$c(u, w) \leq c(u,v) + c(v,w)$$

Определение: гамильтонов цикл - простой цикл, содержащий все вершины множества V.

Задача: найти гамильтонов цикл минимальной стоимости.

Алгоритм

2-приближенный алгоритм

Approx_TSP_Tour(G, c)

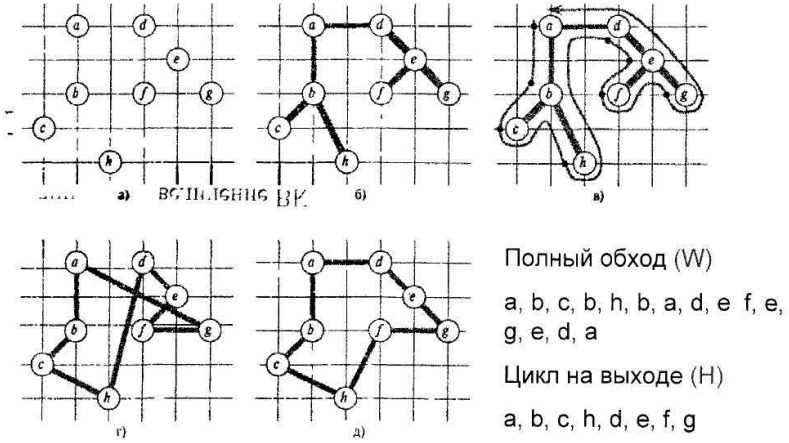
1 Выбирается вершина $r \in V[G]$, которая будет «корневой»

2 Из корня *r* с помощью алгоритма MST-Prim(G c, *r*) строится минимальное остовное дерево T для графа G

3. Пусть L - список вершин, которые посещаются при обходе

вершин дерева T в прямом порядке
4. return Гамильтонов цикл H, который посещает вершины в порядке перечисления в списке L

Пример



Теорема о точности алгоритма

Теорема: Approx_TSP_Tour является 2-приближенным алгоритмом с полиномиальным временем работы.
Доказательство
Время работы $O(V^2)$
Точность
H* - оптимальный тур -
T - минимальное остовное дерево

$$c(T) \le c(H^*) \qquad (1)$$

W - список вершин, посещаемых при полном обходе T

$$c(W) = 2c(T) \qquad (2)$$

$$(1), (2) \Rightarrow c(W) \le 2c(H^*) \qquad (3)$$

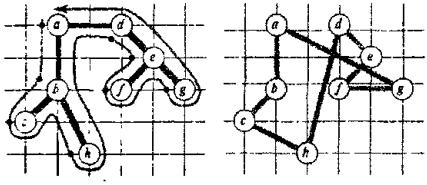
H - цикл, возвращаемый алгоритмом
H получается из W путем исключения повторных вхождений каждой вершины. Согласно неравенству треугольника, стоимость при этом не возрастает.

$$c(H) \le c(W) \qquad (4)$$

$$(3), (4) \Rightarrow c(H) \le 2c(H^*)$$

$$\rho = c(H)/c(H^*) = 2$$

Пример:



Полный обход **(W):**

a, b, c, b, h. b. a, d, e, f, e, g, e, d, a

Цикл на выходе **(H):**

a, b, c, h, d, e, f, g

3/2-приближенный алгоритм

```
Approx-TSP-Improved (G)
```
1. Построить минимальное остовное дерево т графа G
2. Найти минимальное полное паросочетание всех вершин дерева Т нечетной степени
3. Добавить найденные ребра в дерево Т и найти в полученном графе эйлеров цикл
4 . Убрать из полученного цикла все повторения вершин и вернуть полученный цикл

Теорема: Approx_TSP_Tour является 3/2-приближенныл алгоритмом с полиномиальным временем работы.

Доказательство 3/2-приближенности:

Стоимость построенного цикла не провосходит c(T) +c(P), где c(P) — вес минимального паросочетания вершин нечетной степени дерева Т

Нужно показать, что $c(P) \leq c(H^*)/2$

Обозначим через А множество всех вершин нечетной степени дерева Т

Рассмотрим такой гамильтонов цикл на вершинах множества А: вершины множества А в нем будут встречаться в такой последовательности, в какой они идут в Н*.

Разбив вершины этого цикла на четные и нечетные, мы получим два паросочетания.

Вес хотя бы одного из них будет не более c(H*)/2.

Значит, и вес минимального паросочетания не превосходит C(H*)/2

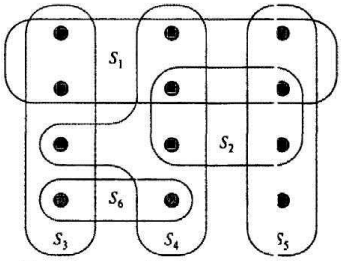
1.9.8. Задача о покрытии множества

Постановка

- **Дано:**
  - конечное множество X
  - семейство F подмножеств множества X такое, что каждый элемент X принадлежит хотя бы одному подмножеству.

  Говорят, что $S \in F$ покрывает содержащиеся в нем элементы.
- **Задача:**

  найти подмножество $C \subseteq F$ минимального размера, члены которого покрывают все множество X: $X = \bigcup_{S \in C} S$



X состоит из 12 черных точек

$F = \{S_1, S_2, S_3, S_4, S_5, S_6\}$

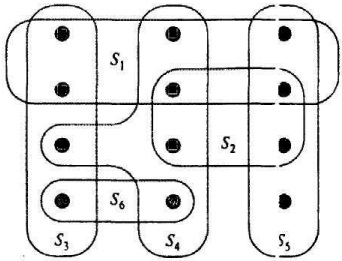
Покрытие минимального размера $C^* = \{S_3, S_4, S_5\}$.

Покрытие, возвращаемое алгоритмом $C = \{S_1, S_4, S_5, S_3\}$

Алгоритм

- **p(n)-приближенный алгоритм**

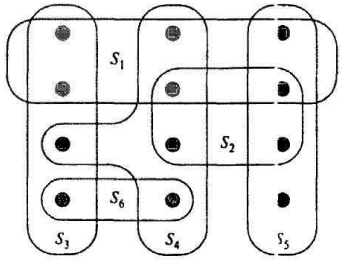
```
Greedy_Set_Cover(X, F)
1.  U ← X
2.  C ← Ø
3.  while U ≠ Ø
4.      do выбирается подмн. S∈F,
        максимизирующее |S ∩ U|
5.      U ← U - S
6.      C ← C ∪ {S}
7.  return C
```



$C = \{\emptyset\}$

□ **p(n)-приближенный алгоритм**

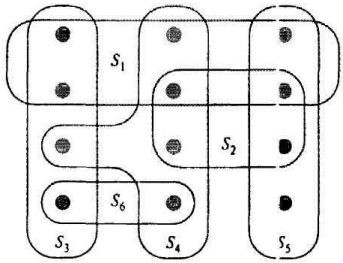
```
Greedy_Set_Cover(X, F)
1. U ← X
2. C ← Ø
3. while U ≠ Ø
4.     do выбирается подмн. S∈F,
       максимизирующее |S ∩ U|
5.     U ← U - S
6.     C ← C ∪ {S}
7. return C
```



$C = \{S_1\}$

□ **p(n)-приближенный алгоритм**

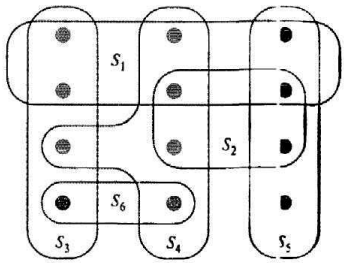
```
Greedy_Set_Cover(X, F)
1. U ← X
2. C ← Ø
3. while U ≠ Ø
4.     do выбирается подмн. S∈F,
       максимизирующее |S ∩ U|
5.     U ← U - S
6.     C ← C ∪ {S}
7. return C
```



$C = \{S_1, S_4\}$

□ **ρ(n)-приближенный алгоритм**

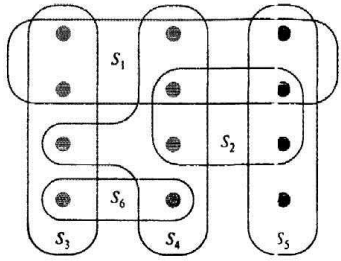
```
Greedy_Set_Cover(X, F)
1. U ← X
2. C ← Ø
3. while U ≠ Ø
4.     do выбирается подмн. S∈F,
       максимизирующее |S ∩ U|
5.     U ← U - S
6.     C ← C ∪ {S}
7. return C
```



$C = \{S_1, S_4, S_5\}$

□ **ρ(n)-приближенный алгоритм**

```
Greedy_Set_Cover(X, F)
1. U ← X
2. C ← Ø
3. while U ≠ Ø
4.     do выбирается подмн. S∈F,
       максимизирующее |S ∩ U|
5.     U ← U - S
6.     C ← C ∪ {S}
7. return C
```



$C = \{S_1, S_4, S_5, S_3\}$

Теорема о точности алгоритма

Обозначение:

$$H(d) = \sum_{i=1}^{d} 1/i$$ d-ое гармоническое число

Теорема: Greedy_Set_Cover - ρ(n) - приближенный алгоритм с полиномиальным временем работы, где

$$\rho(n)=H(\max\{|S| \mid |S| \in F\})$$

Доказательство:
S_i - *i*-ое подмножество, выбранное алгоритмом.
Присвоим стоимость 1.
Если элемент $x \in X$ впервые покрывается подмножеством S , то

$$c_x = \frac{1}{|S_i - (S_1 \cup S_2 \cup \ldots \cup S_{i-1}|}$$

На каждом шаге алгоритма присваивается единичная стоимость, поэтому

$$|C| = \sum_{x \in X} c_x$$

Стоимость, присвоенная оптимальному покрытию, равна

$$\sum_{S \in C^*} \sum_{x \in S} c_x$$

Т.к. каждый $x \in X$ принадлежит хотя бы одному множеству $S \in C^*$

$$\sum_{S \in C^*} \sum_{x \in S} c_x \geq \sum_{x \in X} c_x$$

Получаем

$$|C| \leq \sum_{S \in C^*} \sum_{x \in S} c_x \quad (1)$$

Для любого $S \in F$ выполняется доказательство

$$\sum_{x \in S} c_x \leq H(|S|) \quad (2)$$

Из следует

$$|C| \leq \sum_{S \in C^*} H(|S|) \leq |C^*| \times H(\max\{|S| : S \in F\})$$

Теорема: Greedy _Set_Cover является (Ln|X| + 1)-приблмженным алгоритмом с полиномиальным временем работы.
Доказательство:следует из предыдущей теоремы, с учетом того, что

$$\sum_{k=1}^{n} \frac{1}{k} \le \ln n + 1$$

1.9.9. Задача о сумме подмножества

Постановка

Дано: пара (S,t), где
S - множество $\{x_1, x_2, \ldots, x_n\}$ положительных целых чисел
t - положительное целое число.

Задача: найти подмножество множества S, сумма элементов которого принимает максимально возможное значение, не большее t.

Точный алгоритм (экспоненциальный)

```
Exact_Subset_Sum(S, t)
1. n ← |S|
2. L0 ← <0>
3. for i ← 1 to n
4.      do Li ← Merge_Lists(Li-1,
                 Li-1 + xi)
5.         Из списка Li удаляются
           все элементы, большие t
6. return Максимальный элемент из
   списка Ln
```

Пояснение:
P_i - множество всех значений, которые можно получить, выбрав подмножество $\{x_1, x_2, \ldots, x_i\}$ и просуммировав его элементы.
S={1,4,5}
P_1={0,1}
P_2 ={0,1,4,5}
P_2 ={0,1,4,5,6,9.10}

$P_i = P_{i-1} \cup (P_{i-1} + x_i)$

L_i - отсортированный список, содержащий все xe P_i : $x \le t$

$|L_i| <= 2^i$

Приближенный алгоритм

На основе точного алгоритма разработаем схему аппроксимации с полностью **полиномиальным** временем работы.

Идея: если два значения в списке L мало отличаются друг от друга, то для получения приближенного решения нет смысла явно обрабатывать оба эти значения.

Сократить список L по параметру δ значит удалить из L максимальное количество элементов так, чтобы в полученном L' для каждого удаленного из L элемента *y* содержался z, аппроксимирующий *y*:

$$\frac{y}{1+\delta} \le z \le y$$

Алгоритм сокращения списка

```
Trim (L, δ)
1. m ← |L|
2. L' ← <y1>
3. last ← y1
4. for i ← 2 to m
5.      do if yi > last * (1 + δ)
           //yi ≥ last т.к. список L
           отсортирован
6.           then last ← yi
7. return L'
```

$$\frac{y_i}{1+\delta} \le last \le y_i$$

y_i добавляется в L', если

$$y_i \ge last * (1+\delta)$$

Пример:

δ = 0,1

L = <10, 11, 12, 15, 20, 21, 22, 23, 24, 29>

L' = <10, 12, 15, 20, 23, 29>

Теорема о точности алгоритма

Входные данные: $S = \{x_1, x_2, ..., x_n\}$; t; $0<\varepsilon<1$
Возвращаемое значение z отличается от оптимального не более, чем в 1+ε раз.

```
Approx_Subset_Sum(S, t, ε)
1. n ← |S|
2. L0 ← <0>
3. for i ← 1 to n
4.      do      Li ← Merge_Lists(Li-1, Li-1 + xi)
5.              Li ← Trim(Li, ε/2n)
6.              Из списка Li удаляются все элементы, большие t
7. Пусть z* - максимальное значение в списке Ln
8. return z*
```

Теорема: Approx_Subset_Sum - схема аппроксимации с полностью полиномиальным временем работы, позволяющая решить задачу о сумме подмножества.
Доказательство:
Оптимальное решение $y^* \in P_n$
$z^* \le y^*$
Нужно показать, что $y^*/z^* \le 1 + \varepsilon$

Также нужно показать, что время работы алгоритма выражается полиномиальной функцией и от 1/ε, и от размера входных данных.

Для каждого $y \le t$, $y \in P_i$ существует $z \in L_i$:

$$\frac{y}{(1+\varepsilon/2n)^i} \le z \le y$$

Это неравенство должно выполняться для $y^* \in P_n$, пэтому существует элемент $z \in L_n$:

$$\frac{y^*}{(1+\varepsilon/2n)^n} \le z \le y^*$$

$$\frac{y^*}{z} \le \left(1+\frac{\varepsilon}{2n}\right)^n$$

$$\forall z \in L_n, z^* \ge z \Rightarrow$$

$$\frac{y^*}{z^*} \le \left(1+\frac{\varepsilon}{2n}\right)^n \le e^{\frac{\varepsilon}{2}} \le 1+\frac{\varepsilon}{2}+\left(\frac{\varepsilon}{2}\right)^2 \le 1+\varepsilon$$

Оценим границу длины списка L_i.
Рассмотрим последовательные элементы z и z' в списке L_i после сокращения : $z'/z>1+\varepsilon/2n$.

Каждый список содержит 0, возможно, 1 и до $\lfloor \log_{1+\varepsilon/2n} t \rfloor$ дополнительных значений.
Количество элементов в каждом списке L_t не превышает
$\log_{1+\varepsilon/2n} t + 2 = \ln t \backslash \ln(1+\varepsilon/2n)) + 2 \le$
$\le 2n\,(1+ \varepsilon/2n)^* \ln t)/\varepsilon + 2 \le$
$\le 4n^* \ln t/\varepsilon + 2$
Поскольку время выполнения процедуры Approx_Subset_Sum выражается полиномиальной функцией от длины списка L_i , эта процедура является схемой аппроксимации с полностью полиномиальным временем выполнения.

1.9.10. Рандомизированные приближенные алгоритмы

Рандомизированный $\rho(n)$ приближенный алгоритм имеет коэффициент аппроксимации $\rho(n)$:

$$\max\left(\frac{E(C)}{C^*}, \frac{C^*}{E(C)}\right) \le \rho(n)$$

n - размер входных данных
E(C) - мат. ожидание стоимости решения

C* - стоимость оптимального решения

1.9.11. MAX-3-CNF выполнимость

Постановка задачи CNF - конъюнктивная нормальная форма. В случае 3-CNF в каждых скобках содержится ровно три различных литерала.

Пример:
$$(x_1 \vee \neg x_1 \vee x_2) \wedge (x_3 \vee x_2 \vee x_4) \wedge (\neg x_1 \vee \neg x_3 \vee \neg x_4)$$

Задача: найти присваиваемые переменным значения, при которых максимальное количество подвыражений в скобках принимает значение 1.

Рандомизированный алгоритм

Идея алгоритма: каждой переменной независимо с вероятностью $^1/_2$ присваивается значение 1 и с вероятностью $^1/_2$ - значение 0.

Теорема: Для заданного экземпляра задачи о MAX-3-CNF выполнимости с *n* переменными и *m* выражениями в скобках такой алгоритм является рандомизированным 8/7 - приближенным алгоритмом. **Доказательство:**

Y_i = I {*i*-е подвыражение в скобках выполняется}
Pr {*i*-е подвыражение не выполняется} = $(1/2)^3 = 1/8$
Pr {*i*-е подвыражение выполняется}= 1 - 1/8 = 7/8
$E[Y_i] = 7/8$
$$E[Y] = E[\sum_{i=1}^{m} Y_i] = \sum_{i=1}^{m} E[Y_i] = \sum_{i=1}^{m} 7/8 = 7m/8$$

$\rho(n) = m/(7m/8) = 8/7$

1.9.12. Метод локальных приближений

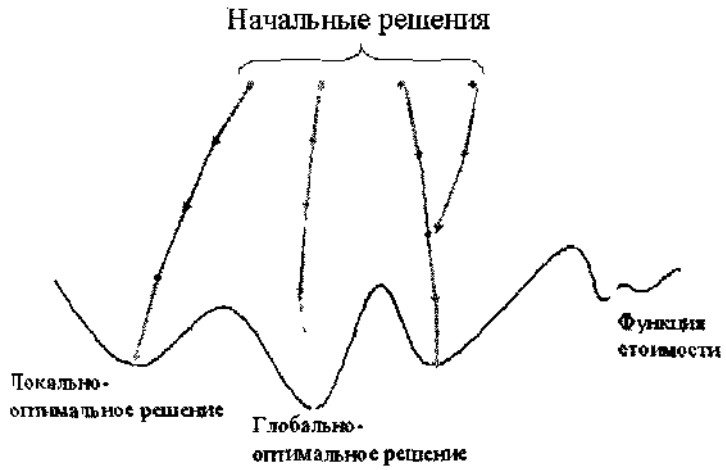
Идея

Начать с произвольного решения.

Для улучшения текущего решения применить к нему какое-либо преобразование из заданной совокупности преобразований. Это улучшенное решение становится текущим решением.

Повторять указанную процедуру до тех пор, пока ни одно из преобразований в заданной совокупности не позволит улучшить текущее решение.

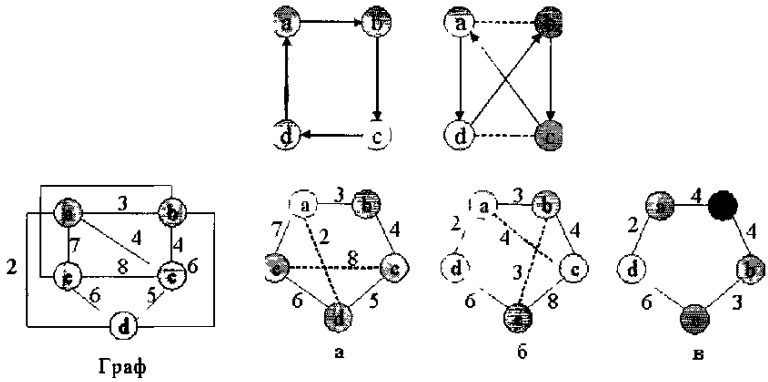
Если заданная совокупность преобразований включает все преобразования (с помощью которых из любого решения можно получить любое другое), то мы получим точное (глобально-оптимальное) решение.



На практике совокупность преобразований ограничивают. С помощью них из ряда произвольных решений получают локально-оптимальные решения и выбирают из них лучшее.

Пример: задача коммивояжера

«Двойной выбор»



k-выбор

Выбор с переменной глубиной

1.9.13. Метод ветвей и границ

Решая дискретную экстремальную задачу, разобьем множество всех возможных вариантов на классы и построим оценки для них.

В результате становится возможным отбрасывать решения целыми классами, если их оценка хуже некоторого рекордного значения.

Дискретная экстремальная (на минимум) задача в общем виде:

Пусть задано дискретное множество A и определенная на нем функция f. Обозначим минимум функции f на X как F(X).

Требуется найти $x_0 \in A$: $f(x_0) = F(A)$

Замечание 1

Пусть $A=(A_0) \bigcup A_1 \bigcup ... \bigcup A_k$, $A_i \cap A_j = \varnothing$, $i \neq j$.

Причем $F(A)<F(A_0)$, т. е. на A_0 минимум не достигается.

Тогда справедливо следующее: $F(A)=\min \{ F(A_i) \mid i \in 1:k \}$.

Замечание 2

Пусть Ф — функция, заданная насовокупности подмножеств множества A так, что $\Phi(X) \leq F(X) \; \forall \; X \subset A$

Пусть x* — произвольный элемент A и пусть f*=f(x*).

Тогда справедливо следующее:

$F(A)=\min \{ f^* \min \{ F(A_i) \mid i \in 1:k, \Phi(A_i) \le f^* \} \}$
Разобьем множество А на подмножества A_i и на каждом из них найдем нижнюю оценку Ф.
Для элементов множества А будем вычислять значения .функции f и запоминать наименьшее в качестве рекордного значения f*.
Все подмножества, у которых оценка выше f*, объединим в подмножество A_0, чтобы в дальнейшем не рассматривать.
Выберем какое-либо из множеств A_i, i>0. Разобьем это множество на более мелкие подмножества. При этом мы будем продолжать улучшать рекордное значение f*.

Этот процесс продолжается до тех пор, пока не будут просмотрены все множества A_i, i>0.

1.9.14. Метод случайного поиска

Обычно выбор решения можно представить последовательностью выборов.
Если делать эти выборы с помощью какого-либо случайного механизма, то решение находится очень быстро, так что можно находить решение многократно и запоминать "рекорд", т. е. наилучшее из встретившихся решений.
Этот наивный подход существенно улучшается, когда удается учесть в случайном механизме перспективность тех или иных выборов, т. е. комбинировать случайный поиск с эвристическим методом и методом локального поиска.

Такие методы применяются, например, при составлении расписаний для Аэрофлота.

2. Вероятностные алгоритмы

2.1. Основные положения теории вероятностей и статистики

Очень много важных задач содержат в той или иной степени неопределенность. Представляющие интерес величины заранее непредсказуемы, а имеют случайный характер, который должен быть отражен в любой корректной модели. Модели такого типа называются *вероятностными* .

В данном разделе мы будем рассматривать некоторые понятия из теории вероятностей и статистики. Ниже кратко описаны некоторые применения этого аппарата к разработке и анализу алгоритмов.

1. *Анализ средней трудоемкости алгоритма.* Предположим, что у нас есть алгоритм (алгоритм *S),* оптимизирующий (в некотором смысле) порядок выполнения заданий на вычислительной машине. К числу важнейших факторов, определяющих время работы алгоритма, относятся общее число *(N)* и типы заданий, ожидающих выполне ния. Если все задания простого типа, то алгоритм S может построить расписание за время *O(N).* Но если все задания очень сложные, то алгоритм S требует $O(N^4)$ времени. Предположим, что между этими крайними случаями имеется 12 различных типов заданий и что пакеты, как правило, содержат смесь из заданий различных типов. Насколько хорош наш алгоритм? Чтобы ответить на этот вопрос, нам нужно знать, каково среднее, представительное множество заданий. Затем мы должны решить, насколько хорошо работает алгоритм в среднем. Отсюда следует необходимость точно определить, что такое «среднее». Возможно, что худший случай $O(N^4)$ для операционной системы неприемлем; например, может оказаться, что вся система будет бездействовать в ожидании, пока подпрограмма не составит расписание. Есди худший случай встречается редко и средняя трудоемкость алгоритма равна $O(N^{3/2})$, то можно использовать алгоритм S в операционной системе. С другой стороны, если средний режим $O(N^4)$, мы, по-видимому, будем вынуждены применить какой-то другой быстрый, грубый, эвристический алгоритм, не гарантирующий оптимального расписания. На этом примере отчетливо виден ряд важных моментов, связанных с анализом алгоритмов.

2. *Моделирование.* Одним из самых важных применений вычислительной машины является ее использование в качестве средства проведения управляемых экспериментов на сложных реальных системах за короткое время и при относительно низких затратах. Для этого строят алгоритм, моделирующий реальную систему. Во многих отношениях это крайнее средство. К моделированию прибегают, как правило, когда задачи безнадежно превышают наши аналитические возможности и сочетают в себе трудоемкость и неопределенность. В сущности, все серьезные модели вероятностны.

3. *Вычислительная статистика.* Вычислительную машину часто используют для обработки и анализа статистической информации. Необходимо разрабатывать и анализировать алгоритмы, выполняющие статистические вычисления аккуратно и эффективно.

4. *Алгоритмы принятия решений.* Вычислительная машина становится все более популярной в мире коммерции и государственного управления как помощник при решении важных вопросов. Все задачи в этой области существенно включают процесс принятия решений в условиях неопределенности.
Рассмотрим, например, следующую задачу. В текущий момент Правительство поддерживает ряд административных и обслуживающих агентств. В условиях катастрофического уменьшения бюджета некоторое число агентств необходимо упразднить или сильно ограничить их в средствах. Имеется огромный объем данных, относящихся к прошлой, настоящей и будущей деятельности этих агентств. Как принять решение? Один из самых обещающих количественных методов разрешения таких проблем известен как *теория статистических решений Байеса.* К сожалению, в этой теории совершенно отсутствуют хорошие вычислительные алгоритмы.
5. *Генерация случайных чисел.* Почти во всех приложениях теории вероятностей и статистики требуется средство внесения в данные неопределенности. Хотелось бы, если возможно, делать это с помощью механизма, внутреннего по отношению к самой вычислительной машине, избегая дорогостоящей и потребляющей много времени внешней активности. Разработка и анализ хороших алгоритмов генерации случайных данных — это серьезная самостоятельная проблема.

Мультипроцессорные системы

Начнем с простой задачи. Имеется мультипроцессорная система с пятью идентичными процессорами. На этой системе нужно пропустить большую сложную программу. Программа требует трех из пяти процессоров и не может быть загружена, если число свободных процессоров меньше трех. Какова вероятность того, что мы сможем загрузить программу в тот момент, когда мы готовы работать с ней? Каждая вероятностная модель формулируется в терминах *эксперимента.* **Эксперимент — это любое точно определенное действие.** В нашем примере эксперимент состоит в том, чтобы определить, сколько в системе свободных процессоров. С каждым экспериментом мы можем связать множество *исходов.* Пометим наши пять процессоров целыми числами от 1 до 5. Один возможный исход эксперимента может быть тогда обозначен вектором с 5 компонентами (1,0, 1,0, 0), что означает состояние системы, когда процессоры 1 и 3 заняты, а процессоры 2, 4 и 5 свободны. Если система находится в этом состоянии, то программа будет принята. Множество всех возможных

исходов называется *пространством элементарных событий* эксперимента. Используя 5-компонентное представление, можно увидеть, что в нашем мультипроцессорном эксперименте имеется 2^5 элементов пространства элементарных событий. Ниже все они явно перечислены:

$s_1 = (0, 0, 0, 0, 0)$
$s_2 = (1, 0, 0, 0, 0)$
$s_3 = (0, 1, 0, 0, 0)$
$s_4 = (0, 0, 1, 0, 0)$
$s_5 = (0, 0, 0, 1, 0)$
$s_6 = (0, 0, 0, 0, 1)$
$s_7 = (1, 1, 0, 0, 0)$
$s_8 = (1, 0, 1, 0, 0)$
$s_9 = (1, 0, 0, 1, 0)$
$s_{10} = (1, 0, 0, 0, 1)$
$s_{11} = (0, 1, 1, 0, 0)$
$s_{12} = (0, 1, 0, 1, 0)$
$s_{13} = (0, 1, 0, 0, 1)$
$s_{14} = (0, 0, 1, 1, 0)$
$s_{15} = (0, 0, 1, 0, 1)$
$s_{16} = (0, 0, 0, 1, 1)$
$s_{17} = (0, 0, 1, 1, 1)$
$s_{18} = (0, 1, 0, 1, 1)$
$s_{19} = (0, 1, 1, 0, 1)$
$s_{20} = (0, 1, 1, 1, 0)$
$s_{21} = (1, 0, 0, 1, 1)$
$s_{22} = (1, 0, 1, 0, 1)$
$s_{23} = (1, 0, 1, 1, 0)$
$s_{24} = (1, 1, 0, 0, 1)$
$s_{25} = (1, 1, 0, 1, 0)$
$s_{26} = (1, 1, 1, 0, 0)$
$s_{27} = (0, 1, 1, 1, 1)$
$s_{28} = (1, 0, 1, 1, 1)$
$s_{29} = (1, 1, 0, 1, 1)$
$s_{30} = (1, 1, 1, 0, 1)$
$s_{31} = (1, 1, 1, 1, 0)$
$s_{32} = (1, 1, 1, 1, 1)$

Пусть $S = \{s_1, s_2, s_3, \ldots\}$— пространство элементарных событий эксперимента. Предположим на время, что S содержит конечное или бесконечное счетное число элементов. ***Событие* — это любое подмножество пространства элементарных событий**. Обычно событие словесно выражается в терминах эксперимента, а затем переводится на язык подмножества *S*. Рассмотрим, например, следующее событие: заняты по крайней мере четыре процессора. В терминах *S* это событие состоит из всех исходов, которые представляют состояния системы, когда используются четыре или пять процессоров, т. е.

$$E = \{(0, 1, 1, 1, 1), (1, 0, 1, 1, 1), (1, 1, 0, 1, 1), (1, 1, 1, 0, 1), (1, 1, 1, 1, 0), (1, 1, 1, 1, 1)\} = \{s_{27}, s_{28}, s_{29}, s_{30}, s_{31}, s_{32}\}.$$

Единичный акт эксперимента называется *испытанием.* Пусть *E* — событие, определенное в пространстве S. Если исход испытания эксперимента принадлежит *E,* то мы говорим, что событие *E произошло.* При каждом испытании может иметь место только один исход *s* в S . Однако тем самым произойдет каждое событие, включающее s. События могут быть сформированы из других событий с помощью стандартных операций над множествами: объединения, пересечения и дополнения. Если

E_1 = событие, состоящее в том, что заняты по крайней мере четыре процессора,

и E_2 = событие, состоящее в том, что заняты самое большее четыре процессора,

то $E_3 = E_1 \cap E_2$=событие, состоящее в том, что заняты в точности

четыре процессора =

$$= \{(0, 1, 1, 1, 1), (1, 0, 1, 1, 1), (1, 1, 0, 1, 1), (1, 1, 1, 0, 1), (1, 1, 1, 1, 0)\} =$$

$$= \{s_{27}, s_{28}, s_{29}, s_{30}, s_{31}\}.$$

При чтении $\cap$ читается как *и,* $\cup$ как *или* и дополнение как *не.*

Если E_1 и E_2 — любые два *непересекающихся события* (т. е. $E_1 \cap E_2 = \varnothing$), то они называются *несовместными.* Если E_1 и E_2 — несовместные события, то невозможно, чтобы при одном и том же испытании произошли оба события. В рассматриваемом примере, если

E_1 = процессор 1 занят

и

E_2 = процессор 1 свободен,

то $E_1 \cap E_2 = \varnothing$. Проверьте это, явно перечислив все элементы E_1 и E_2.

Теперь мы готовы начать обсуждение вероятности. Сначала попытаемся сформулировать стандартное интуитивное понятие вероятности с помощью нашего нового словаря. Нас интересует вероятность того, что данное событие случится в результате эксперимента $\mathscr{E}$. Эксперимент $\mathscr{E}$ повторяется N раз, причем N — большое число. Каждое испытание дает исход из S. Этот исход либо в E, и в этом случае событие E произошло, либо нет. Предположим, что событие E произошло n раз в N испытаниях. В этом случае мы склонны определить *вероятность события E* как

$$P(E) = \frac{n}{N}.$$

Данное число можно интерпретировать как относительную частоту появления события E при выполнении эксперимента $\mathscr{E}$; в этом сущность *частотной интерпретации вероятности.*

Очень важно понять, что частотная интерпретация не является существенной частью *теории вероятностей,* хотя она и очень популярна и полезна *на практике.* Теория вероятностей построена на формальной аксиоматической базе, которую не следует отбрасывать как бесполезную абстракцию чистой математики. *Аксиоматическая теория определяет все правила, используемые при подсчете вероятности.* Как мы увидим, эта теория обладает фундаментальными слабостями, и частотная интерпретация дает один из возможных способов справиться с этими слабостями.

Пусть S — пространство элементарных событий эксперимента $\mathscr{E}$. Пусть P — связанная с S *мера вероятности,* которая ставит в соответствие *некоторым* событиям $E \subseteq S$ действительное число $P(E)$, называемое *вероятностью события E.* Эти вероятности должны удовлетворять трем основным правилам (называемым *аксиомами):*

I $P(E) \geqslant 0$ для любого $E \subseteq S$;
II $P(S)=1$;
III $P(E_1 \cup E_2)=P(E_1)+P(E_2)$, если E_1 и E_2 несовместны, т. е. если $E_1 \cap E_2=\varnothing$.

Как только этим «некоторым» событиям приписываются вероятности, согласующиеся с перечисленными тремя аксиомами, становится возможным (в принципе) вычислять вероятность любого другого события, определенного на *S,* пользуясь приписанными вероятностями и тремя аксиомами.

По-видимому, самый трудный аспект в построении вероятностных моделей — это установление вероятностей событий. Общая теория не говорит нам, как для данной конкретной задачи решить, какие выбрать события и какие им приписать числа. Этот выбор целиком оставляется человеку, анализирующему задачу. Обычно вероятности устанавливаются на базе эксперимента и частотной интерпретации. Другой широко распространенной схемой является схема *равновероятного распределения,* которая описана в следующем разделе. Очень важно понять, что этот выбор при моделировании может сильно повлиять (и влияет!) на полезность самой модели. Теория нейтральна; она будет давать предсказания независимо от выбора конкретных значений. Но, если выбор значений сделан недостаточно точно, предсказания станут неверными и не будут отражать поведение моделируемой задачи из «реального мира».

Вернемся к нашей мультипроцессорной задаче. Предположим, что надежные экспериментальные данные показывают, что каждое из 32 состояний пространства *S* с равной вероятностью может быть фактическим состоянием системы (исходом) в любом испытании. Поэтому перед испытанием у нас нет оснований предпочитать какое-либо однсГсостоя-ние другому. Отметим, что *все* 32 элемента в *S* взаимно исключают друг друга (Напомним, что
$s_{17} \cap s_{18}=(0, 0, 1, 1, 1) \cap (0, 1, 0, 1, 1)=\varnothing$, а не (0, 0, 0, 1, 1).
Это не характеристические векторы ; не забывайте их интерпретацию.).
Используя метод индукции и аксиомы II и III, легко показать (покажите!), что

$$P(S)=P(s_1)+P(s_2)+\ldots+P(s_{31})+P(s_{32})=1$$

(Заметьте, что $S=s_1 \cup s_2 \cup \ldots \cup s_{31} \cup s_{32}$.)

Так как каждое состояние равновероятно, то

$$P(s_1)=P(s_2)=\ldots=P(s_{32})=1/32.$$

Итак, основные значения вероятностей установлены.

Теперь мы можем *решить* нашу мультипроцессорную задачу. Какова вероятность того, что мы сможем загрузить нашу программу, когда она готова к выполнению? Интересующее нас событие — это

E = свободны три или более процессоров.

В терминах элементов множества S

$$E=\{s_1,\ s_2,\ \ldots,\ s_{15},\ s_{16}\}.$$

Следовательно,

$$\begin{aligned} P(E) &= P(s_1 \cup s_2 \cup \ldots \cup s_{16}) = \\ &= P(s_1) + P(s_2) + \ldots + P(s_{16}) = \\ &= \frac{1}{32} + \frac{1}{32} + \ldots + \frac{1}{32} = \frac{16}{32} = \frac{1}{2}. \end{aligned}$$

Таким образом, нам, по-видимому, удастся загрузить программу в половине попыток, учитывая, что мы пользуемся *моделью с равными вероятностями и частотной интерпретацией конечного результата.* Если система работает в более тяжелом режиме и состояниям $s_{27}, \ldots, s_{32}$ соответствуют большие вероятности, чем состояниям $s_1, \ldots, s_{26}$, то теория предскажет меньшие шансы на успех.

Очень заманчиво всегда подходить к решению вероятностных задач, используя основную процедуру, описанную в этом примере. Сформулируем ее:

1. *Идентификация пространства элементарных событий S.* В этой книге S будет часто содержать конечное число элементов. Пытайтесь выбирать S так, чтобы все его элементы были *несовместными* и в *совокупности исчерпывающими,* т. е. чтобы никакие два события не могли произойти одновременно и чтобы при каждом испытании происшедшее событие входило в множество *S.* (Более формально, множество подмножеств $A_1, \ldots, A_n$ в S состоит из несовместных и в совокупности исчерпывающих подмножеств, если $A_i \cap A_j = \emptyset$ для всех $i \neq j$, и $\bigcup_{i=1}^{n} A_i = S$.)

2. *Присваивание вероятностей.* Присвойте вероятности элементам *S.* Это присваивание должно согласовываться с аксиомами I, II и III. Пространство S должно состоять из конечного числа несовместных и в совокупности исчерпывающих элементов. Поэтому вероятности должны быть присвоены элементам S таким образом, чтобы сумма всех присвоенных значений равнялась 1. (Почему сумма должна быть равна 1?)

3. *Идентификация событий решения.* Переведите искомое решение на язык событий из S.

4. *Вычисление искомых вероятностей.* Используя правила вычисления (аксиомы I, II и III), подсчитайте искомые вероятности.

Есть одно настолько важное правило, что мы должны обсудить его очень подробно. Оно известно как *формула условной вероятности.* Пусть *E* и *F* — два события, определенные в пространстве *S,* и пусть *P(E)* и *P(F)* — вероятности, соответствующие этим событиям. Пусть *P(E|F)* обозначает вероятность события *E* при условии, что произошло событие *F.* Как правило, $P(E) \neq P(E|F)$. *Как только мы знаем, что событие F произошло, вычисление вероятности E зависит уже не от всего пространства S; теперь нам нужно рассматривать только те элементы S, которые обусловливают выполнение события F.* Все это учитывается автоматически в формуле $P(E \mid F) = \frac{P(E \cap F)}{P(F)}$ (имеет смысл, только когда $P(F) \neq 0$).

Проверим формулу условной вероятности на нашем мультипроцессорном примере. Предположим, что на пути в машинный зал для запуска программы мы встретили друга, который сказал нам, что он только что занял своей программой первый процессор и что программа будет работать очень долго. Он ничего не сказал о состоянии остальных четырех процессоров. Какова теперь вероятность того, что мы сможем загрузить свою программу? Пусть

E = событие, состоящее в том, что нам удается загрузка, т. е. что свободны три или более трех процессоров;

F = событие, состоящее в том, что процессор 1 занят.

Если событие *F* произошло, то мы знаем, что система должна находиться в одном из следующих 16 состояний: s_2, $s_7 - s_{10}$, $s_{21} - s_{26}$, $s_{28} - s_{32}$. Каждое из этих 16 состояний по-прежнему равновероятно; иначе говоря, у нас нет оснований предпочесть какое-то одно состояние всем остальным. Поэтому вероятность того, что предстоящее испытание будет иметь некоторый конкретный исход из этого множества, равна 1/16, т. е. $P(s_i|F) = \frac{1}{16}$ для $i = 2, 7-10, 21-26$ и $28-32$. В этом ограниченном пространстве исходы s_2, s_7, s_8, s_9 и s_{10} означают появление события *E.* Следовательно, по аксиоме III

$$P(E \mid F) = P(s_2 \mid F) + P(s_7 \mid F) + P(s_8 \mid F) + P(s_9 \mid F) + P(s_{10} \mid F) = \frac{5}{16}.$$

Чтобы воспользоваться формулой условной вероятности, рассматриваем полное пространство *S.* Тогда

$$E = \{s_1, s_2, \ldots, s_{16}\};$$
$$F = \{s_2, s_7 - s_{10}, s_{21} - s_{26}, s_{28} - s_{32}\};$$
$$E \cap F = \{s_2, s_7, s_8, s_9, s_{10}\};$$
$$P(E \cap F) = P(s_2) + P(s_7) + P(s_8) + P(s_9) + P(s_{10}) \text{ (аксиома III)} = \frac{5}{32};$$
$$P(F) = P(s_2) + P(s_7) + \ldots + P(s_{10}) + P(s_{21}) + \ldots + P(s_{26}) + P(s_{28}) + \ldots + P(s_{32}) \text{ (аксиома III)} = \frac{16}{32};$$

и $P(E \mid F) = \frac{P(E \cap F)}{P(F)} = \frac{\frac{5}{32}}{\frac{16}{32}} = \frac{5}{16}$.

А что, если появление события F никак не связано с ожидаемым появлением E? Тогда

$P(E) = P(E \mid F)$.

Но $P(E \mid F) = \frac{P(E \cap F)}{P(F)}$,

поэтому $P(E) = \frac{P(E \cap F)}{P(F)}$, или $P(E \cap F) = P(E) \cdot P(F)$.

Если события E и F таковы, что вероятность их совместного появления равна произведению вероятностей их раздельного появления, то говорят, что эти события *независимы.* Эта формула легко обобщается на случай трех или более взаимно независимых событий, т. е. если множества $A_1, \ldots, A_n$ таковы, что

$P(A_i \cap A_j) = P(A_i) \cdot P(A_j)$ для любых $i \neq j$,

то $P(A_1 \cap A_2 \cap \ldots \cap A_n) = P(A_1) P(A_2) \ldots P(A_n)$.

В большинстве задач имеется возможность определить одну или более действительных величин, которые задают информацию в более удобной форме, чем явное описание событий. Например, целочисленная функция

X = число свободных процессоров

сообщает нам все, что мы хотим знать в нашем мультипроцессорном эксперименте. Искомая конечная вероятность может быть выражена как P (X≥3) и равна вероятности того, что значение X больше или равно 3. Обратите внимание на то, что X в действительности является *функцией,* область определения которой есть пространство S, а область значений — множество {0, 1, 2, 3, 4, 5}. Любая такая *функция* из пространства (и определенной на нем вероятностной меры) задачи на множество действительных чисел называется *случайной переменной.* Фактически все серьезные вычисления вероятностей выполняются в терминах случайных переменных.

Определение вероятностей на области значений случайной переменной называется *распределением.* В нашем мультипроцессорном примере, предполагая, что элементы S равновероятны, мы имеем

$$P(X=0)=\frac{1}{32},$$

$$P(X=1)=\frac{5}{32},$$

$$P(X=2)=\frac{10}{32},$$

$$P(X=3)=\frac{10}{32},$$

$$P(X=4)=\frac{5}{32},$$

$$P(X=5)=\frac{1}{32}.$$

Чтобы понять, как вычисляются эти значения, рассмотрим P(X=4)=P (имеются в точности четыре свободных процессора)

$$=P(s_2 \cup s_3 \cup s_4 \cup s_5 \cup s_6)=$$
$$=P(s_2)+P(s_3)+P(s_4)+P(s_5)+P(s_6)=$$
$$=\frac{1}{32}+\frac{1}{32}+\frac{1}{32}+\frac{1}{32}+\frac{1}{32}=\frac{5}{32}.$$

Случайная переменная, область значений которой состоит из счетного числа возможных значений, называется *дискретной.* Если случайная переменная *Y* должна принимать одно из значений $y_1, y_2, \ldots$, то

$$P(Y=y_i)>0 \quad \text{при} \quad i=1, 2, \ldots;$$
$$P(Y=y)=0 \quad \text{для всех остальных значений } y.$$

Дискретные распределения этого вида называются *массовыми функциями вероятности.* Так как переменная *Y* должна принимать в эксперименте одно из значений y_i, массовая функция должна удовлетворять отношению

$$\sum_i P(Y=y_i)=1.$$

Иначе говоря, с вероятностью 1 (т. е. уверенностью) переменная *Y* должна равняться одному из допустимых значений (заметим, что *Y* не может равняться двум разным значениям одновременно). Легко проверить, что данная массовая функция вероятности случайной величины удовлетворяет этому условию.

Математическое ожидание, или *среднее значение,* или *среднее,* случайной переменной *Y* определяется как

$$E[Y]=\sum y_i P(Y=y_i).$$

Согласно частотной интерпретации вероятности, среднее есть среднее значение случайной величины, во время наблюдения за которой было проведено большое число испытаний. В нашем примере

$$E(X) = \frac{1}{32}(0) + \frac{5}{32}(1) + \frac{10}{32}(2) + \frac{10}{32}(3) + \frac{5}{32}(4) + \frac{1}{32}(5) = \frac{80}{32} = 2.5.$$

Отметим, что среднее не обязательно принадлежит области значений случайной переменной. Любая функция от случайной переменной *G(Y)* также является случайной переменной (почему?), математическое ожидание которой задается формулой

$$E[G(Y)] = \sum_i G(y_i) P(Y = y_i). \qquad (1)$$

(Эту формулу часто называют «законом стихийного статистика», так как она нередко вводится как определение (как сделали и мы) «стихийным» статистиком, забывающим, что эта формула может быть выведена.)

Пусть опять переменная *X* обозначает число свободных процессоров. Если случайная переменная *G* характеризует дефицит процессоров, т. е. равна числу процессоров, которых нам не хватает для того, чтобы запустить нашу программу, то

$$G(X) = \begin{cases} 3, & \text{если } X = 0, \\ 2, & \text{если } X = 1, \\ 1, & \text{если } X = 2, \\ 0, & \text{если } X = 3. \end{cases}$$

Согласно равенству (1), получаем

$$\begin{aligned} E[G(X)] &= 3[P(G=3)] + 2[P(G=2)] + 1[P(G=1)] + 0[P(G=0)] = \\ &= 3[P(X=0)] + 2[P(X=1)] + 1[P(X=2)] + 0[P(X \geqslant 3)] = \\ &= 3\left(\frac{1}{32}\right) + 2\left(\frac{5}{32}\right) + 1\left(\frac{10}{32}\right) = \frac{23}{32}. \end{aligned}$$

Польза понятия математического ожидания увеличивается, когда фактическое значение, принимаемое случайной переменной, как правило, мало отличается от среднего. Одной из мер этого отличия является так называемая *дисперсия* случайной переменной. Пусть *Y* — случайная переменная со средним $\mu = E(Y)$. Тогда дисперсия *Y*, обозначаемая как var *[Y]*, определяется следующим образом:

$$\text{var}[Y] = E[(Y-\mu)^2] = \sum_i (y_i - \mu)^2 P(Y = y_i). \qquad (2)$$

Здесь мы воспользовались формулой математического ожидания для функции от случайной переменной. *Стандартное отклонение* случайной переменной определяется как

$$\sigma[Y] = \sqrt{\text{var}[Y]}.$$

Для случайной переменной *X* в нашем примере

$$\mathbf{var}[X] = (0-2.5)^2\left(\frac{1}{32}\right) + (1-2.5)^2\left(\frac{5}{32}\right) + (2-2.5)^2\left(\frac{10}{32}\right) +$$
$$+ (3-2.5)^2\left(\frac{10}{32}\right) + (4-2.5)^2\left(\frac{5}{32}\right) + (5-2.5)^2\left(\frac{1}{32}\right) = \frac{40}{32} = 1.25,$$

и $\sigma[X] = 1.12$.

Сортировка методом прямого включения

Этот раздел мы завершим анализом математического ожидания, или среднего, для трудоемкости одного простого алгоритма сортировки. Сортировка методом прямого включения работает со списком неупорядоченных положительных целых чисел (обычно называемых ключами), сортируя их в порядке возрастания. Это делается примерно так же, как большинство игроков упорядочивают сданные им карты, поднимая каждый раз по одной карте. Покажем работу общей процедуры на примере следующего неотсортированного списка из восьми целых чисел:

27 412 71 81 59 14 273 87.

Отсортированный список создается заново; вначале он пуст. На каждой итерации первое число неотсортированного списка удаляется из него и помещается на соответствующее ему место в отсортированном списке. Для этого отсортированный список просматривается, начиная с наименьшего числа, до тех пор, пока не находят соответствующее место для нового числа, т. е. пока все отсортированные числа с меньшими значениями не окажутся впереди него, а все числа с большими значениями — после него. Следующая последовательность списков показывает, как это делается:

Итерация 0 Неотсортированный 412 71 81 59 14 273 87
Отсортированный 27

Итерация 1 Неотсортированный *412* 71 81 59 14 273 87
Отсортированный 27 *412*

Итерация 2 Неотсортированный *71* 81 59 14 273 87
Отсортированный 27 *71* 412

Итерация 3 Неотсортированный *81* 59 14 273 87
Отсортированный 27 71 *81* 412

Итерация 4 Неотсортированный *59* 14 273 87
Отсортированный 27 *59* 71 81 412

Итерация 5 Неотсортированный *14* 273 87
Отсортированный *14* 27 59 71 81 412

Итерация 6 Неотсортированный *273* 87
Отсортированный 14 27 59 71 81 *273* 412

Итерация 7 Неотсортированный *87*

Отсортированный 14 27 59 71 81 *87* 273 412

В следующем алгоритме заводится только один список, и переорганизация чисел производится в старом списке.

Algorithm SIS *(Сортировка Прямым включением).* Отсортировать на старом месте последовательность целых чисел $I(1), I(2), \ldots, I(N)$ в порядке возрастания.

Шаг 1. [Основная итерация]

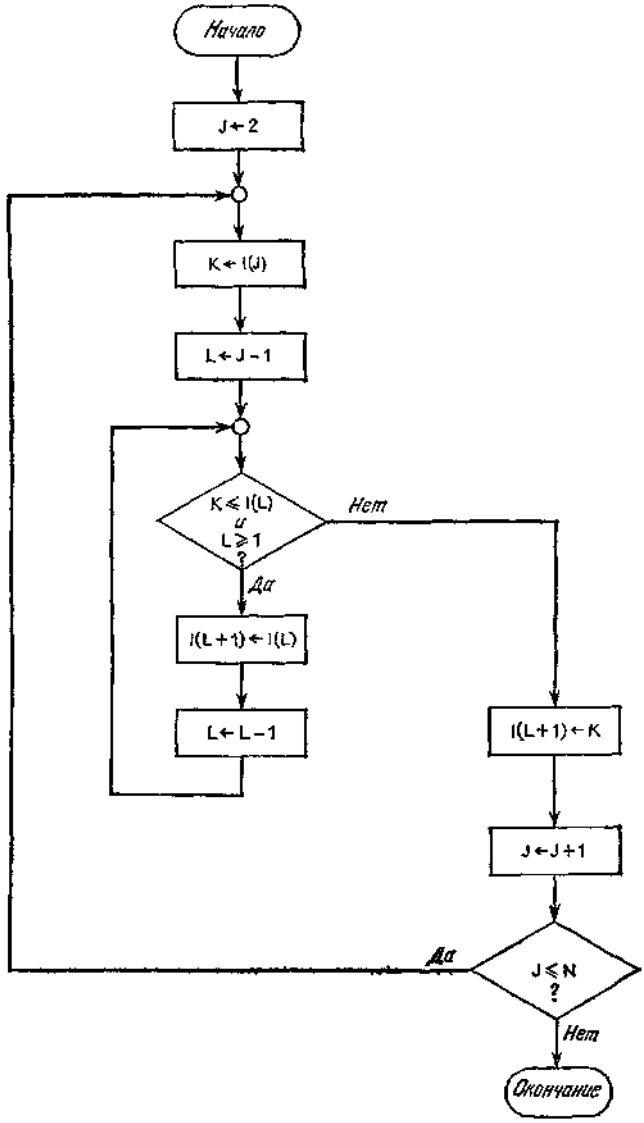
For *J* ← 2 **to N do through шаг** 4 **od; and** STOP.

Шаг 2. [Выбор следующего целого] **Set** *K*← *I(J)*; **and** *L*←*J—1*.

Шаг 3. [Сравнение с отсортированными целыми] **While** *K*<*I(L)* AND L≥1 **do set** *I(L* + 1) ←*I(L)*; **and** *L*←L—1 **od**.

Шаг 4. [Включение] **Set** *I(L+1)* ←*K*.

Ниже приводится блок-схема алгоритма SIS.



Блок-схема алгоритма SIS.

Оценим теперь среднее число сравнений, необходимых для сортировки случайных данных методом прямого включения. В качестве уп-

ражнения мы оставляем определение максимального и минимального числа требуемых сравнений.

Пусть N — число подлежащих сортировке ключей. Рассмотрим i-й проход по циклу на шаге 1. В *начале* этого прохода первые i ключей неотсортированного списка ранее были правильно упорядочены, и теперь мы собираемся включить (i+1)-й ключ в список из i отсортированных ключей. Имеется i—1 «промежутков» между отсортированными ключами и еще две позиции на концах списка, что в сумме дает i+1 возможных позиций для следующего ключа. Предположим, что новый ключ с равной вероятностью 1/(i+1) может быть помещен в любой из этих промежутков. Пусть X_i — случайная переменная, равная числу сравнений, требуемых для помещения ключа i+1 в правильную позицию на этом проходе. Тогда

$$E[X_i] = 1\left(\frac{1}{i+1}\right) + 2\left(\frac{1}{i+1}\right) + 3\left(\frac{1}{i+1}\right) + \ldots + \frac{i}{i+1} + \frac{i}{i+1} =$$
$$= \frac{1}{i+1}[1+2+3+\ldots+(i-1)+i+i] =$$
$$= \frac{1}{i+1}\frac{i(i+1)}{2} + \frac{i}{i+1} \quad \text{(используем формулу Гаусса)} =$$
$$= \frac{i}{2} + \frac{i}{i+1}.$$

Заметим, что, когда нужная позиция находится на дальнем конце, выполняется только i сравнений. Так как имеется N ключей, в цикле на шаге 1 выполняется N—1 итераций. Если случайная переменная Y обозначает общее число сравнений при сортировке, то

$$Y = X_1 + X_2 + \ldots + X_{N-1}.$$

Отметим, что если X_1.....X_n —случайные переменные с совместной массовой функцией $P(x_1, x_2, \ldots, x_n) = P(X_1 = x_1, X_2 = x_2, \ldots, X_n = x_n)$ и если $a_1, \ldots, a_n$ — произвольные константы, то

$$E[a_1X_1 + a_2X_2 + \ldots + a_nX_n] = a_1E[X_1] + \ldots + a_nE[X_n].$$

Используя сказанное выше, получим

$$
\begin{aligned}
E[Y] &= E[X_1] + E[X_2] + \ldots + E[X_{N-1}] = \\
&= \left(\frac{1}{2} + \frac{1}{2}\right) + \left(\frac{2}{2} + \frac{2}{3}\right) + \ldots + \left(\frac{N-1}{2} + \frac{N-1}{N}\right) = \\
&= \frac{1}{2}[1 + 2 + \ldots + (N-1)] + \left[\frac{1}{2} + \frac{2}{3} + \ldots + \frac{N-1}{N}\right] = \\
&= \frac{1}{2}\sum_{i=1}^{N-1} i + \sum_{i=1}^{N-1} \frac{i}{i+1} = \\
&= \frac{1}{2}\frac{N(N-1)}{2} + N - \sum_{i=1}^{N} \frac{1}{i} \quad \text{(см. упр. 2.4.10)} = \\
&= \frac{N^2}{4} + \frac{3N}{4} - H_N,
\end{aligned}
$$

где $H_N = \sum_{i=1}^{N} \frac{1}{i}$ называется N-м *гармоническим числом* (Когда $N \to \infty$, H_N приблизительно равно $\ln N + 0{,}577 + O(1/N)$.)

Таким образом, среднее число сравнений, требуемых алгоритмом SIS для сортировки N ключей, приблизительно равно $N^2/4$, а средняя сложность равна $O(N^2)$. С помощью аналогичного анализа можно обнаружить, что среднее число перемещений $I(L+1) \leftarrow I(L)$ также $O(N^2)$.

На примере сортировки методом прямого включения мы продемонстрировали простой вероятностный анализ сложности.

Чтобы показать основное различие между вероятностным и статистическим подходом, вернемся к нашему анализу ожидаемой трудоемкости алгоритма SIS. По заданному или предполагаемому распределению сортируемых элементов мы вычислили ожидаемое число сравнений, выполняемых при сортировке N ключей. При этом мы воспользовались определениями и правилами вычисления из теории вероятностей. Таким образом, *заданное распределение* было использовано для *предсказания поведения алгоритма.* Этот тип анализа характерен для большинства вероятностных моделей.

Во многих отношениях статистические задачи «обратны» вероятностным задачам. *Наблюдается поведение системы* и делается попытка *вывести заключения о лежащем в основе неизвестном распределении* случайных величин, которые описывают функционирование системы. Определим случайную переменную Y, которая равна для любой случайной последовательности N целых чисел числу сравнений, требуемых алгоритмом SIS для сортировки этой последовательности. Каждое применение алгоритма SIS к случайной последовательности целых чисел дает, таким образом, *выборку* Y. Хотя нам и не известно распределение Y, мы можем взять

несколько выборок, вычислить среднее Y от этих полученных экспериментальным путем чисел и принять Y как оценку математического ожидания Y (т. е. использовать Y для аппроксимации $E[Y]$). Следовательно, для вывода заключений о случайной переменной используются полученные из наблюдений значения. Не представляется возможным дать общие рекомендации относительно того, какая из форм анализа лучше. На практике они должны дополнять друг друга. В нашем примере мы сделали предположение о распределении и затем смогли вычислить $E[Y]$ вероятностными методами. То есть мы предположили, что $(i+1)$-й ключ с равной вероятностью может оказаться помещенным в любую из позиций на концах или внутри отсортированного списка из i элементов. Верно ли это предположение для «реальных» данных? С другой стороны, статистическая оценка для $E[Y]$ может оказаться плохой из-за того, что наблюдаемая выборка была в некотором смысле «непредставительной». Если нам сдали случайным образом четыре карты и все они оказались тузами, будем ли мы считать, что вся колода состоит только из тузов? Если провести оба анализа — сравнить статистическую оценку $\bar{Y}$ с выведенным значением $E[Y]$ — и значения окажутся близкими, будут основания считать, что мы кое-что знаем о средней трудоемкости алгоритма. Если они сильно отличаются, возникает причина для беспокойства.

Продолжим обсуждение оценок и статистического анализа алгоритмов в несколько более общих терминах. Нас интересует ожидаемая или средняя трудоемкость некоторого алгоритма A на множестве возможных исходных данных D. Множество D очень велико, и алгоритм A достаточно сложен. Рассмотрим обработку алгоритмом A некоторых случайным образом выбранных из D исходных данных как эксперимент, который определяет случайную переменную T, время работы алгоритма A. Распределение T неизвестно. Для наших целей не требуется обязательно статистически оценивать все распределение T. Будет достаточно оценить значения некоторых свойств распределения, а именно математическое ожидание и дисперсию.

Общая процедура оценки свойства θ распределения следующая. Возьмем ряд наблюдений $(X_1, X_2, \ldots, X_n)$ за интересующей нас случайной переменной X. Построим функцию $F(X_1, X_2, \ldots, X_n)$, область определения которой есть множество наблюдений, а область значений — множество возможных значений свойства θ. Такая функция называется *оценкой* θ и сама представляет собой случайную переменную (почему?).

Как мы выбираем конкретную функцию в качестве оценки' Поскольку имеется много возможностей выбора, следует дать некоторый

критерий, гарантирующий выбор хорошей оценки. Интуитивно самый очевидный критерий состоит в выборе для оценки такой функции *F*, собственное распределение (напоминаем, что *F* — случайная переменная!) которой сильно концентрируется вокруг истинного значения θ . Рассмотрим три конкретные характеристики хороших оценок, которые отражают это интуитивное требование. Случайная переменная *F* является *несмещенной* оценкой θ , если $E[F]=\theta$. Существуют очень простые несмещенные оценки для математического ожидания и дисперсии любой случайной переменной. Оценка

$$\bar{X}(X_1, \ldots, X_n)=\frac{1}{n}\sum_{i=1}^{n} X_i$$

является несмещенной оценкой математического ожидания *E [X]*. Это следует из

$$\begin{aligned} E[\bar{X}] &= E\left[\frac{1}{n}\sum_{i=1}^{n} X_i\right] = \\ &= \frac{1}{n}\sum_{i=1}^{n} E[X_i] \\ &= \frac{1}{n} n\,(E[X] \text{ (почему?)} \\ &= E[X]. \end{aligned}$$

Так как var *[X]* определяется через *E [X]* (см. уравнение (2) и утверждение: если *X* — произвольная дискретная случайная переменная, то $\mathbf{var}[X]=E[X^2]-(E[X])^2$.), выбранная для var *[X]* оценка будет зависеть от того, знаем ли мы *E [X]* или *E [X]* должно быть оценено. Если значение *E [X]* известно, то оценку

$$G(X_1, \ldots, X_n)=\frac{1}{n}\sum_{i=1}^{n}(X_i-E[X])^2$$

естественно подсказывает определение var *[X]*. Легко установить, что оценка *G* не смещена:

$$\begin{aligned} E[G] &= E\left[\frac{1}{n}\sum_{i=1}^{n}(X_i-E[X])^2\right]=\frac{1}{n}\sum_{i=1}^{n}E[(X_i-E[X])^2]= \\ &= \frac{1}{n}\sum_{i=1}^{n}(E[X_i^2]-2E[X_i]E[X]+(E[X])^2)= \\ &= \frac{1}{n} nE[X^2]-n(E[X])^2=\operatorname{var}[X] \end{aligned}$$

При этом используются следующие результаты: если X — произвольная дискретная случайная переменная, то

$\mathrm{var}[X] = E[X^2] - (E[X])^2$. Если значение $E[X]$ неизвестно, как обычно и бывает, то оценка

$$S^2(X_1, \ldots, X_n) = \frac{1}{n-1}\sum_{i=1}^{n}(X_i - \bar{X})^2$$

является несмещенной оценкой для var $[X]$. Доказательство этого оставлено в качестве упражнения.

Согласующаяся оценка $G(X_1, \ldots, X_n)$ свойства θ — это оценка, значение которой будет как угодно близко к θ с вероятностью, приближающейся к 1, когда $n \to \infty$. Более формально, $G(X_1, \ldots, X_n)$ является согласующейся оценкой для θ, если

$$\lim_{n \to \infty} P\{|G(X_1, \ldots, X_n) - \theta| < \varepsilon\} = 1 \text{ для любого } \varepsilon > 0.$$

Согласующаяся характеристика отражает наше интуитивное ожидание того, что, чем больше выборка, тем она более надежна. Это интуитивное предположение математически подтверждается одним из самых важных результатов в теории вероятностей.

Теорема. *(слабый закон больших чисел). Пусть* $X_1, \ldots, X_n$ — *независимые случайные переменные и* (а) $E[X_i] = m$, (б) $\mathrm{var}[X_i] = q^2$ *для* i=1, 2, ..., n. *Если*

$$\bar{X}(X_1, \ldots, X_n) = \frac{1}{n}\sum_{i=1}^{n} X_i,$$

то для любого $\varepsilon > 0$

$$\lim_{n \to \infty} P\{|\bar{X} - m| < \varepsilon\} = 1.$$

Более сильная формулировка этой теоремы позволяет опустить условие (б) конечности дисперсии. Из закона больших чисел немедленно следует, что $\bar{X}$ — согласующаяся оценка математического ожидания случайной переменной.

Несмещенная оценка с минимальной дисперсией — это несмещенная оценка, дисперсия которой наименьшая среди всех несмещенных оценок или, возможно, среди всех несмещенных оценок определенного рода. Дисперсию часто можно рассматривать как меру точности оценки. Чем меньше дисперсия, тем больше шансы, что значение оценки будет близко к значению свойства, которое она оценивает. Таким образом, малая величина дисперсии является желательной характеристикой оценки.

Пример. Время работы динамически выполняемого алгоритма измеряется с помощью двух системных часов. Показания первых часов T_1,

вторые часы регистрируют время T_2. Те и другие часы неточны в связи с наличием шумов в системе. Но природа шума такова, что каждые часы с равной вероятностью дают отклонения в большую и меньшую сторону на одну и ту же величину. Поэтому

$$E[T_1]=E[T_2]=T=\text{истинное время работы.}$$

Таким образом, и первые, и вторые часы несмещенные, но известно, что их точность различна, т. е.

$$\operatorname{var}[T_1]\neq\operatorname{var}[T_2].$$

Предположим, что $\operatorname{var}[T_1]<\operatorname{var}[T_2]$.

Какую из величин, T_1 или T_2, следует принять для оценки T? Почему бы не использовать и те и другие часы для получения некоторого взвешенного среднего? Соответственно пусть

$$H(T_1, T_2)=sT_1+(1-s)T_2$$

при некоторой константе $0\leqslant s\leqslant 1$, которая пока что не определена. Ясно, что оценка $H(T_1, T_2)$ несмещенная, так как

$$E[H]=sE[T_1]+(1-s)E[T_2]=sT+(1-s)T=T.$$

Константа s будет выбрана так, чтобы получить оценку с наименьшей возможной дисперсией среди всех оценок, записанных в форме простого взвешенного среднего от T_1 и T_2.

Чтобы найти значение s, которое минимизирует var *[H],* поступаем следующим образом, прибегая к стандартному приему дифференциального исчисления, помня, что для любой случайной переменной X и константы a $\operatorname{var}[aX]=a^2\operatorname{var}[X]$.

$$\operatorname{var}[H]=s^2\operatorname{var}[T_1]+(1-s)^2\operatorname{var}[T_2];$$

$$\frac{d}{ds}\operatorname{var}[H]=2s\operatorname{var}[T_1]+(-2+2s)\operatorname{var}[T_2]=0;$$

$$\operatorname{var}[T_2]=s\operatorname{var}[T_1]+s\operatorname{var}[T_2];$$

$$s=\frac{\operatorname{var}[T_2]}{\operatorname{var}[T_1]+\operatorname{var}[T_2]}.$$

Нетрудно проверить, что это значение s действительно минимизирует var *[H].*

Вероятностные алгоритмы могут носить самый разнообразный характер, и, как правило, они всегда создаются исходя из неполной информации о системе процесса обучения.

Вероятностный алгоритм находит ближайшую пару за ожидаемое время O (*n*), т.е. быстрее любого обычного алгоритма.

Вероятностные алгоритмы различают по использованию априорной вероятности (не использованию) и объему применяемой выборки. Эти

алгоритмы решают задачу определения вероятности данного заболевания при наличии данных симптомов. Логические алгоритмы долгое время считали все симптомы одинаково ценными для диагноза. Поэтому возникла идея приписать им веса. Симптомы, характерные для этого заболевания, имели высокий вес, нехарактерные или отсутствующие - низкий, безразличные - промежуточный. Каждый симптом имел один вес как признак заболевания А и другой как признак заболевания В.

Вероятностный алгоритм - алгоритм, все или часть правил которого являются неоднозначными неопределенными и задают поведение системы лишь в статистическом смысле.

Численные вероятностные алгоритмы предназначены для получения приблизительных ответов на некоторые математические вопросы. Чем дольше работает такой алгоритм, тем точнее полученный ответ.

В основе вероятностных алгоритмов лежит идея о том, что иногда лучше попробовать угадать ответ, чем вычислить его. Вероятностные алгоритмы делятся на **четыре класса** - численные, Монте Карло, Лас Вегас и Шервуд - хотя зачастую все вероятностные алгоритмы называются методами Монте Карло. Общим свойством всех этих категорий является то, что чем дольше алгоритм работает, тем лучший результат он выдает.

Доказательства свойств вероятностных алгоритмов требуют хорошего понимания теории вероятностей, тем не менее, понимание этих доказательств отнюдь не обязательно для программистов, использующих алгоритмы. Осмотрительный программист в любом случае проверит утверждения, не зависимо от того, как они обоснованы (например, убеждаясь в качестве генератора случайных чисел или иных особенностей реализации), и, следовательно, сможет использовать эти методы со знанием дела. Рандомизованные BST-деревья - вероятно, простейший способ поддержки АТД заполненной таблицы символов при гарантировании почти оптимальной производительности. Именно поэтому они находят применение во многих практических приложениях.

Четкое формулирование вероятностного алгоритма показателя точности с учетом физических особенностей самого СИ служит предпосылкой для выбора рационального статистического алгоритма контроля.

На практике этот вероятностный алгоритм работает несколько медленнее, чем комбинаторные алгоритмы для нахождения паросочетаний

В некоторых ситуациях вероятностные алгоритмы позволяют получить результаты, которых нельзя достигнуть обычными методами. В нашу задачу не входит исчерпывающее описание этого подхода. Приведенные примеры призваны лишь проиллюстрировать спектр имеющихся возможностей.

Замечание1 . Чтобы получить полиномиальный вероятностный алгоритм проверки простоты числа , нужно дважды применить приведенный алгоритм.

Фаза не имеет аналога в классических вероятностных алгоритмах. Она возникает в квантовой механике, где амплитуда вероятности комплексна. N 2*n* возможных состояний, мы можем осуществить преобразование M на каждом бите независимо, последовательно изменяя состояние системы.

Понятие квантового алгоритма является расширением понятия *вероятностного алгоритма*; в его основе лежит быстрое унитарное преобразование. Было показано, что квантовый алгоритм может вычислять за полиномиальное время преобразование Фурье ф*n* для циклической группы Zn порядка *n* для гладких *n*, а именно, когда *n p* - ре является произведением попарно различных малых простых чисел. Мы оставляем также в стороне вопрос о *вероятностных алгоритмах*, где выбор образа для того или иного слова определяется не однозначно, а случайно, в соответствии с какими-либо вероятностными критериями. Ясно, что применительно к шахматным алгоритмам рассмотрение случайных переходов является весьма уместным.

Вероятности случайных событий, связанных с результатами работы *вероятностных алгоритмов*, будут определяться распределениями используемых в этих алгоритмах случайных битов и явно указанных параметров, если не оговорено противное.

Существуют, однако, и модели обучения, использующие *вероятностные алгоритмы*. Наиболее известной моделью является так называемая машина Бодьцмана, предложенная в 1984 г.

Вычисление вектора невязки системы x в условиях неконтролируемости возмущающих воздействий в *вероятностном алгоритме* следует производить с использованием идеальной эталонной модели.

Конструктивные методы декодирования всегда опираются: математическую структуру кода, а последовательное декодирование *вероятностный алгоритм*, почти всегда пригодный для почти всех случайно выбранных сверточных кодов. Грубо говоря, при последовательном декодировании производится попытка принять решение для каждого полученного символа, но при этом учитываются раннее принятые решения. Если при декодировании последовательности символов на некотором шаге невозможно сделать разумный выбор, декодер возвращается назад и изменяет некоторые из ранее принятых решений.

Автоматизация управления производством требует решения задачи алгоритмизации производства, которое может быть описано либо детерминированным, либо *вероятностным алгоритмом*. Алгоритмы управления производственными процессами подразделяются на следующие основные виды: 1) программное управление; 2) самонастройка; 3) самообучение.

Второй путь повышения достоверности связан с рациональным выбором алгоритма контроля показателя точности, в основу которого должен быть также положен *четкий вероятностный алгоритм*, отражающий физический смысл самого показателя.

Мы будем строить быстрый квантовый алгоритм не для решения задачи факторизации, а для решения другой задачи НАХОЖДЕНИЕ ПЕРИОДА, к которой задача факторизации сводится с помощью *классического вероятностного алгоритма*.

Выше рассмотрены алгоритмы распознавания: детерминированные алгоритмы, основанные на проведении в признаковом пространстве решающей границы (границы, разделяющей классы и представляющей собой некоторую гиперповерхность или гиперплоскость),

вероятностные алгоритмы, основанные на теории статистических решений, алгоритмы вычисления оценок (АВО), логические алгоритмы, базирующиеся на алгебре логики, и, наконец, структурные (лингвистические) алгоритмы.

Решение задачи оптимального построения трансферентной системы учебного процесса возможно только с помощью ЭВМ, так как это связано с множеством разнообразных характеристик, оценок параметров по экспериментальным данным с интерпретацией эксперимента на вспомогательных имитационных моделях и проверкой *многообразных экспериментальных и вероятностных алгоритмов*.

В основе вероятностных алгоритмов лежит идея о том, что иногда лучше попробовать угадать ответ, чем вычислить его. *Вероятностные алгоритмы* делятся на четыре класса - численные, Монте Карло, Лас Вегас и Шервуд - хотя зачастую все вероятностные алгоритмы называются методами Монте Карло. Общим свойством всех этих категорий является то, что чем дольше алгоритм работает, тем лучший результат он выдает.

Вероятностный алгоритм AL для задачи Р использует датчик случайных чисел. После фиксации случайного выбора г алгоритм выполняется (вполне детерминированно.

Численные вероятностные алгоритмы всегда дают ответ, и этот ответ будет тем точнее, чем дольше они работают. Алгоритмы Монте Карло всегда дают ответ, но иногда ответ оказывается неправильным. Чем дольше выполняется алгоритм Монте Карло, тем выше вероятность того, что он даст правильный ответ. Повторный вызов алгоритма Монте Карло также приводит к улучшению результата. Соответственно этим этапам распознавания любой алгоритм распознавания может быть представлен в виде двух последовательно выполняемых алгоритмов В и С. Применительно к *вероятностным алгоритмам* элементами матрицы являются апостериорные вероятности отнесения каждого объекта к соответствующему классу. Алгоритм С - решающее правило - переводит полученную матрицу в матрицу ответов, составленных из символов 1 0, х (1 - объект относится к данному классу, 0 - объект не относится к данному классу, х - не установлено, относится объект к данному классу или не относится), имеющую ту же размерность.

Классический пример задачи из BPP представляет ПРОВЕРКА ПРОСТОТЫ числа: дано число *n*, требуется определить, простое ли оно. Для этой задачи существует *вероятностный алгоритм*, работающий за полиномиальное время.

Значение АПШ, *n* оценивают опытным путем на поверочных установках по определенной программе, предусмотренной применяемой методикой. Оцениваемому значению показателя точности соответствует *обычно вероятностный алгоритм* (учитывающий наличие случайной составляющей), а действительному значению - статистический алгоритм математической обработки исходных результатов контроля.

Любопытен открытый пока вопрос: верно ли, что R P. Заманчиво ответить на него да, исходя из философских соображений, что случайное бросание монеты не может принести много пользы, когда ответ должен быть определенным - да или нет. С ним связан другой вопрос: является ли *вероятностный алгоритм* (показывающий принадлежность задачи классу R) для. В конце концов, вероятностные алгоритмы можно выполнять, используя генераторы псевдослучайных, чисел, имеющиеся для большинства компьютеров, и вероятность ошибки 2 - 100 пренебрежимо мала.

Использование вероятностного подхода возможно и при небольших выборках, однако в этом случае необходимы некоторые оценки информативности выборки.

Практические работы по созданию диагностирующих программных систем на базе *вероятностных алгоритмов* широко велись в 60 - е годы XX века.

Еще одно усовершенствование метода быстрой сортировки заключается в использовании такого разделяющего элемента, который с достаточно большой вероятностью делил бы файл вблизи его середины. Наиболее безопасный выбор, минимизирующий вероятность возникновения наихудшего случая, состоит в использовании в качестве разделяющего элемента случайного элемента массива. Тогда вероятность возникновения наихудшего случая становится ничтожно малой. Этот метод представляет собой пример *вероятностного алгоритма* (probabilistic algorithm) - такого алгоритма, который использует случайный характер величин для достижения высокой

эффективности с большой вероятностью, независимо от степени упорядоченности входных данных. На практике использование в рамках быстрой сортировки генератора случайных чисел с этой целью может оказаться излишним: простой произвольный выбор оказывается достаточно эффективным.

Подведем итоги обсуждению алгоритмов. Численные вероятностные алгоритмы всегда дают ответ, и этот ответ будет тем точнее, чем дольше они работают.

Алгоритмы Монте-Карло всегда дают ответ, но иногда ответ оказывается неправильным. Чем дольше выполняется алгоритм Монте-Карло, тем выше вероятность того, что он даст правильный ответ. Повторный вызов алгоритма Монте-Карло также приводит к улучшению результата.

Алгоритмы Лас Вегаса не могут вернуть неправильного результата, но они могут и не вернуть никакого результата, если им не удалось найти правильный ответ.

Шервудскую технику можно применять к любому детерминированному алгоритму. Она не влияет на правильность алгоритма, а лишь уменьшает вероятность реализации наихудшего поведения. Вероятность наилучшего поведения при этом, правда, тоже понижается.

Вероятностные алгоритмы для NP-полных задач:

Рассмотрим задачу максимальная выполнимость: (MAX-SAT):

Даны m скобок КНФ – конъюктивной нормальной формы с n переменными. Найти значения переменных, максимизирующее число выполненных скобок.

Следующее утверждение дает приближенный вероятностный алгоритм решения методом Монте-Карло:

0,5 приближение: Для любых *m* скобок существуют значения переменных, при которых выполнено не менее *m/2* скобок.

Предположим, что каждая переменная принимает значения 0 или 1 независимо и равновероятно. Пусть для $1 \le i \le m$, $Z_i = 1$ если i-тая скобка выполнена, и $Z_i = 0$, в противном случае.

Для каждой дизъюнкции с k литералами (переменными или их отрицаниями), вероятность, что она не равна 1 при случайном выборе значений переменных равна 2^{-k}. Значит вероятность того, что скобка равна 1, есть $1 - 2^{-k} \ge \frac{1}{2}$. И математическое ожидание $EZ_i \ge \frac{1}{2}$. Отсюда математическое ожидание числа выполненных скобок (равных 1) равно $\sum_{i=1}^{m} EZ_i \ge \frac{m}{2}$. Это означает, что есть такой набор значений переменных, что выполняется $\sum_{i=1}^{m} EZ_i \ge \frac{m}{2}$.

Определение. ***Вероятностный (рандомизированный) приближенный алгоритм*** имеет коэффициент аппроксимации p(n), если:

$$\max\left(\frac{E(C)}{C^*}, \frac{C^*}{E(C)}\right) \le p(n),$$

n - размер входных данных;

E(C) – математическое ожидание стоимости решения:

C^* - стоимость оптимального решения.

Таким образом, описанный приближенный вероятностный алгоритм для задачи максимальная выполнимость дает точность 1/2.

Общая схема применения вероятностных алгоритмов:

1. Найти вероятностный алгоритм, работающий за полиномиальное время

2. Оценить вероятность успеха p.

3. Повторять исходный алгоритм $\frac{c}{p}$ раз.

4. Итоговая вероятность станет константой: $(1-p)^{c/p} \le e^{-c}$.

2.2. Алгоритмы Монте-Карло

Метод Монте-Карло

Ме́тод Мо́нте-Ка́рло (методы Монте-Карло, ММК) — общее название группы численных методов, основанных на получении большого числа реализаций стохастического (случайного) процесса, который формируется таким образом, чтобы его вероятностные характеристики совпадали с аналогичными величинами решаемой задачи.
Используется для решения задач в различных областях физики, химии, математики, экономики, оптимизации, теории управления и др.

Алгоритм Бюффона для определения числа Пи

Случайные величины использовались для решения различных прикладных задач достаточно давно. Примером может служить способ определения числа Пи, который был предложен Бюффоном еще в 1777 году. Суть метода была в бросании иглы длиной L на плоскость,

расчерченную параллельными прямыми, расположенными на расстоянии *r* друг от друга (см. рис. 1).

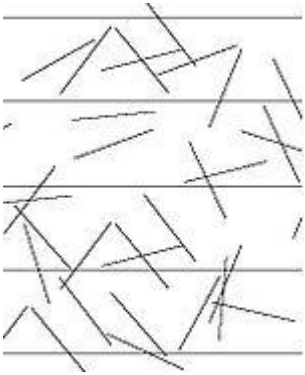


Рис. 1. Метод Бюффона

Вероятность (как видно из дальнейшего контекста, речь идёт не о вероятности, а о математическом ожидании количества пересечений за один опыт; вероятностью это становится лишь при условии, что $r>L$) того, что отрезок пересечет прямую, связана с числом Пи:

$$p = \int_0^{\pi} \int_0^{l\sin\theta} \frac{1}{r\pi} dA d\theta,$$

где

- A — расстояние от начала иглы до ближайшей к ней прямой;
- θ — угол иглы относительно прямых.

Этот интеграл просто взять: $p = \frac{2L}{r\pi}$ (при условии, что $r>L$), поэтому подсчитав долю отрезков, пересекающих прямые, можно приближенно определить это число. При увеличении количества попыток точность получаемого результата будет увеличиваться.

В 1864 году капитан Фокс, выздоравливая после ранения, чтобы как-то занять себя, реализовал эксперимент по бросанию иглы. Результаты представлены в следующей таблице:

	Число бросаний	**Число пересечений**	**Длина иглы**	**Расстояние между прямыми**	**Вращение**	**Значение Пи**	**Ошибка**
Первая попытка	500	236	3	4	отсутствует	3.1780	$+3{,}6 \times 10^{-2}$
Вторая попытка	530	253	3	4	присутствует	3.1423	$+7{,}0 \times 10^{-4}$
Третья попытка	590	939	5	2	присутствует	3.1416	$+4{,}7 \times 10^{-5}$

Комментарии:

- Длины указаны в дюймах;
- Вращение плоскости применялось (и как показывают результаты — успешно) для того, чтобы уменьшить систематическую ошибку;
- В третьей попытке длина иглы была больше расстояния между линиями, что позволило, не увеличивая числа бросаний, эффективно увеличить число событий и повысить точность (за одно бросание могло возникнуть несколько пересечений).

Связь стохастических процессов и дифференциальных уравнений

Создание математического аппарата стохастических методов началось в конце XIX века. В 1899 году лорд Релей показал, что одномерное случайное блуждание на бесконечной решётке может давать

приближенное решение одного из видов параболического дифференциального уравнения. А. Н. Колмогоров в 1931 году дал большой толчок развитию стохастических подходов к решению различных математических задач, поскольку он сумел доказать, что цепи Маркова связаны с некоторыми интегро-дифференциальными уравнениями. В 1933 году И. Г. Петровский показал, что случайное блуждание, образующее Марковскую цепь, асимптотически связано с решением эллиптического дифференциального уравнения в частных производных. После этих открытий стало понятно, что стохастические процессы можно описывать дифференциальными уравнениями и, соответственно, исследовать при помощи хорошо на тот момент разработанных математических методов решения этих уравнений.

Рождение метода Монте-Карло в Лос-Аламосе

Сначала Энрико Ферми в 1930-х годах в Италии, а затем Джон фон Нейман и Станислав Улам в 1940-х в Лос-Аламосе предположили, что можно использовать связь между стохастическими процессами и дифференциальными уравнениями «в обратную сторону». Они предложили использовать стохастический подход для аппроксимации многомерных интегралов в уравнениях переноса, возникших в связи с задачей о движении нейтрона в изотропной среде.

Идея была развита Уламом, который, раскладывая пасьянсы во время выздоровления после болезни, задался вопросом, какова вероятность того, что пасьянс сложится. Вместо того, чтобы использовать обычные для подобных задач соображения комбинаторики, Улам предположил, что можно просто поставить эксперимент большое число раз и, подсчитав число удачных исходов, оценить вероятность. Он же предложил использовать компьютеры для расчётов методом Монте-Карло.

Появление первых электронных компьютеров, которые могли с большой скоростью генерировать псевдослучайные числа, резко расширило круг задач, для решения которых стохастический подход оказался более эффективным, чем другие математические методы. После этого произошёл большой прорыв, и метод Монте-Карло применялся во многих задачах, однако его использование не всегда было оправдано из-за большого количества вычислений, необходимых для получения ответа с заданной точностью.

Годом рождения метода Монте-Карло считается 1949 год, когда в свет выходит статья Метрополиса и Улама «Метод Монте-Карло». Название метода происходит от названия коммуны в княжестве Монако, широко известного своими многочисленными казино, поскольку именно рулетка является одним из самых широко известных генераторов случайных чисел. Станислав Улам пишет в своей автобиографии «Приключения математика», что название было предложено Николасом Метрополисом в честь его дяди, который был азартным игроком. В 1950-х годах метод использовался для расчётов при разработке водородной бомбы. Основные заслуги в развитии метода в это время принадлежат сотрудникам лабораторий ВВС США и корпорации RAND. Одними из первых Метод Монте-Карло для расчёта ливней частиц применили физики А. А. Варфоломеев и И. А. Светлолобов. В 1970-х годах в новой области математики — теории вычислительной сложности было показано, что существует класс задач, сложность (количество вычислений, необходимых для получения точного ответа) которых растёт с размерностью задачи экспоненциально. Иногда можно, пожертвовав точностью, найти алгоритм, сложность которого растёт медленнее, но есть большое количество задач, для которого этого нельзя сделать (например, задача определения объёма выпуклого тела в *n*-мерном евклидовом пространстве) и метод Монте-Карло является единственной возможностью для получения достаточно точного ответа за приемлемое время. В настоящее время основные усилия исследователей направлены на создание эффективных Монте-Карло алгоритмов различных физических, химических и социальных процессов для параллельных вычислительных систем.

Интегрирование методом Монте-Карло

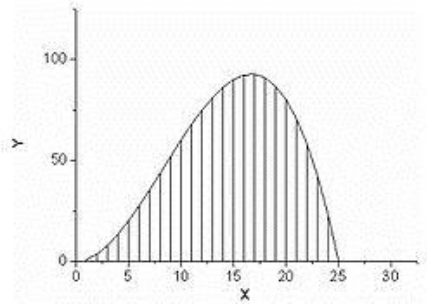


Рис 2. Численное интегрирование функции детерминистическим методом

Предположим, необходимо взять интеграл от некоторой функции. Воспользуемся неформальным геометрическим описанием интеграла и будем понимать его как площадь под графиком этой функции.

Для определения этой площади можно воспользоваться одним из обычных численных методов интегрирования: разбить отрезок на подотрезки, подсчитать площадь под графиком функции на каждом из них и сложить. Предположим, что для функции, представленной на рисунке 2, достаточно разбиения на 25 отрезков и, следовательно, вычисления 25 значений функции. Представим теперь, мы имеем дело с *n*-мерной функцией. Тогда нам необходимо 25^n отрезков и столько же вычислений значения функции. При размерности функции больше 10 задача становится огромной. Поскольку пространства большой размерности встречаются, в частности, в задачах теории струн, а также многих других физических задачах, где имеются системы со многими степенями свободы, необходимо иметь метод решения, вычислительная сложность которого бы не столь сильно зависела от размерности. Именно таким свойством обладает метод Монте-Карло.

Обычный алгоритм Монте-Карло интегрирования

Предположим, требуется вычислить определённый интеграл $\int_a^b f(x)dx$

Рассмотрим случайную величину u , равномерно распределённую на отрезке интегрирования [*a*,*b*]. Тогда *f(u)* также будет случайной величиной, причём её математическое ожидание выражается как

$$Ef(x) = \int_a^b f(x)\varphi(x)dx,$$

где φ(*x*) — плотность распределения случайной величины *u*, равная $\frac{1}{b-a}$ на участке [*a*,*b*].

Таким образом, искомый интеграл выражается как

$$\int_a^b f(x)dx = (b-a)Ef(u)\cdot$$

Но математическое ожидание случайной величины *f(u)* можно легко оценить, смоделировав эту случайную величину и посчитав выборочное среднее.

Итак, бросаем N точек, равномерно распределённых на [*a,b*], для каждой точки u_i вычисляем $f(u_i)$. Затем вычисляем выборочное среднее: $\frac{1}{N}\sum_{i=1}^{N} f(u_i)$.

В итоге получаем оценку интеграла: $\int_a^b f(x)dx \approx \frac{b-a}{N}\sum_{i=1}^{N} f(u_i)$

Точность оценки зависит только от количества точек N.

Этот метод имеет и геометрическую интерпретацию. Он очень похож на описанный выше детерминистический метод, с той разницей, что вместо равномерного разделения области интегрирования на маленькие интервалы и суммирования площадей получившихся «столбиков» мы забрасываем область интегрирования случайными точками, на каждой из которых строим такой же «столбик», определяя его ширину как $\frac{b-a}{N}$, и суммируем их площади.

Геометрический алгоритм Монте-Карло интегрирования

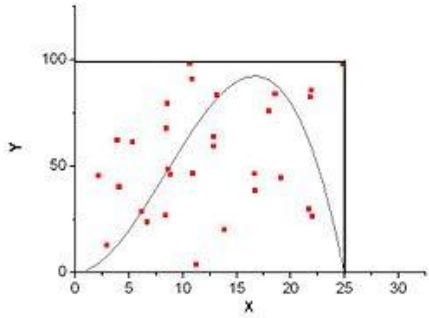


Рис. 3. Численное интегрирование функции методом Монте-Карло

Для определения площади под графиком функции можно использовать следующий стохастический алгоритм:

- ограничим функцию прямоугольником (*n*-мерным параллелепипедом в случае многих измерений), площадь которого S_{par} можно легко вычислить; *любая сторона прямоугольника содержит хотя бы 1 точку графика функции, но не пересекает его;*
- «набросаем» в этот прямоугольник (параллелепипед) некоторое количество точек (N штук), координаты которых будем выбирать случайным образом;
- определим число точек (K штук), которые попадут под график функции;
- площадь области, ограниченной функцией и осями координат, $S = S_{par}\frac{K}{N}$ даётся выражением

Для малого числа измерений интегрируемой функции производительность Монте-Карло интегрирования гораздо ниже, чем производительность детерминированных методов. Тем не менее, в некоторых случаях, когда функция задана неявно, а необходимо определить область, заданную в виде сложных неравенств, стохастический метод может оказаться более предпочтительным.

Использование выборки по значимости

При том же количестве случайных точек, точность вычислений можно увеличить, приблизив область, ограничивающую искомую функцию, к самой функции. Для этого необходимо использовать случайные величины с распределением, форма которого максимально близка к форме интегрируемой функции. На этом основан один из методов улучшения сходимости в вычислениях методом Монте-Карло: выборка по значимости.

Различные вариации метода Монте-Карло можно использовать для решения задач оптимизации. Например, алгоритм имитации отжига.

Компьютерное моделирование играет в современной физике важную роль и метод Монте-Карло является одним из самых распространённых

во многих областях от квантовой физики до физики твёрдого тела, физики плазмы и астрофизики.

Алгоритм Метрополиса

Традиционно метод Монте-Карло применялся для определения различных физических параметров систем, находящихся в состоянии термодинамического равновесия. Предположим, что имеется набор W(S) возможных состояний физической системы S. Для определения среднего значения $\overline{A}$ некоторой величины A необходимо рассчитать $\overline{A} = \sum_S A(S)P(S)$, где суммирование производится по всем состояниям S из W(S). P(S) -вероятность состояния S.

Динамическая (кинетическая) формулировка

Прямое моделирование методом Монте-Карло

Прямое моделирование методом Монте-Карло какого-либо физического процесса подразумевает моделирование поведения отдельных элементарных частей физической системы. По сути это прямое моделирование близко к решению задачи из первых принципов, однако обычно для ускорения расчётов допускается применение каких-либо физических приближений. Примером могут служить расчёты различных процессов методом молекулярной динамики: с одной стороны система описывается через поведение её элементарных составных частей, с другой стороны, используемый потенциал взаимодействия зачастую является эмпирическим.

Примеры прямого моделирования методом Монте-Карло:

- Моделирование облучения твёрдых тел ионами в приближении бинарных столкновений.
- Прямое Монте-Карло моделирование разреженных газов.
- Большинство кинетических Монте-Карло моделей относятся к числу прямых (в частности, исследование молекулярно-пучковой эпитаксии).

Квантовый метод Монте-Карло

Квантовый метод Монте-Карло широко применяется для исследования сложных молекул и твёрдых тел. Это название объединяет несколько разных методов. Первый из них это вариационный метод Монте-Карло, который по сути является численным интегрированием многомерных интегралов, возникающих при решении уравнения Шрёдингера. Для решения задачи, в которой участвует 1000 электронов, необходимо взятие 3000-мерных интегралов, и при решении таких задач метод Монте-Карло имеет огромное преимущество в производительности по сравнению с другими численными методами интегрирования. Другая разновидность метода Монте-Карло — это диффузионный метод Монте-Карло.

2.3.Сущность метода Монте-Карло и моделирование случайных величин

Метод Монте-Карло – это численный метод решения математических задач при помощи моделирования случайных величин.

Датой рождение метода Монте-Карло принято считать 1949 г., когда появилась статья под названием «Метод Монте-Карло» (Н. Метрополис, С. Улам). Создателями этого метода считают американских математиков Дж. Неймана и С. Улама.

Однако теоретическая основа метода была известна давно. Кроме того, некоторые задачи статистики рассчитывались иногда с помощью случайных выборок, т.е. фактически методом Монте-Карло. Однако до появления ЭВМ этот метод не мог найти сколько-нибудь широкого применения, так как моделировать случайные величины вручную – очень трудоёмкая работа. Таким образом, возникновение метода Монте-Карло как весьма универсального численного метода стало возможным только благодаря появлению ЭВМ.

Первоначально метод Монте-Карло использовался главным образом для решения задач нейтронной физики, где традиционные численные методы оказались малопригодными. Далее его влияние распространилось на широкий круг задач статистической физики, очень разных по своему содержанию. К разделам науки, где всё в

большей мере используется метод Монте-Карло, следует отнести задачи теории массового обслуживания, задачи теории игр и математической экономики, задачи теории передачи сообщений при наличии помех и ряд других.

Метод Монте-Карло оказал и продолжает оказывать существенное влияние на развитие методов вычислительной математики и при решении многих задач успешно сочетается с другими вычислительными методами и дополняет их. Его применение оправдано в первую очередь в тех задачах, которые допускают теоретико-вероятностное описание. Это объясняется как естественность получения ответа с некоторой заданной вероятностью в задачах с вероятностным содержанием, так и существенным упрощением процедуры решения.

В подавляющем большинстве задач, решаемых методами Монте-Карло, вычисляют математические ожидания некоторых случайных величин. Так как чаще всего математические ожидания представляют собой обычные интегралы, в том числе и кратные, то центральное положение в теории методов Монте-Карло занимают методы вычисления интегралов.

2.3.1. Теоретическая часть

1 Сущность метода Монте-Карло и моделирование случайных величин

Предположим, что нам необходимо вычислить площадь плоской фигуры S. Это может быть произвольная фигура, заданная графически или аналитически (связная или состоящая из нескольких частей). Пусть это будет фигура, заданная на рис. 1.1.

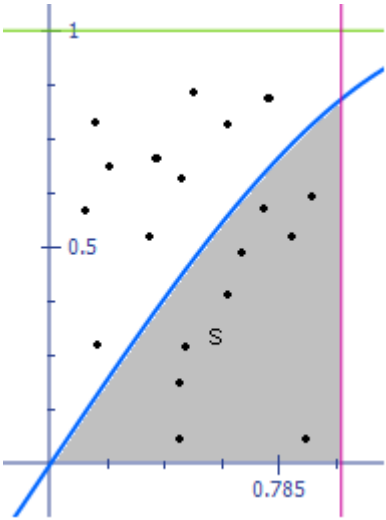


Рис. 1.1

Предположим, что эта фигура расположена внутри единичного квадрата.

Выберем внутри квадрата N случайных точек. Обозначим через N_1 число точек, попавших внутрь фигуры S. Геометрически видно, что площадь фигуры S приближенно равна отношению N_1/N. Причем, чем больше число N, тем больше точность этой оценки.

Для того чтобы выбирать точки случайно, необходимо перейти к понятию случайная величина. Случайная величина ε непрерывная, если она может принимать любое значение из некоторого интервала (a, b).

Непрерывная случайная величина ε определяется заданием интервала (a, b), содержащего возможные значения этой величины, и функции $p(x)$, которая называется плотностью вероятностей случайной

величины ε (плотностью распределения ε). Физический смысл $p(x)$ следующий: пусть (a', b') - произвольный интервал, такой что $a \le a' < b' \le b$, тогда вероятность того, что ε окажется в интервале (a', b'), равна интегралу

$$P\{a' < \varepsilon < b'\} = \int_{a'}^{b'} p(x)dx \qquad (1.1)$$

Множество значений ε может быть любым интервалом (возможен случай $a = -\infty, b = \infty$). Однако плотность $p(x)$ должна удовлетворять двум условиям:

1) плотность $p(x)$ положительна:

$p(x) > 0$; (1.2)

2) интеграл от плотности $p(x)$ по всему интервалу (a, b) равен 1:

$$\int_a^b p(x)dx = 1 \qquad (1.3)$$

Математическим ожиданием непрерывной случайной величины называется число

$$M\varepsilon = \int_a^b xp(x)dx \qquad (1.4)$$

Дисперсией непрерывной случайной величины называется число:

$$D = \int_a^b (x - M\varepsilon)^2 \, p(x) dx$$

Нормальной случайной величиной называется случайная величина ζ, определённая на всей оси $(-\infty, \infty)$ и имеющая плотность

$$p(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left\{-\frac{(x-a)^2}{2\sigma^2}\right\} \quad (1.5)$$

где a, σ - числовые параметры

Любые вероятности вида $P\{x' < \zeta < x''\}$ легко вычисляются с помощью таблицы, в которой приведены значения функции

$$\Phi(x) = \frac{2}{\sqrt{2\pi}} \int_0^x e^{-t^2/2} dt$$

, называемой обычно интегралом вероятностей.

Согласно (1.1)

$$P\{x' < \zeta < x''\} = \frac{1}{\sigma\sqrt{2\pi}} \int_{x'}^{x''} \exp\left\{-\frac{(x-a)^2}{2\sigma^2}\right\} dx$$

В интеграле сделаем замену переменной $x - a = \sigma t$, тогда получим

$$P\{x' < \zeta < x''\} = \frac{1}{\sqrt{2\pi}} \int_{t_1}^{t_2} e^{-t^2/2} dt$$

,

где $t_1 = (x' - a)/\sigma, t_2 = (x'' - a)/\sigma$ Отсюда следует, что $P\{x' < \zeta < x''\} = 0.5\left[\Phi(t_2) - \Phi(t_1)\right]$. Также $\Phi(-x) = -\Phi(x)$

Нормальные случайные величины очень часто встречаются при исследовании самых различных по своей

природе вопросов.

Выбрав $x' = a - 3\sigma$, $x'' = a + 3\sigma$, найдём $t_1 = -3, t_2 = 3$. Следовательно,

$$P\{a - 3\sigma < \zeta < a + 3\sigma\} = \Phi(3) = 0.997 \quad (1.6)$$

Вероятность 0.997 настолько близка к 1, что иногда последнюю формулу интерпретируют так: при одном испытании практически невозможно получить значение ζ, отличающееся от $M\zeta$ больше чем на 3σ.

Проводя большое количество опытов, и получая большое количество случайных величин можно воспользоваться центральной предельной теоремой теории вероятностей. Эта теорема впервые была сформулирована П. Лапласом. Обобщением этой теоремы занимались многие выдающиеся математики, в том числе П.Л. Чебышёв, А.А. Марков, А.М. Ляпунов. Её доказательство достаточно сложно.

Рассмотрим N одинаковых независимых случайных величин $\varepsilon_1, \varepsilon_2, \ldots, \varepsilon_N$, так что распределения вероятностей этих величин совпадают. Следовательно, их математические ожидания и дисперсии также совпадают. Величины эти могут быть как непрерывными, так и дискретными.

Обозначим

$$M\varepsilon_1 = M\varepsilon_2 = ... = M\varepsilon_N = m$$
$$D\varepsilon_1 = D\varepsilon_2 = ... = D\varepsilon_N = b^2$$

Сумму всех этих величин обозначим через $\rho_N = \varepsilon_1 + \varepsilon_2 + ... + \varepsilon_N$

Используя соотношения

$$M(\varepsilon + \eta) = M\varepsilon + M\eta$$
$$D(\varepsilon + \eta) = D\varepsilon + D\eta$$

получаем

$$M\rho_N = M(\varepsilon_1 + \varepsilon_2 + ... + \varepsilon_N) = Nm,$$
$$D\rho_N = D(\varepsilon_1 + \varepsilon_2 + ... + \varepsilon_N) = Nb^2$$

Рассмотрим теперь нормальную случайную величину ζ_N с такими же параметрами: $a = Nm, \sigma = b\sqrt{N}$.

В центральной предельной теореме утверждается, что для любого интервала (a', b') при больших N

$$P\{a' < \rho_N < b'\} \approx \int_{a'}^{b'} \rho_{\zeta_N}(x)dx$$

Смысл этой теоремы в том, что сумма ρ_N большого числа одинаковых случайных величин приближенно нормальна. На самом деле эта теорема справедлива при гораздо более широких условиях: все слагаемые не обязаны быть одинаковыми и независимыми; существенно только, чтобы отдельные слагаемые не играли большой роли в сумме. Эта теорема оправдывает часто встречающиеся нормальные случайные величины. В самом деле, когда встречается суммарное воздействие большого числа незначительных случайных

факторов, результирующая случайная величина оказывается нормальной.

Используя эти данные из теории вероятностей можно перейти к описанию общей схемы метода Монте-Карло. Допустим, что требуется вычислить какую-то неизвестную величину m. Попытаемся придумать такую случайную величину ε, чтобы $M\varepsilon = m$. Пусть при этом $D\varepsilon = b^2$.

Рассмотрим N независимых случайных величин $\varepsilon_1, \varepsilon_2, ..., \varepsilon_N$ распределения которых совпадают с распределением ε. Если N достаточно велико, то, согласно центральной предельной теореме, распределение суммы $\rho_N = \varepsilon_1 + \varepsilon_2 + ... + \varepsilon_N$ будет приблизительно нормальным с параметрами $a = Nm, \sigma = b\sqrt{N}$. Из (1.6) следует, что $P\left\{Nm - 3b\sqrt{N} < \rho_N < Nm + 3b\sqrt{N}\right\} \approx 0.997$.

Последнее соотношение перепишем в виде:

$$P\left\{\left|\frac{1}{N}\sum_{j=1}^{N}\varepsilon_j - m\right| < \frac{3b}{\sqrt{N}}\right\} \approx 0.997 \qquad (1.7)$$

Это соотношение даёт и метод расчёта m, и оценку погрешности.

В самом деле, найдём N значений случайной величины ε. Из (1.7) видно, что среднеарифметическое этих значений будет приближенно равно m. С большой вероятностью погрешность приближения не превосходит величины $3b/\sqrt{N}$. Эта погрешность стремится к нулю с ростом N. На практике часто используют не оценку сверху $3b/\sqrt{N}$, а на вероятную ошибку, которая приближенно равна $r_N = 0.6745b/\sqrt{N}$ Именно такой обычно порядок фактической погрешности расчёта, которая равна

$$\left|\frac{1}{N}\sum_{j=1}^{N}\varepsilon_j - m\right|.$$

Для получения случайных чисел используют обычно три способа: таблицы случайных величин, генераторы случайных чисел и метод псевдослучайных чисел.

Таблицы случайных чисел используют предпочтительно при расчётах вручную. Определяющую роль в этом играют два факта: 1) при использовании ЭВМ легче и удобней воспользоваться генератором случайных чисел, получаемых тут же, чем загружать из памяти значения таблицы, которая к тому же, будет занимать там место. 2) При подсчёте вручную нет необходимости использовать ЭВМ, так как часто необходимо выяснить лишь порядок искомой величины.

Генераторы случайных чисел анализируют какой-либо процесс, доступный для них (шумы в электронных лампах, скачки напряжения) и составляют последовательность из 0 и 1, из которых составляются числа с определёнными разрядами, однако такой метод получения случайных величин имеет свои недостатки. Во-первых, трудно проверить вырабатываемые числа. Проверки приходится делать периодически, так как из-за каких-либо неисправностей может возникнуть так называемый дрейф распределения (нули и единицы в каком-либо из разрядов станут появляться не одинаково часто). Во-вторых, обычно все расчёты на ЭВМ проводятся несколько раз, чтобы исключить возможность сбоя. Но воспроизвести те же самые случайные числа невозможно, если их только не запоминать по ходу счёта. А если запоминать, то снова появляется случай таблиц.

Таким образом, самым эффективным способом получения случайных чисел – это использование псевдослучайных чисел.

Числа, получаемые по какой-либо формуле и имитирующие значения случайной величины γ, называются псевдослучайными числами.

Первый алгоритм для получения псевдослучайных чисел был предложен Дж. Нейманом. Он называется

методом середины квадратов.

Пусть задано 4-значное число $\gamma_0 = 0.8754$. Возведём его квадрат. Получим 8-значное число $\gamma^2_0 = 0.76632516$. Выберем 4 средние цифры этого числа и положим $\gamma_1 = 0.6325$.Далее $\gamma_1^2 = 0.40005625$ и т.д.

Но этот алгоритм не оправдал себя, так как получается слишком много малых значений. Поэтому были разработаны другие алгоритмы. Наибольшее распространение получил алгоритм, называемый методом сравнений (Д. Лемер): определяется последовательность целых чисел m_k, в которой начальное число $m_0 = 1$ задано, а все последующие числа $m_1, m_2, \ldots$ вычисляются по одной и той же формуле

$$m_{k+1} = 5^{17} m_k (\mathrm{mod}\, 2^{40}) \text{ при } k = 0, 1, 2, \ldots \quad (1.8)$$

По числам m_k вычисляются псевдослучайные числа

$$\gamma_k = 2^{-40} m_k \quad (1.9)$$

Формула (1.8) означает, что число m_{k+1} равно остатку, полученному при делении $5^{17} m_k$ на 2^{40}, такой остаток называют наименьшим положительным вычетом по модулю 2^{40} Формулы (1.8), (1.9) легко реализовать на ЭВМ.

Достоинства метода псевдослучайных чисел довольно очевидны. Во-первых, на получение каждого числа затрачивается всего несколько простых операций, так что скорость генерирования случайных чисел имеет тот же порядок, что и скорость работы ЭВМ. Во-вторых, программа занимает не так много места в памяти. В-третьих, любое из чисел γ_k может быть легко воспроизведено. В-четвёртых, необходимо лишь один раз проверить «качество» такой последовательности, затем её можно много раз безбоязненно использовать при расчёте однотипных задач.

Единственный недостаток метода – ограниченность количества псевдослучайных чисел, так как если последовательность чисел $\gamma_0, \gamma_1, \ldots, \gamma_k, \ldots$ вычисляется на ЭВМ по формуле вида

$$\gamma_{k+1} = F(\gamma_k)$$

то эта последовательность обязательно периодическая. Впрочем, для наиболее распространённых псевдослучайных чисел период столь велик, что превосходит любые практические потребности. Подавляющее большинство расчётов по методу Монте-Карло осуществляется с использованием псевдослучайных чисел.

Значения любой случайной величины можно получить путём преобразования значений одной какой-либо случайной величины. Обычно роль такой случайной величины играет случайная величина γ, равномерно распределённая в $(0,1)$. Процесс нахождения значения какой-либо случайной величины ε путём преобразования одного или нескольких значений γ называется разыгрыванием случайной величины ε.

Допустим, что необходимо получать значения случайной величины ε, распределённой в интервале (a,b), с плотностью $p(x)$. Докажем, что значения ε можно находить из уравнения

$$\int_a^{\varepsilon} p(x)dx = \gamma \qquad (1.10)$$

т.е. выбрав очередное значение γ, надо решить уравнение (1.10) и найти очередное значение ε.

Для доказательства рассмотрим функцию

$$y = \int_a^x p(x)dx$$.

Из общих свойств плотности (1.2), (1.3) следует, что

$$y(a) = 0, y(b) = 1, y'(x) = p(x) > 0$$

Значит, функция $y(x)$ монотонно возрастает от 0 до 1, и любая прямая $y = \gamma$, где $0 < \gamma < 1$, пересекает график $y = y(x)$ в одной единственной точке, абсциссу которой мы и принимаем за ε. Таким образом, уравнение (1.10) всегда имеет одно и только одно решение.

Выберем теперь произвольный интервал (a', b'), содержащийся внутри (a, b). Точкам этого интервала $a' < x < b'$ отвечают ординаты кривой $y = y(x)$, удовлетворяющие неравенству $y(a') < y < y(b')$.

Поэтому, если ε принадлежит интервалу $a' < x < b'$, то γ принадлежит интервалу $y(a') < y < y(b')$, и наоборот. Значит $P\{a' < \varepsilon < b'\} = P\{y(a') < \gamma < y(b')\}$.

Так как γ равномерно распределена в $(0,1)$, то

$$P\{y(a') < \gamma < y(b')\} = y(b') - y(a') = \int_{a'}^{b'} p(x)dx$$,

итак

$$P\{a' < \varepsilon < b'\} = \int_{a'}^{b'} p(x)dx$$,

а это и означает, что случайная величина ε, являющаяся корнем уравнения (1.10), имеет плотность вероятностей $p(x)$.

Может оказаться, что разрешить уравнение (1.10) относительно ε трудно, например, в случаях, когда интеграл от $p(x)$ не выражается через элементарные функции или когда плотность $p(x)$ задана графически. Предположим, что случайная величина ε определена на конечном интервале (a, b) и плотность её ограничена $p(x) \le M_0$.

Разыгрывать значение ε можно следующим образом:

1) выбираются два значения γ' и γ'' случайной величины γ и строится

случайная точка ight=21 src="images/referats/7462/image088.png">с координатами

$$\eta' = a + \gamma'(b-a), \eta'' = \gamma'' M_0$$

2) если точка Γ лежит под кривой $y = p(x)$, то полагаем $\varepsilon = \eta'$, если же точка Γ лежит над кривой $y = p(x)$, то пара (γ', γ'') отбрасывается и выбирается новое значение.

2.3.2 Вычисление интегралов

Рассмотрим функцию $g(x)$, заданную на интервале $a < x < b$, требуется приближенно вычислить интеграл

$$I = \int_a^b g(x)dx \qquad (2.1)$$

Этот интеграл может быть несобственным, но абсолютно сходящимся.

Выберем произвольную плотность распределения $p(x)$, определённую на интервале (a,b). Наряду со случайной величиной ε, определённой в интервале (a,b) с плотностью $p(x)$, необходимо определить случайную величину

$$\eta = \frac{g(\varepsilon)}{p(\varepsilon)}$$

$$Mf(\varepsilon) = \int_a^b f(x)p(x)dx$$

Согласно соотношению получим

$$M\eta = \int_a^b \frac{g(x)}{p(x)} p(x)dx = I$$

Рассмотрим теперь N одинаковых независимых случайных величин $\eta_1, \eta_2, \ldots, \eta_N$ и применим к их сумме центральную предельную теорему. Формула (1.7) в этом случае запишется так:

$$P\left\{\left|\frac{1}{N}\sum_{j=1}^{N}\eta_j - I\right| < 3\sqrt{\frac{D_\eta}{N}}\right\} \approx 0.997$$

Последнее соотношение означает, что если выбирать N значений $\varepsilon_1, \varepsilon_2, \ldots, \varepsilon_N$, то при достаточно большом N

$$\frac{1}{N}\sum_{j=1}^{N}\frac{g(\varepsilon_j)}{p(\varepsilon_j)} \approx I \qquad (2.2)$$

Оно показывает также, что с очень большой вероятностью погрешность приближения (2.2) не превосходит $3\sqrt{\frac{D_\eta}{N}}$.

Для расчёта интеграла (2.1) можно использовать любую случайную величину ε. Определённую в интервале (a,b) с плотностью $p(x)>0$. В любом случае $M\eta = M\left[g(\varepsilon)/p(\varepsilon)\right] = I$. Однако дисперсия $D\eta$, а с ней и оценка погрешности формулы (2.2) зависят от того, какая величина ε используется, так как

$$D\eta = M\eta^2 - I^2 = \int_a^b \frac{g^2(x)}{p(x)}dx - I^2 \qquad (2.3)$$

Докажем, что это выражение будет минимальным тогда, когда $p(x)$ пропорциональна $|g(x)|$.

Для этого воспользуемся неравенством

$$\left(\int_a^b |u(x)v(x)|dx\right)^2 \le \int_a^b u^2(x)dx\int_a^b v^2(x)dx$$

, в которым положим $u = g(x)/\sqrt{p(x)}$, $v = \sqrt{p(x)}$. Получим неравенство

$$\left(\int_a^b |g(x)|dx\right)^2 \le \int_a^b \frac{g^2(x)}{p(x)}dx\int_a^b p(x)dx = \int_a^b \frac{g^2(x)}{p(x)}dx \qquad (2.4)$$

Из (2.3), (2.4) следует, что

$$D_\eta \geq \left(\int_a^b |g(x)|\,dx \right)^2 - I^2 \qquad (2.5)$$

Остается доказать, что нижняя граница дисперсии (2.5) реализуется при выборе плотности $p_0(x) = c|g(x)|$. Так как

$$c = \left(\int_a^b |g(x)|\,dx \right)^{-1}.$$

Следовательно,

$$\int_a^b \frac{g^2(x)}{p_0(x)}\,dx = \frac{1}{c}\int_a^b |g(x)|\,dx = \left(\int_a^b |g(x)|\,dx \right)^2,$$

и правая часть (2.3) обращается в правую часть (2.5)

Использовать плотность $p_0(x)$ для расчёта практически невозможно, так как для этого нужно знать значение интеграла $\int_a^b |g(x)|\,dx$. А его вычисление представляет собой задачу, равноценную задаче о вычислении интеграла (2.1). Поэтому ограничиваются следующей рекомендацией: желательно, чтобы плотность $p(x)$ была пропорциональна $|g(x)|$.

Конечно, выбирать очень сложные $p(x)$ нельзя, так как процедуры разыгрывания ε станет очень трудоёмкой. Оценку (2.2) с плотностью $p(x)$, сходной $|g(x)|$, называют существенной выборкой.

Также если стоит задача вычислить интеграл (2.1), преобразуем его к виду

$$\int_a^b \frac{g(x)}{p(x)} p(x)dx = I \quad (2.6)$$

Если теперь обозначить $\frac{g(x)}{p(x)} = h^*(x)$ (2.7)

То интеграл принимает вид

$$I = \int_a^b h^*(x)\, p(x)dx \quad (2.8)$$

и может быть вычислен при помощи метода статистических испытаний.

В частном случае, если a и b конечны или их можно считать конечными приближенно, в качестве $f(x)$ целесообразно выбрать равномерный закон распределения.

Как известно, плотность вероятности равномерного закона распределения в интервале (a, b) равна:

$$p(x) = \frac{1}{b-a} \quad (2.9)$$

Подставим в интеграл (2.6) значение $p(x)$ из формулы (2.9) и получим:

$$I = (b-a)\int_a^b g(x)\frac{1}{b-a}dx \quad (2.10)$$

и рассмотрим процедуру вычисления:

из множества равномерно распределённых случайных чисел

выбирается x_i. Для каждого значения x_i вычисляется , затем вычисляется среднее значение

$$\overline{g}(x_i) = \frac{1}{N}\sum_{i=1}^{N} g(x_i) \quad (2.11)$$

функции $g(x)$ на интервале (a,b)

Таким образом, величина интеграла (2.10) может быть представлена в виде следующей формулы

$$I \approx (b-a)\overline{g}(x_i) \quad (2.12)$$

Рассмотренный частный случай находит широкое применение интегралов методом статистического моделирования в силу того, что границы области определения могут быть легко приведены к пределам интегрирования (a,b)

2.3.3. Вычисление кратных интегралов

Обычно при вычислении кратных интегралов методом Монте-Карло используют один из двух способов.

Первый способ.

Пусть требуется вычислить $m-$кратный интеграл

$$I = \int\int\ldots\int f(x_1, x_2, \ldots, x_m)dx_1\ldots dx_m \quad (3.1)$$

по области G, лежащей в $m-$мерном единичном кубе

$$0 \le x_i \le 1 \ (i = 1, 2, \ldots, m)$$

Выберем m равномерно распределённых на отрезке $[0,1]$ последовательностей случайных чисел

$$x_1^{(1)}, x_2^{(1)}, x_3^{(1)}, \ldots, x_n^{(1)}, \ldots,$$
$$x_1^{(2)}, x_2^{(2)}, x_3^{(2)}, \ldots, x_n^{(2)}, \ldots,$$
$$\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots$$
$$x_1^{(m)}, x_2^{(m)}, x_3^{(m)}, \ldots, x_n^{(m)}, \ldots,$$

Тогда точки $M_i\left(x_i^{(1)}, x_i^{(2)}, \ldots, x_i^{(m)}\right), (i = 1, 2, \ldots)$ можно рассматривать как случайные, равномерно распределённые в $m-$мерном единичном кубе.

Пусть из общего числа N случайных точек n точек попали в область G, остальные $N - n$ оказались вне G. Тогда при достаточно большом N имеет место приближенная формула:

$$I \approx \frac{V_G}{n} \sum_{i=1}^{n} f\left(M_i\right) \quad (3.2)$$

где под V_G понимается $m-$мерный объём области интегрирования. Если вычисление объёма V_G затруднительно, то можно принять $V_G \approx n / N$, и для приближенного вычисления интеграла получим:

$$I \approx \frac{1}{N} \sum_{i=1}^{n} f(M_i) \quad (3.3)$$

Указанный способ можно применить к вычислению кратных интегралов и для произвольной области G, если существует такая замена переменных, при которой новая область интегрирования будет заключена в $m-$мерном единичном кубе.

Второй способ.

Если функция $y = f(x_1, x_2, ..., x_m) \geq 0$, то интеграл (3.1) можно рассматривать как объём тела в $(m+1)-$мерном пространстве, т.е.

$$I = \int\int...\int dx_1 dx_2 ... dx_m dy \quad (3.5)$$

где область интегрирования V определяется условиями $x = (x_1, x_2, ..., x_m) \in G, 0 \leq y \leq f(x)$

Если в области G $0 \leq f(x) \leq B, 0 \leq x_i \leq 1, (i = 1, 2, ..., m)$, то введя новую переменную $\eta = \frac{1}{B} y$, получим

$$I = B\int\int...\int dx_1 dx_2 ... dx_m d\eta$$

где область v лежит в единичном $(m+1)-$мерном кубе $0 \leq x_i \leq 1, (i = 1, 2, ..., m), 0 \leq \eta \leq 1$

Возьмём $m+1$ равномерно распределенных на отрезке $[0,1]$ случайных последовательностей

$x_1^{(1)}, x_2^{(1)}, x_3^{(1)}, \ldots, x_n^{(1)}, \ldots,$

$x_1^{(2)}, x_2^{(2)}, x_3^{(2)}, \ldots, x_n^{(2)}, \ldots,$

$x_1^{(m)}, x_2^{(m)}, x_3^{(m)}, \ldots, x_n^{(m)}, \ldots,$

$\eta_1, \eta_2, \eta_3, \ldots, \eta_n, \ldots.$

Составим соответствующую последовательность случайных точек $M_i\left(x_i^{(1)}, x_i^{(2)}, \ldots, x_i^{(m)}, \eta_i\right), (i = 1, 2, \ldots)$

Пусть из общего числа N случайных точек n точек принадлежат объёму v, тогда имеет место приближенная формула

$$I \approx B\frac{n}{N} \quad (3.6)$$

2.3.4. Практическая часть

Пример 1

Вычислим приближенно интеграл $I = \int_0^{\pi/2} \sin(x)dx$

Точное значение его известно: $\int_0^{\pi/2} \sin(x)dx = -\cos(x)\,|_0^{\pi/2} = 1$

Используем для вычисления две различные случайные величины ε, с постоянной плотностью $p(x) = 2/\pi$ (т.е. ε равномерна распределена в интервале $(0, \pi/2)$) и с линейной плотностью $p(x) = 8x/\pi^2$.Линейная плотность более соответствует рекомендации о

пропорциональности $p(x)$ и $|\sin(x)|$. Поэтому следует ожидать, что второй способ вычисления даст лучший результат.

1) Пусть $p(x) = 2/\pi$, формула для разыгрывания ε имеет вид $\varepsilon = (\pi/2)\gamma$. А формула (2.2) примет вид $I \approx \frac{\pi}{2N}\sum_{j=1}^{N} \sin \varepsilon_i$.

Пусть $N = 10$. В качестве значений γ используем тройки чисел из табл. 1 (см. приложение), умноженные на 0.001. Промежуточные результаты сведены в табл. 2.1. Результат расчёта $I \approx 0.952$

Таблица 2.1

i	1	2	3	4	5	"9%" valign=top > 6	7	8	9	10
γ_i	0.865	0.159	0.079	0.566	0.155	0.664	0.345	0.655	0.812	0.332
ε_i	1.359	0.250	0.124	0.889	0.243	1.043	0.542	1.029	1.275	0.521
$\sin \varepsilon_i$	0.978	0.247	0.124	0.776	0.241	0.864	0.516	0.857	0.957	0.498

2) пусть теперь $p(x) = 8x/\pi^2$. Для разыгрывания ε используем формулу

$$\int_0^{\varepsilon} \frac{8x}{\pi^2} dx = \gamma,$$

откуда получаем

$$\varepsilon = (\pi/2)\sqrt{\gamma}$$

формула (2.2) имеет вид

$$I \approx \frac{\pi^2}{8N} \sum_{j=1}^{N} \frac{\sin \varepsilon_i}{\varepsilon_i}$$

Пусть $N = 10$. Числа выберем те же, что и в случае 1. Промежуточные результаты сведены в табл. 2.2. Результат расчёта $I \approx 1.016$

Таблица 2.2

i	1	2	3	4	5	6	7	8	9	10
γ_i	0.865	0.159	0.079	0.566	0.155	0.664	0.345	0.655	0.812	0.332
ε_i	1.461	0.626	0.442	1.182	0.618	1.280	0.923	1.271	1.415	0.905
$\frac{\sin \varepsilon_i}{\varepsilon_i}$	0.680	0.936	0.968	0.783	0.937	0.748	0.863	0.751	0.698	0.868

Как и ожидалось, второй способ вычислений дал более точный результат.

3) По значениям, приведённым в таблицах (2.1) и (2.2) можно приближенно сосчитать дисперсии D_η для обоих методов расчёта:

для 1:

$$D_\eta \approx \frac{\pi^2}{9 \cdot 4} \left[\sum_{j=1}^{10} \left(\sin \varepsilon_j \right)^2 - \frac{1}{10} \left(\sum_{j=1}^{10} \sin \varepsilon_j \right)^2 \right] = \frac{\pi^2}{36} (4.604 - 3.670) = 0.256$$

для 2:

$$D_\eta \approx \frac{\pi^2}{9 \cdot 64} \left[\sum_{j=1}^{10} \left(\frac{\sin \varepsilon_j}{\varepsilon_j} \right)^2 - \frac{1}{10} \left(\sum_{j=1}^{10} \frac{\sin \varepsilon_j}{\varepsilon_j} \right)^2 \right] = \frac{\pi^2}{576} (6.875 - 6.777) = 0.016$$

Несмотря на то, что значение $N = 10$ невелико и приближенная нормальность оценки (2.2) не гарантирована, вычислим для обоих методов величины $0.6745\sqrt{\frac{D\eta}{N}}$. Получим значения 0.103 и 0.027. Также фактические абсолютные погрешности при расчёте I, равные 0.048 и 0.016, – величины того же порядка. Точные же значения D_η в рассмотренном примере равны 0.233 и 0.0166. Таким образом, и при оценке дисперсий метод 2 оказался точнее метода 1.

Пример 2

Рассмотрим пример:

Требуется вычислить интеграл

$$I = \iint_G (x^2 + y^2)dxdy \qquad (3.4)$$

где область G задаётся следующими неравенствами:
$1/2 \le x \le 1, 0 \le y \le 2x-1$

Область интегрирования принадлежит единичному квадрату $0 \le x \le 1, 0 \le y \le 1$. Для вычисления интеграла воспользуемся таблицей случайных чисел (см. приложение), при этом каждые два последовательных числа из этой таблицы примем за координаты случайной точки $M(x, y)$.

Записываем координаты x и y случайных точек в табл. 3.1, округляя до 3 знаков после запятой, и выбираем те из них, которые принадлежат области интегрирования.

Заполним табл. 3.1 по правилу:

1) Среди всех значений x выделяем те, которые заключены между $\underline{x}=0.5$ и $\overline{x}=1$.Для этих значений полагаем $\varepsilon_1=1$, для всех остал

ьных $\varepsilon_1=0$

2) Среди всех значений y . Соответствующих выделенным x, выбираем те, которые заключены между $\underline{y(x)}=0, \overline{y(x)}=2x-1$

Для этих значений полагаем $\varepsilon_2=1$, для всех остальных $\varepsilon_2=0$

Таблица 3.1

x	$\underline{x}$	$\overline{x}$	ε_1	y	$\underline{y(x)}$	$\overline{y(x)}$	ε_2	ε	$f(x,y)$
0.577	0.500	1.000	1	0.716	0	0.154	0	0	
0.737	0.500	1.000	1	0.701	0	0.474	0	0	
0.170	0.500	1.000	0	0.533				0	
0.432	0.500	1.000	0	0.263				0	
0.059	0.500	1.000	0	0.663				0	
0.355	0.500	1.000	0	0.094				0	
0.303	0.500	1.000	0	0.552				0	
0.640	0.500	1.000	1	0.205	0	0.280	1	1	0.452
0.002	0.500	1.000	0	0.557				0	
0.870	0.500	1.000	1	0.323	0	0.740	1	1	0.855
0.116	0.500	1.000	0	0.930				0	
0.930	0.500	1.000	1	0.428	0	0.860	1	1	1.048
0.529	0.500	1.000	1	0.095	0	0.058	0	0	
0.996	0.500	1.000	1	0.700	0	0.992	1	1	1.482
0.313	0.500	1.000	0	0.270				0	
0.653	0.500	1.000	1	0.934	0	0.306	0	0	
0.058	0.500	1.000	0	0.003				0	
0.882	0.500	1.000	1	0.986	0	0.764	0	0	
0.521	0.500	1.000	1	0.918	0	0.042	0	0	
0.071	0.500	1.000	0	0.139				0	
всего								4	3.837

3) Вычисляем $\varepsilon = \varepsilon_1 \varepsilon_2$. Области тнтегрирования принадлежат только те точки, для которых $\varepsilon = 1$. В примере $N = 10, n = 4$

4) Вычисляем значения подынтегральной функции в полученных точках.

После заполнения табл. 3.1 вычисляем

площадь области интегрирования $V_G = \frac{1}{2} \cdot 1 \cdot \frac{1}{2} = \frac{1}{4}$ и по формуле (3.2) находим

$$I \approx \frac{1}{4} \cdot \frac{1}{4} \cdot (0.452 + 0.855 + 1.048 + 1.482) = \frac{1}{16} \cdot 3.837 = 0.240$$

Для сравнения приведём точное значение интеграла
$I = 7/32 = 0.21875$

Результат имеет сравнительно небольшую точность потому, что число точек $N = 20$ недостаточно велико.

Пример 3

Рассмотрим пример: найдём приближенно объём, ограниченный поверхностями

$$z = 2 + \sqrt{(0.5)^2 - (x - 0.5)^2 - (y - 0.5)^2},$$
$$(x - 0.5)^2 + (y - 0.5)^2 = (0.5)^2, z = 0$$

Искомый объём численно равен величине интеграла

$$I = \iiint_V dxdydz \qquad (3.7)$$

Так как в области V $0 \le z \le 2.5$, вводим новую переменную $\eta = \frac{z}{2.5}$, в результате чего интеграл (3.7) переходит в интеграл

$$I = 2.5 \iiint_v dxdyd\eta \qquad (3.8)$$

где v – область, ограниченная поверхностями

$$(x-0.5)^2 + (y-0.5)^2 = (0.5)^2,$$
$$\eta = 0.8 + 0.4\sqrt{(0.5)^2 - (x-0.5)^2 - (y-0.5)^2}, \eta = 0,$$

т.е. v принадлежит единичному кубу $0 \le x \le 1, 0 \le y \le 1, 0 \le \eta \le 1$.

Берём теперь три равномерно распределенные на отрезке $[0,1]$ последовательности случайных чисел и записываем их в качестве координат x, y, η случайных точек в табл. 3.2. Затем проверяем, какие из этих точек принадлежат области v.

Таблица 3.2

i	x	y	$\lvert x-0.5\rvert\lvert y-0.5\rvert$	$(x-0.5)^2+\varepsilon_1$	η	$\lvert\eta-0.8\rvert$	$6.25\cdot(\eta-0.8)^2$	ε_2	ε
1	0.577	0.116	0.077	0.384	0.147	1	0.667	1	1
2	0.716	0.930	0.216	0.430	0.232	0.993	0.193	0.231	0
3	0.737	0.930	0.237	0.430	0.241	1	0.242	1	1
4	0.701	0.428	0.201	0.072	0.045	0.940	0.140	0.122	1
5	0.170	0.529	0.330	0.029	0.110	1	0.610	1	1

6	0.533	0.095	0.033	0.405	0.165	1	0.131	1	1
7	0.432	0.996	0.068	0.496	0.251	0	0.352	1	0
8	0.263	0.699	0.237	0.199	0.096	1	0.645	1	1
9	0.059	0.313	0.441	0.187	0.229	1	0.646	1	1
10	0.663	0.270	0.163	0.230	0.080	1	0.680	1	1
11	0.355	0.653	0.145	0.153	0.046	1	0.577	1	1
12	0.094	0.934	0.406	0.434	0.353	0	0.716	1	0
13	0.303	0.058	0.197	0.442	0.234	1	0.737	1	1
14	0.552	0.003	0.052	0.497	0.250	1	0.701	1	1
15	0.640	0.882	0.140	0.382	0.165	1	0.169	1	1
16	0.205	0.986	0.295	0.486	0.323	0	0.533	1	0
17	0.002	0.521	0.498	0.021	0.248	1	0.432	1	1
18	0.557	0.918	0.057	0.418	0.178	1	0.263	1	1
19	0.870	0.071	0.370	0.429	0.318	0	0.059	1	0
20	0.313	0.139	0.187	0.361	0.185	1	0.663	1	1

n=15

Заполним табл. 3.2 по правилу:

1) выделяем точки, у которых $\eta \le 0.8$, и полагаем для них $\varepsilon_2 = 1$

2) среди выделенных точек области v принадлежат те, для которых

выполняется неравенство
c="images/referats/7462/image228.png">.

Для этих точек $\varepsilon_1 = 1$, для остальных $\varepsilon_1 = 0$

3) вычисляем $\varepsilon = \varepsilon_1 \varepsilon_2$. Области v принадлежат те точки, для которых $\varepsilon = 1$

4) среди точек, у которых $0.8 < \eta < 1$, области v принадлежат те точки, координаты которых удовлетворяют неравенству

$$(x-0.5)^2 + (y-0.5)^2 + 6.25(\eta - 0.8)^2 \le (0.5)^2$$

Для этих точек $\varepsilon = 1$.

В примере общее количество точек $N = 20$, а число точек, принадлежащих области v, равно 15. По формуле (3.6) получаем

$I \approx 2.5\frac{n}{N} = 2.5\frac{15}{20} = 1.875$, а точное значение объёма V равно 1.833

Погрешность формулы (3.6) обратно пропорциональна корню из числа испытаний, т.е. $R = O\left(\frac{1}{\sqrt{N}}\right)$.

Это означает, что для обеспечения большой точности число точек N должно быть очень велико. Но так как приближенные формулы (3.3), (3.6) не зависят от размерности интеграла, метод Монте-Карло оказывается выгодным при вычислении интегралов большой размерности.

Приложения

1. Таблица 400 случайных цифр

86615	90795	66155	66434	56558	12332	94377	57802
69186	03393	42505	99224	88955	53758	91641	18867
41686	42163	85181	38967	33181	72664	53807	00607
86522	47171	88059	89342	67248	09082	12311	90316
72587	93000	89688	78416	27589	99528	14480	50961
52452	42499	33346	83935	79130	90410	45420	77757
76773	97526	27256	66447	25731	37525	16287	66181
04825	82134	80317	75120	45904	75601	70492	10274
87113	84778	45863	24520	19976	04925	07824	76044
84754	57616	38132	64294	15218	49286	89571	42903

2. Таблица 40 случайных чисел, равномерно распределенных на отрезке $[0,1]$

<

/tr>

0.57705	0.35483	0.11578	0.65339
0.71618	0.09393	0.93045	0.93382
0.73710	0.30304	0.93011	0.05758
0.70131	0.55186	0.42844	0.00336
0.16961	0.64003	0.52906	0.88222
0.53324	0.20514	0.09461	0.98585
0.43166	0.00188	0.99602	0.52103
0.26275	0.55709	0.69962	0.91827
0.05926	0.86977	0.31311	0.07069
0.66289	0.31303	0.27004	0.13928

3. Листинг программы

Вычисляются значения кратных интегралов из примера 2–3.

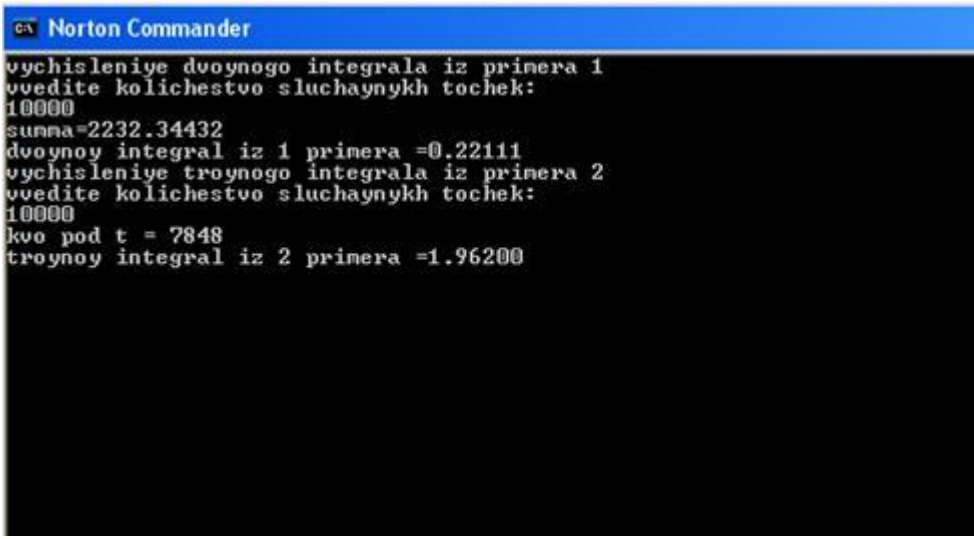
```
program pmk;
```

```
uses crt;

var

w, u, h, k, v, y, p, s, g, x, x2, y2, z2, niu, Integral, Integral2:real;

n, m, i, a, b, e1, e2, e, e3, e4, e5:integer;

begin

clrscr;

writeln ('vychisleniye dvoynogo integrala iz primera 1');

writeln ('vvedite kolichestvo sluchaynykh tochek:');

readln(n);

for i:=1 to n do

begin

g:=random;

p:=random;

x:=g;

y:=p;

if ((0.5<=x) and (x<=1)) then e1:=1

else e1:=0;

if ((0<=y) and (y<=2*x-1)) then e2:=1

else e2:=0;
```

```
e:=e1*e2;

if e=1 then s:=s+x*x+y*y;

if e=1 then a:=a+1;

v:=1/4;

delay(1000);

end;

Integral:=(v/a)*(s);

writeln ('summa=', s:5:5);

writeln ('dvoynoy integral iz 1 primera =', Integral:5:5);

writeln ('vychisleniye troynogo integrala iz primera 2');

writeln ('vvedite kolichestvo sluchaynykh tochek:');

readln(m);

for i:=1 to m do

begin

w:=random;

u:=random;

h:=random;

x2:=w;

y2:=u;
```

```
niu:=h;

if niu<=0.8 then e3:=1;

if (x2–0.5)*(x2–0.5)+(y2–0.5)*(y2–0.5)<=(0.5)*(0.5) then e4:=1

else e4:=0;

e5:=e3*e4;

if (((0.8<niu) and (niu<1)) and ((x2–0.5)*(x2–0.5)+(y2–0.5)*(y2–
0.5)+6.25*(niu-0.8)*(niu-0.8)<=(0.5)*(0.5))) then e5:=1;

if e5=1 then b:=b+1;

delay(1000);

end;

Integral2:=2.5*(b/m);

writeln ('kvo pod t =', b:5);

writeln ('troynoy integral iz 2 primera =', Integral2:5:5);

readln;

end.
```

4. Пример работы программы при 10000 случайных точек



Рассмотрим 2 весьма изящных метода Мрнте-Карло факторизации, предложенных Поллардом. Они очень просты и позволяют быстро извлечь из составного числа все его небольшие простые делители.

2.3.5. Метод Монте-Карло 1

Время работы этого метода порядка корень квадратный из минимального простого числа, делящего m. То есть в худшем случае, когда m есть произведение двух простых чисел примерно одного порядка, число m раскладывается этим методом на множители за время корень четвертой степени из m. Алгоритм очень прост, его запись намного короче, чем объяснение, почему он работает. Слова "Монте-Карло" присутствуют в его названии потому, что работа алгоритма зависит от случайного выбора начального числа.

Рассмотрим отображение f кольца Z_m в Z_m:

f: Z_m --> Z_m
f(x) = x^2 + 1 (mod m)

Выберем случайным образом элемент b_0 кольца Z_m.
Рассмотрим бесконечную последовательность элементов кольца Z_m:

$$b\_0, b\_1 = f(b\_0), b\_2 = f(b\_1), b\_3 = f(b\_2), \ldots \quad (2)$$

Последовательность представляет собой орбиту элемента b_0 при отображении f. Поскольку все элементы последовательности принадлежат конечному множеству, последовательность циклическая -- точнее, она содержит начальный апериодический отрезок и далее бесконечно повторяющийся период.

Пусть p -- делитель числа m. Рассмотрим элементы последовательности (2) по модулю p (т.е. образы элементов b_i при каноническом эпиморфизме $Z_m \dashrightarrow Z_p$):

$$c\_0 == b\_0 \pmod p, c\_1 == b\_1 \pmod p, c\_2 == b\_2 \pmod p, \ldots \quad (3)$$

Так как в Z_p меньше элементов, чем в Z_m, то с большой вероятностью период последовательности (3) меньше, чем период последовательности (2).
Следовательно, найдется пара индексов i, j такая, что

$$c\_i = c\_j, b\_i \mathrel{!}= b\_j.$$

Это означает, что

$$b\_i == b\_j \pmod p, b\_i \mathrel{!}= b\_j \pmod m.$$

Отсюда вытекает, что (b_i - b_j) делится на p, но не делится на m. Следовательно, НОД(b_i - b_j, m) нетривиален, и нам удалось разложить m на множители.

Итак, алгоритм Полларда 1 сводится к поиску цикла в бесконечной рекурсивной последовательности, состоящей из элементов конечного множества. При этом вместо того, чтобы сравнивать между собой два элемента, мы вычисляем наибольший общий делитель их разности и числа m. Алгоритм завершается, когда наибольший общий делитель нетривиален.

Можно предложить 2 способа решения задачи поиска цикла в последовательности. Первый способ наиболее простой. Второй чуть-чуть сложнее, но зато более быстрый.

```
Способ 1
--------

Выполняется следующая последовательность сравнений:
    b0 <---> b1
    b1 <---> b3
    b2 <---> b5
    b3 <---> b7
    b4 <---> b9
       . . .
   b_i <---> b_{2*i+1}
       . . .
```

Рано или поздно мы дойдем до равенства двух элементов, поскольку расстояние между сравниваемыми элементами на каждом шаге увеличивается ровно на единицу; кроме того, левый элемент сдвигается вправо, так что он рано или поздно войдет в периодический участок последовательности.

Выпишем алгоритм нахождения делителя.

```
алгоритм факторизация1(вход: целое число m, выход:
целое число d): успех
| дано: целое число m
| надо: получить нетривиальный делитель d числа m
| возвращаемое значение: true, если удалось
разложить,
|                        false в противном случае
начало
| maxSteps := 1000000                              //
Максимальное число шагов
| step := 0
|
| b0 := случайное число в интервале 0..m
| b1 := mod(b0 * b0 + 1, m)
```

```
| d := gcd(b1 - b0, m)
|
| цикл пока step < maxSteps && d == 1 выполнять //
Пока НОД тривиален
| | b0 = mod(b0 * b0 + 1, m)                    //
Применяем отображение f
| | b1 = mod(b1 * b1 + 1, m);                   //
один раз к b0 и дважды
| | b1 = mod(b1 * b1 + 1, m)                    //
к b1
| | d := gcd(b1 - b0, m)
| | step := step + 1
| конец_цикла
|
| вернуть (d != 1)                            //
Успех := d нетривиален
конец_алгоритма
```

На каждом шаге цикла мы трижды вычисляем значение отображения f. Небольшая модификация алгоритма позволяет делать это только один раз.

```
Способ 2
--------

Выполняется следующая бесконечная
последовательность сравнений

    b0 <---> b1

    b1 <---> b2
    b1 <---> b3

    b2 <---> b4
    b2 <---> b5
    b2 <---> b6
    b2 <---> b7

    b4 <---> b8
    b4 <---> b9
    b4 <---> b10
       . . .
    b4 <---> b15
```

```
    b8 <---> b16
    b8 <---> b17
       . . .
    b8 <---> b31

       . . .
```

Вся последовательность сравнений разбивается на серии. В очередной серии мы сравниваем элемент b_s, где s -- степень двойки, последовательно с элементами b_{2s}, b_{2s+1}, b_{2s+2}, ..., b_{4s-1}. Серия содержит 2s сравнений.

Выпишем алгоритм.

```
алгоритм факторизация2(вход: целое число m, выход:
целое число d): успех
| дано: целое число m
| надо: получить нетривиальный делитель d числа m
| возвращаемое значение: true, если удалось
разложить,
|                        false в противном случае
начало
| maxSteps := 19                 // Максимальная
длина серии 2^19
| step := 0
|
| b0 := случайное число в интервале 0..m
| b1 := mod(b0 * b0 + 1, m)
| a := b1                        // Первый элемент
серии
| seriesLength := 1              // Длина серии
|
| цикл пока step < maxSteps && d == 1 выполнять
// пока НОД тривиален
| | Инвариант: b0 -- элемент последовательности с
индексом,
| |                    равным нулю или степени
двойки
| |            a  -- элемент, индекс которого равен
удвоенному индексу
| |                   элемента b0 (или 1, если
индекс b0 равен 0)
```

```
| |            seriesLength == удвоенному индексу
элемента a
| | d := gcd(b1 - b0, m)
| | len := 0
| |
| | цикл пока d == 1 и len < seriesLength выполнять
| | | b1 = mod(b1 * b1 + 1, m);
| | | d := gcd(b1 - b0, m)
| | | len := len + 1
| | конец_цикла
| |
| | b0 := a
| | a := b1
| | seriesLength := seriesLength * 2
| конец_цикла
|
| вернуть (d != 1)                              //
Успех := d нетривиален
конец_алгоритма
```

2.3.6. Метод Монте-Карло 2

Пусть m --- целое число, которое мы раскладываем на множители. Оно представимо в виде произведения степеней простых чисел

m = p1^e1 p2^e2 ... pk^ek

Предположим, что p1 - 1 представимо в виде произведения степеней простых чисел, причем каждая из этих степеней не очень велика. Более точно, существует N такое, что

p1 - 1 = q1^a1 q1^a2 ... qr^ar q1^a1 < N, q2^a2 < N, ..., qr^ar < N.

Рассмотрим всевозможные максимальные степени простых чисел, не превосходящие N. Например, пусть N = 20, тогда рассматриваются степени простых 16, 9, 5, 7, 11, 13, 17, 19. Обозначим эти степени простых через t1, t2, ..., ts.

Выберем произвольное целое число b. Рассмотрим последовательность

```
b0 = b,
b1 = b0^t1 (mod m),
b2 = b1^t2 (mod m),
...
bs = b_{s-1}^ts (mod m)
```

Каждый раз, вычислив bi, вычисляем одновременно

НОД(bi - 1, m).

Утверждается, что с большой вероятностью на каком-то шаге этот НОД будет нетривиальным делителем N. Действительно, покажем, что

p | НОД(bs - 1, m).

Действительно,

bs = b^(t1 t2 ... ts),

и, поскольку по предположению, p1 - 1 | t1 t2 ... t3, то есть t1 t2 ... ts = (p1 - 1)g, то

bs = b^(t1 t2 ... ts) = b^((p1 - 1) g) = (b^(p1 - 1))^g = 1 (mod p1)

по малой теореме Ферма. Значит, bs - 1 делится на p1, число m также делится на p1, следовательно, НОД(bs - 1, m) делится на p1.

Проиллюстрируем алгоритм на простом примере. Возьмем N = 20. Выпишем все степени простых, не превосходящие 20:

t1 = 16, t2 = 9, t3 = 5, t4 = 7,
t5 = 11, t6 = 13, t7 = 17, t8 = 19.

Попытаемся разложить на множители число m = 41779 = 41 * 1019. Выберем b = 2. Последовательно вычисляем

```
    2     ^ 16 (mod 41779) = 23757, gcd(23757 - 1,
41779) = 1,
    23757 ^ 9  (mod 41779) = 7970,  gcd(7970  - 1,
41779) = 1,
```

```
    7970  ^ 5  (mod 41779) = 33580, gcd(33580 - 1,
41779) = 41.
```

Мы получили нетривиальный делитель на третьем шаге, поскольку 41 - 1 = 8 * 5 делит (t1 * t2 * t3) = 16 * 9 * 5.

Мощность алгоритма зависит от числа N --- чем больше оно, тем большие числа можно разложить с помощью этого алгоритма. Работа алгоритма разбивается на 2 шага. Сначала мы генерируем все максимальные степени простых чисел, не превосходящие N. Этот шаг выполняется только один раз и не зависит от входного числа m, поэтому сгенерированные степени можно, к примеру, записать в файл и в дальнейшем использовать многократно. Затем мы выбираем случайным образом число b и вычисляем указанную выше последовательность степеней b. Для каждой степени bi вычисляется НОД(bi - 1, m).

Алгоритм завершается успешно, если вычисленный НОД нетривиален. Алгоритм можно убыстрить, если вычислять НОД не на каждом шаге, а, скажем, на каждом сотом шаге. При этом на промежуточных шагах последовательно вычисляется произведение

(b_i - 1) (b_{i+1} - 1) (b_{i+2} - 1) ... (b_{i+99} - 1) = n (mod m)

и затем вычисляется НОД(n, m).

2.4. Алгоритмы Лас Вегаса

Алгоритмы Лас Вегаса никогда не возвращают неправильный ответ, хотя иногда они не возвращают вообще никакого ответа. Чем дольше работают эти алгоритмы, тем выше вероятность того, что они вернут правильный ответ.

Алгоритм Лас Вегаса принимает случайное решение, а затем проверяет, приводит ли это решение к успеху. Программа, использующая алгоритм Лас Вегаса, вызывает его раз за разом, пока он не достигнет результата. Если обозначить через $st(x)$ и $ft(x)$ время, необходимое для того, чтобы получить соответственно

положительный или отрицательный ответ на входных данных длины x, а через $p(x)$ - вероятность успешного завершения работы алгоритма, то мы приходим к равенству

$$time(x) = p(x) * st(x) + (1 - p(x)) * (ft(x) + time(x))$$

Это равенство означает, что в случае успеха затраченное время совпадает с временем получения успешного результата, а в случае неудачи затраченное время равно сумме времени на достижение неудачного результата и еще на один вызов функции. Решая это уравнение относительно $time(x)$, мы получаем

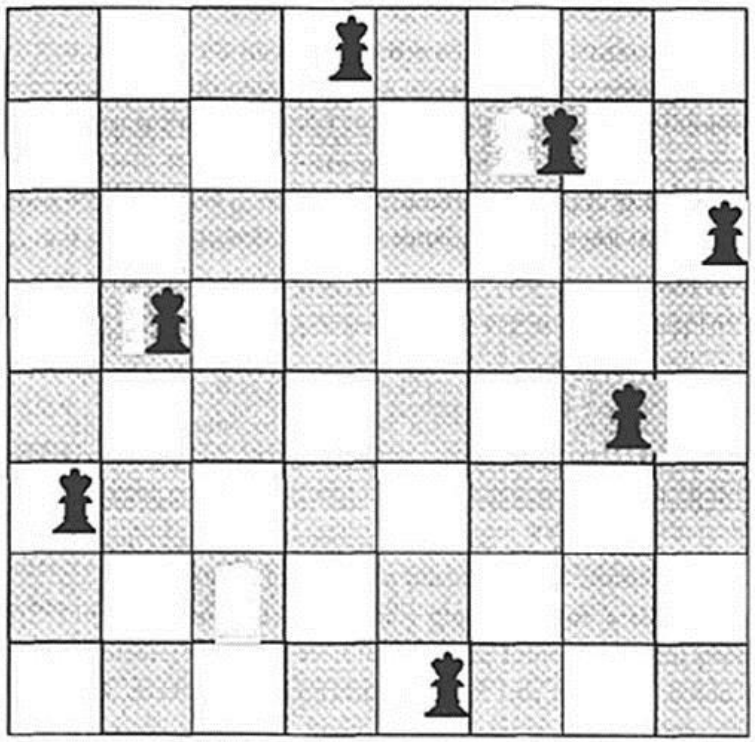
$$time(x) = st(x) + \frac{1 - p(x)}{p(x)} * ft(x)$$

.

Эта формула означает, что время выполнения зависит от времени получения успешного результата, безуспешного результата и вероятности каждого из этих исходов. Интересно, что при убывании вероятности p(x) успешного результата время выполнения все равно может быть невысоким, если скорость получения безуспешного результата возрастает. Поэтому эффективность можно повысить, если быстрее получать безуспешный результат. Как такой подход работает на практике?

Задача о расстановке ферзей.

Обратимся к задаче о расстановке восьми ферзей на шахматной доске так, чтобы они не били друг друга.



На рисисунке изображено одно из решений этой задачи. Рекурсивный алгоритм ее решения помещает ферзя в первой клетке первой вертикали, а затем вызывает себя для того, чтобы поставить ферзя на вторую горизонталь. Если в какой-то момент алгоритму не удается найти положения для очередного ферзя на очередной горизонтали, то алгоритм возвращается на предыдущий шаг и пробует другое размещение ферзя на предыдущей строке.

Имеется вероятностная альтернатива детерминированному рекурсивному алгоритму. Мы можем поочередно размещать ферзей на доске случайным образом на очередной свободной горизонтали доски. Отличие алгоритма Лас Вегаса от стандартного рекурсивного алгоритма состоит в том, что при невозможности разместить

очередного ферзя алгоритм попросту сдается и сообщает о неудаче. Рекурсивный же алгоритм пытается добиться положительного результата.

Полный статистический анализ алгоритма показывает, что вероятность успеха равна 0.1293, а среднее число повторений, необходимых для его достижения, около 6.971. Приведенное выше уравнение показывает, что алгоритм выполнит при этом 55 проходов. Рекурсивному же алгоритму понадобится по крайней мере вдвое больше проходов.

2.5. Тройная дуэль

В настоящем примере проанализируем ожидаемую трудоемкость необычайно простой и эффективной стратегии в довольно своеобразной игре. Сэм, Билл и Джон (ниже обозначенные буквами С, *Б* и *Д)* договорились сразиться на дуэли втроем по следующим правилам: (1) они тянут жребий, кто стреляет первым, вторым и третьим; (2) далее они располагаются в вершинах равностороннего треугольника; (3) обмениваются выстрелами в порядке вытянутых номеров, пока двое не будут убиты, и (4) очередной стреляющий может стрелять в любого из остальных.

Известно, что С — снайпер и никогда не промахивается на данной дистанции, *Б* поражает мишень в 80% случаев, а Д — приблизительно в 50% случаев. Какова наилучшая стратегия для каждого из участников и каковы их вероятности выживания, если они следуют оптимальным стратегиям?

Если *С* стреляет первым, у него нет лучшего выбора, чем убить *Б.* В самом деле, если он убьет *Д,* то *Б* (с вероятностью 0,8) получит право следующего выстрела и должен целиться в С (так как *Д* уже убит). Ясно, что *С* предпочтет, чтобы стрелял *Д.*

Аналогично, если *Б* стреляет первым, он попытается убить *С.* Если он этого не сделает, то *С* наверняка убьет его при первой возможности, ведь *С* — снайпер.

Перед *Д* стоит дилемма. Если он стреляет первым, то ему в действительности не выгодно никого убивать. В самом деле, если он сделает это, то оставшийся в живых (а он хороший стрелок) будет стрелять в него в следующую очередь независимо от вытянутого номера. Таким образом, *Д* решает промахиваться нарочно, пока один из остальных дуэлянтов не будет убит. Тогда наступит очередь *Д,* и он постарается

сделать все, на что он способен. Эта стратегия преобладает на дуэли в том смысле, что, если *Д* принимает ее, она максимизирует вероятность его выживания. Если кто-нибудь из противников, С или *Б,* предпочтет отклониться от предварительно выбранной стратегии (не отменяя самой дуэли), вероятность выживания *Д* соответственно уменьшится. Мы моделируем ход дуэли деревом, которое первоначально имеет только две ветви, так как *Д* на самом деле не вступает в дело, пока не убит *С* или *Б.* В этом случае дерево остается двоичным и довольно маленьким.

Каждая вершина дерева означает событие (например, *Д* убивает *Б),* а ребра имеют веса, равные вероятностям событий на их концах (в вершинах, наиболее далеких от корня СТАРТ), при условии, что произошло событие, расположенное в начале ребра (вершина, ближайшая к корню). Дерево дуэли приведено на рисунке.

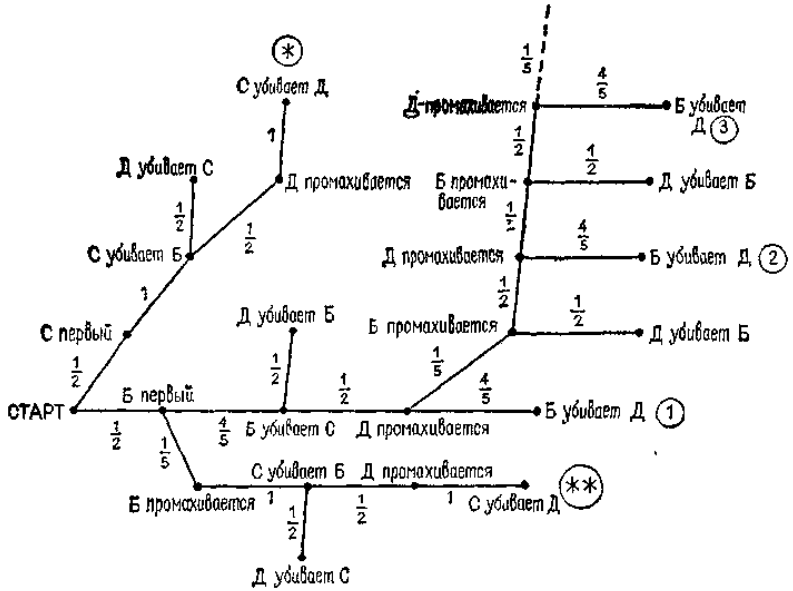


Рисунок. Дерево вероятностей для тройной дуэли.

Пунктирная ветвь в верхней правой части рисунка означает бесконечное повторение основной последовательности, в которой *Б* и *Д* продолжают стрелять друг в друга (и, возможно, промахиваются до бесконечности).

Теперь мы вычислим различные вероятности выживания. Первым рассмотрим *С.* Существуют два пути в дереве, означающие победу С,

они отмечены символами * и **. Так как эти два пути представляют взаимно исключающие события, то их вероятности аддитивны. Рассмотрим событие *. По определению условной вероятности получим:

$$P(*) = P(C \text{ первый}) \times P(C \text{ убьет } Б \mid C \text{ первый}) \times P(Д \text{ промахнется по } C \mid C \text{ первый}, C \text{ убьет } Б) \times P(C \text{ убьет } Д \mid C \text{ первый}, C \text{ убьет } Б, Д \text{ промахнется по } C) =$$
$$= (1/2)(1)(1/2)(1) = 1/4$$

Аналогично

$$P(**) = (1/2)(1/5)(1)(1/2)(1) = 1/20$$

Обратите внимание на то, что структура дерева гарантирует взаимно-однозначное соответствие между конечным событием и единственным путем от корня в конечной вершине. Дерево было построено так, чтобы вычисление различных сложных условных вероятностей сводилось не более, чем к перемножению всех чисел вдоль ребер данного пути. Мы успешно вычисляем составные вероятности по мере «наращивания» дерева. Тогда $P_C = P(C$ выживет$) = 1/4+1/20=3/10$. Теперь перейдем к *Б:*

$$P_Б = P(Б \text{ выживет}) = P(Б \text{ убьет } C) \times P(Б \text{ убьет } Д \mid Б \text{ убьет } C) =$$
$$= P(Б \text{ первый}) \times (Б \text{ убьет } C \mid Б \text{ первый}) \times$$
$$\times P(Б \text{ убьет } Д \mid Б \text{ убьет } C) =$$
$$= (1/2)(4/5)\sum_{n=1}^{\infty} \frac{4}{10^n}(2/5)(4/9) = 8/45$$

с учетом того, что все исходы, в которых *Б* побеждает расположены на правой верхней ветви дерева рисунка (все эти события обозначены номерами в кружочках).

Вероятность выживания *Д* $P_Д$ можно получить из условия нормировки $P_C + P_Б + P_Д = 1$. Получим $P_Д = 47/90$. Таким образом, немного подумав, *Д* может гарантировать себе наибольшие шансы на выживание в ситуации, которая на первый взгляд казалась для него крайне невыгодной.

2.6. Алгоритм извлечения выборки

В ряде задач обработки данных и в игровых задачах требуется несмещенная выборка в точности *M* предметов из множества *N* пред-

метов. Например, может возникнуть желание смоделировать раздачу карт при игре в покер или взять ограниченную выборку из набора записей.

Один из наивных подходов состоит в том, чтобы выбирать записи с вероятностью *M/N*. Этот метод удобен тем, что каждый предмет принимается или отвергается в момент его прохождения, и можно делать только один проход вдоль набора предметов. К сожалению, такая процедура не гарантирует, что будет выбрано точно *M* предметов.

Чтобы убедиться в этом, определим случайные переменные для $i=1, \ldots, N$:

$$X_i = \begin{cases} 1, & \text{если выбран предмет } i; \\ 0, & \text{если предмет } i \text{ не выбран.} \end{cases}$$

Вероятностная мера для каждого X_i равна

$$P(X_i = 0) = 1 - \frac{M}{N};$$

$$P(X_i = 1) = \frac{M}{N}.$$

X_i — независимые, одинаково распределенные случайные переменные. Каждая имеет среднее *M/N* и дисперсию $(M/N)\cdot[1-(M/N)]$. Случайная переменная

$$X = X_1 + X_2 + \ldots + X_N$$

означает число предметов, извлеченных в некотором испытании. Тогда

$$E[X] = E[X_1] + \ldots + E[X_N] = N\frac{M}{N} = M;$$

$$\operatorname{var}[X] = \operatorname{var}[X_1] + \ldots + \operatorname{var}[X_N] = N\frac{M}{N}\left(1 - \frac{M}{N}\right) = M\left(1 - \frac{M}{N}\right).$$

Таким образом, только *среднее* число выбранных предметов равно *M,* а большая дисперсия приводит к тому, что действительное число выбранных предметов в некотором испытании может оказаться существенно больше или меньше, чем *M.*

Другой интуитивный подход при выборе *M* чисел из *N* состоит в том, чтобы многократно генерировать случайные целые числа до тех пор, пока не окажется *M* различных. Эту процедуру можно сформулировать в виде алгоритма следующим образом:

Algorithm TRY AGAIN *(Еще одна попытка)* для извлечения несмещенной выборки $1 \leqslant M \leqslant N$ чисел из множества 1, 2, . . ., *N*. Выбранные числа будут помещены в массив SELECT (1), SELECT (2), . . ., SELECT (M).

Шаг 0. [Инициализация) **Set** $L \leftarrow 0$ (переменная *L* используется для записи текущего числа выбранных объектов); **and** пометить все целые числа 1, . . ., . *N* как «невыбранные».

Шаг 1. [цикл] **Do through** шаг **3 while** $L<M$ **od; and** STOP.

Шаг 2. [Генерация случайного целого числа] Генерируется (с равномерным распределением) случайное число K между 1 и N.
Шаг 3. [Было ли выбрано K?*)* **If** K помечено «не выбрано» **then set** $L \leftarrow L+1$; SELECT (L) $\leftarrow K$; **and** пометить K «выбрано» **fi**.

Хотя алгоритм TRYAGAIN интуитивно привлекателен и прост, он может оказаться очень медленным и неэффективным, потому что может понадобиться сгенерировать большое количество случайных чисел прежде, чем появится следующее невыбранное число. Проанализируем ситуацию.
Пусть $p_L = (N-L)/N$ — вероятность генерации невыбранного числа из 1, . . ., N при условии, что уже выбрано L целых чисел. Пусть $q_L = 1-p_L = L/N$ — вероятность генерации одного из L ранее выбранных чисел.
Если X_L — случайная переменная, равная количеству целых чисел, сгенерированных прежде, чем появилось невыбранное число, при условии, что уже было выбрано L чисел, то

$$P(X_L = k) = q_L^{k-1} p_L.$$

Таким образом, X_L имеет геометрическое распределение и

$$E(X_L) = \sum_{k=1}^{\infty} k q_L^{k-1} p_L = \frac{1}{p_L}.$$

Например, если нужно выбрать 97 целых чисел из 100 с помощью алгоритма TRYAGAIN и уже выбрано 96, потребуется сгенерировать в среднем еще 100/(100—96)=25 чисел прежде, чем будет получено одно из оставшихся трех чисел.
Если T — случайная переменная, равная общему количеству целых чисел, сгенерированных при выборе M чисел из N, то

$$E(T) = \sum_{L=0}^{M-1} E(X_L);$$

$$E(T) = 1 + \frac{N}{N-1} + \frac{N}{N-2} + \ldots + \frac{N}{N-(M-1)} = N \sum_{k=M-1}^{N} \frac{1}{k} =$$

$$= N(H_N - H_{M-1});$$

$$H_N = \sum_{k=1}^{N} \frac{1}{k} \text{ (это } N\text{-е гармоническое число).}$$

Таким образом, $E(T) = O(N \log_2 N)$.
Нам желательно иметь алгоритм, гарантирующий извлечение точно M предметов при каждом испытании, но обладающий тем же свойством,

что и первая описанная процедура: решение «принять / отвергнуть» достигается за один проход.

Предположим, что мы собираемся рассматривать (k +1)-й предмет и что уже выбрано q предметов. Какое значение вероятности следует приписать принятию этого предмета? Убедимся, что существует $\binom{N-k}{M-q}$ способов выбрать $M—q$ предметов из остающихся $N—k$. Все способы следует рассматривать как равновероятные, и один должен иметь место, если мы в конце концов выберем M предметов.

Аналогично, если выбран (k +1)-й предмет, то существует $\binom{N-k-1}{M-q-1}$ способов выбрать $M—q—1$ предметов из $N—k—1$ оставшихся. При всех способах выбора M предметов из N, когда q выбраны из k первых, в точности

$$\frac{\binom{N-k-1}{M-q-1}}{\binom{N-k}{M-q}} = \frac{M-q}{N-k}$$

из них выбирают (k +1)-й элемент. Мы утверждаем, что это та вероятность, которую следует приписать выбору (k +1)-го предмета.

Прежде чем принять это утверждение, сформулируем весь алгоритм извлечения выборки.

Algorithm SS *(извлечение выборки)* для извлечения несмещенной выборки $1 \le M \le N$ предметов из множества N предметов.

Шаг 0. [Инициализация] **Set** $K \leftarrow 0$; **and** $Q \leftarrow 0$. (K+1 указывает рассматриваемый предмет, а Q обозначает число уже выбранных предметов.)

Шаг 1. [Цикл] **Do through** шаг 4 **while** $Q<M$ **od; and** STOP.

Шаг 2. [Генерация случайного числа] Генерируется (с равномерным распределением) случайное число Y между 0 и 1.

Шаг 3. [Проверка: взять или отвергнуть] **If** $(N—K)Y<M—Q$ **then** выбрать предмет K+l; **and** set Q←Q+l **fi.**

Шаг 4. [Увеличение K] **Set** $K \leftarrow K+1$.

На первый взгляд вовсе не очевидно, что алгоритм SS выбирает в точности M предметов и что выборка будет несмещенной, т. е. что все предметы будут выбираться с равной вероятностью.

Теорема 1. *Алгоритм SS выбирает точно M предметов.*

Доказательство. Очевидно, что невозможно выбрать более M предметов, так как алгоритм прекращает работу, как только выбран M-й предмет.

На шаге 3, если уже выбрано q предметов и остается просмотреть только $M—q$ предметов, то все остающиеся предметы будут выбраны. Таким образом, выбирается по меньшей мере M предметов. Следовательно, выбирается точно M предметов.

Теорема 2. *Алгоритм SS выбирает каждый предмет несмещенным образом.*

Рассмотрим (k+1)-й предмет. При любом испытании вероятность принятия этого предмета равна $(M—q)/(N—k)$, где q — число уже выбранных предметов. Значение q является случайной переменной, зависящей от выборов, сделанных при первых k пробах. Мы хотим показать, что при любых k, M и N *среднее* значение величины $(M—q)/(N—k)$ равно в точности M/N. Именно в этом смысле алгоритм SS является несмещенным.

Так как полное доказательство теоремы 2 слишком сложно, мы просто продемонстрируем его правильность для $k=2$.

Так как $k=2$, рассмотрим предмет 3. Следующая таблица возможных исходов для первых двух предметов не нуждается в пояснениях. Для указанных действий с первыми двумя предметами показаны значения q и вероятности выбора предмета 3.

Предмет 1	Предмет 2	q	P (выбрать предмет 3)
Принять	Принять	2	$(M-2)/(N-2)$
Принять	Отвергнуть	1	$(M-1)/(N-2)$
Отвергнуть	Принять	1	$(M-1)/(N-2)$
Отвергнуть	Отвергнуть	0	$M/(N-2)$

Вероятность того, что $q=2$, равна

$$P(q=2)=P\ (\text{принять предмет 1})\ [P\ (\text{принять предмет 2}\,|\,\text{предмет 1 принят})]=\frac{M}{N}\,\frac{M-1}{N-1}.$$

Аналогично мы можем вычислить остальную часть вероятностной меры для q:

$$P(q=1)=\frac{M}{N}\left(1-\frac{M-1}{N-1}\right)+\left(1-\frac{M}{N}\right)\frac{M}{N-1}=\frac{2M(N-M)}{N(N-1)};$$

$$P(q=0)=\left(1-\frac{M}{N}\right)\left(1-\frac{M}{N-1}\right)=\frac{(N-M)(N-M-1)}{N(N-1)}.$$

Убедитесь, что $P(q=2)+P(q=1)+P(q=0)=1$.

Среднее значение P (принять предмет 3) равно

$$P_{\text{сред}}(k=3)=\frac{M-2}{N-2}P(q=2)+\frac{M-1}{N-2}P(q=1)+$$
$$+\frac{M}{N-2}P(q=0)=\frac{M}{N}\ \text{(проверьте это!).}$$

Очевидно, что в худшем случае алгоритм SS просмотрит все предметы, прежде чем получит полную выборку из *M* предметов. В лучшем случае будет просмотрено только *M* предметов. В качестве упражнения попробуйте доказать, что среднее значение *k,* при котором алгоритм SS прекращает работу, равно

$$k_{\text{сред}}=\frac{M(N+1)}{M+1}.$$

Дадим еще один метод выбора *M* чисел из *N*.

Algorithm SELECT *(выбор)* для извлечения несмещенной выборки $1\le M\le N$ чисел из набора 1, 2, . . ., *N*. Выбранные числа помещаются в массив SELECT (1),. . ., SELECT (M).

Шаг 0. [Инициализация] **For** *I*←1 **to** *N* **do set** ITEM(I)←1 **od**; **set** LAST ←*N;* **and set** L ←1.

Шаг 1. [Повторение *M* раз] **Do through** шаг 4 **while** $L\le M$ **od**; **and** STOP.

Шаг 2. [Генерация случайного числа] Генерируется с равномерным распределением случайное число *K* между 1 и LAST.

Шаг 3. [Выбор следующего числа] **Set** SELECT *(L)* ←ITEM(K); **set** *L*←*L+1.*

Шаг 4. [Уменьшение размера выборки] **Set** ITEM(K) ← ITEM(LAST); **and** LAST ←LAST— 1.

Рисунок иллюстрирует применение алгоритма SELECT на примере выбора 5 чисел из 10.

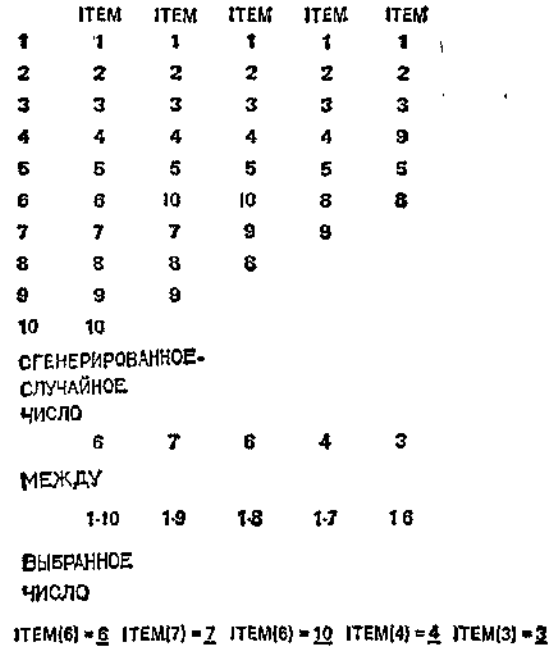


Рис. Пример работы алгоритма SELECT (выбор).

Легко доказать, что алгоритм является несмещенным. Пусть $p_i\,(k)$ — вероятность выбора 1-го числа на k-й итерации алгоритма SELECT. Тогда вероятность p_i выбора числа i равна

$$p_i = \sum_{k=1}^{M} p_i\,(k),$$

где

$$p_i\,(k) = \frac{N-1}{N} \times \frac{N-2}{N-1} \times \frac{N-3}{N-2} \times \ldots \times \frac{N-(k-1)}{N-(k-2)} \times \frac{1}{N-(k-1)} = \frac{1}{N}.$$

Таким образом,

$$p_i = \sum_{k=1}^{M} \frac{1}{N} = \frac{M}{N}.$$

Заметим, что для выбора M различных чисел из N при использовании алгоритма SELECT должно генерироваться только фиксированное количество M случайных чисел. В действительности объем необходимой работы равен $O(N+M)$, т. е. алгоритм SELECT выполняет:
N+2 операторов присваивания на шаге 1;

4*M* операторов присваивания на шаге 2;
M обращений к генератору случайных чисел на шаге 3.
Сравните это с алгоритмом SS, выполняющим:
2 оператора присваивания на шаге 0;
O(N) обращений к генератору случайных чисел на шаге 2;
O(N) проверок на шаге 3;
3M операторов присваивания на шаге 3;
O(N) сложений на шаге 4.
Заметим также, что в качестве побочного продукта алгоритм SELECT дает случайную перестановку чисел 1, . . ., *N,* что еще увеличивает его полезность.

2.7. Генераторы случайных чисел

Ранее мы определили *дискретные случайные переменные,* т. е. случайные переменные, множества значений (напоминаем, случайная переменная является функцией) которых конечны или являются бесконечными счетными множествами. Случайные переменные, множества значений которых — бесконечные несчетные множества, называются *непрерывными.* Хотя большинство элементарных приложений теории вероятности и статистики в вычислительной математике используют только дискретные случайные переменные, существует два важных непрерывных распределения, возникающих столь часто, что они заслуживают специального рассмотрения.
Случайная переменная *X* является непрерывной, если существует неотрицательная функция *f(x),* определенная на всей вещественной прямой и обладающая тем свойством, что если S — произвольное множество вещественных чисел, то

$$P(X \in S) = \int_S f(x)\,dx.$$

Функция *f(x)* называется *плотностью распределения вероятности X.* В частности, если S — вся вещественная прямая, то, поскольку *X* обязательно принимает некое значение,

$$1 = P(X \in (-\infty,\ \infty)) = \int_{-\infty}^{\infty} f(x)\,dx.$$

Аналогично, если S — некоторый интервал [*a, b*] на вещественной прямой, то

$$P(a \leqslant X \leqslant b) = \int_a^b f(x)\,dx.$$

Непрерывным аналогом дискретной переменной с равновероятным распределением является *равномерно распределенная переменная.* Если X — случайная переменная, принимающая с одинаковой вероятностью любое значение в интервале [0, 1], ее плотность вероятности равна

$$f(x)=\begin{cases} 1, & 0\leqslant x\leqslant 1; \\ 0, & \text{в остальных случаях.} \end{cases}$$

Тогда, например,

$$P(1/4\leqslant X\leqslant 3/4)=\int_{1/4}^{3/4} 1\,dx=3/4-1/4=1/2;$$

$$P(X=1/2)=\int_{1/2}^{1/2} 1\,dx=1/2-1/2=0.$$

Последний результат особенно важен. Вероятность того, что непрерывная случайная переменная окажется равной некоторому частному значению из своего интервала, всегда равна 0, а не *f(X).* Невозможно поставить в соответствие конечную вероятность каждому из бесконечного множества значений, так как «сумма» всех этих вероятностей окажется бесконечной (а не единицей). Не забывайте об этом.
Равномерное распределение на интервале [0,1] (иногда для обозначения этого распределения мы будем пользоваться символом U(0, 1) —или (0,1); имеет ли это какое-нибудь значение? — по существу является именно тем распределением, которое моделируют программные генераторы случайных чисел. То есть случайное число, генерируемое компьютером, по существу представляет собой выборку из равномерно распределенной переменной.
Математическое ожидание и дисперсия непрерывной случайной переменной определяются аналогично соответствующим величинам для дискретной случайной переменной, за исключением того, что суммы заменяются на интегралы. Таким образом,

$$E[X]=\int_{-\infty}^{\infty} xf(x)\,dx;$$

$$\operatorname{var}[X]=\int_{-\infty}^{\infty} (x-E[X])^2 f(x)\,dx.$$

Для равномерно распределенной переменной U в [0,1] получим

$$E[U]=\int_{-\infty}^{\infty} xf(x)\,dx=\int_{0}^{1} x\,dx=\frac{x^2}{2}\Big|_0^1=\frac{1}{2};$$

$$\operatorname{var}[U]=\int_{-\infty}^{\infty}(x-E[X])^2 f(x)\,dx=\int_{0}^{1}\left(x-\frac{1}{2}\right)^2 dx=\frac{1}{12}.$$

Еще одно важное непрерывное распределение — это *нормальное распределение.* Плотность вероятности для случайной переменной *X,* обладающей нормальным распределением с параметрами μ и σ^2, задается формулой

$$f(X)=\frac{1}{\sqrt{2\pi}\sigma}e^{[-(x-\mu)^2]/2\sigma^2},\quad -\infty<x<\infty.$$

Это знаменитая «колоколообразная кривая теории вероятностей». График этой функции приведен на рис. Кривая симметрична относительно μ, имеет резко заостренный вид при малых значениях σ^2 и более «сглаженный» при больших.

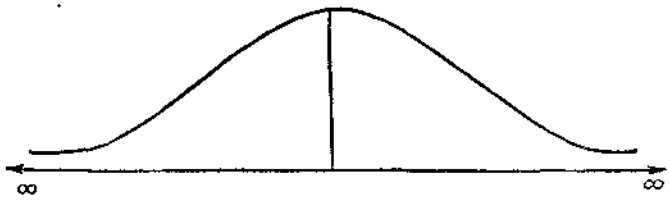


Рис. Функция плотности нормального распределения.

Нормальное распределение с параметрами μ и σ^2 обозначается как $N(\mu, \sigma^2)$. Распределение N (0,1) называется *стандартным нормальным.*

Простые, но несколько трудоемкие выкладки показывают, что если *X* — нормальная случайная переменная с параметрами μ и σ^2, то

$$E[X]=\mu \text{ и } \operatorname{var}[X]=\sigma^2.$$

Неожиданно большое количество случайных переменных обладает нормальным распределением или близким к нормальному. Многие из этих случайных переменных испытывают суммарное влияние со стороны других случайных переменных. Теоретически это следует из ряда важных теорем, известных под совместным названием *центральных предельных теорем.* Грубо говоря, эти теоремы гласят, что случайная переменная, являющаяся суммой многих других случайных переменных, приближается к нормально распределенной.

Можно показать, что $\bar{X}$ и s^2 являются несмещенными и состоятельными оценками с минимальной дисперсией для μ и σ^2 любой нормально

распределенной случайной переменной. В тех случаях, когда предполагается, что случайная переменная отражает суммарное влияние других случайных переменных, нормальное распределение с параметрами $\vec{X}$ и s^2 (основанное на статистических наблюдениях) может оказаться хорошей оценкой для всего неизвестного распределения случайной переменной.

Ранее мы уже видели, что важно иметь машинно-генерируемые «случайные числа» для облегчения тестирования и анализа алгоритмов. Следовательно, важно иметь эффективные алгоритмы для генерации случайных чисел.

Машинные генераторы случайных чисел чаще всего генерируют значения выборки случайной переменной, распределенной равномерно на интервале от 0 до 1. Остальные распределения случайных переменных можно затем легко моделировать, используя это основное распределение. Итак, алгоритм, генерирующий случайные числа, пытается выбрать число между 0 и 1 так, чтобы вероятность того, что число лежит в некотором частном подынтервале единичного интервала, была равна длине этого частного подынтервала.

Одним из очевидных механических средств достижения этого является колесо с единичной окружностью. Это колесо вращается против конца очень острой иглы, и координата точки окружности против конца иглы служит очередным случайным числом. Почему такое устройство не вводят в состав аппаратуры компьютера? Такие колеса не применяются потому, что случайные числа можно генерировать быстро, дешево, точно и почти так же надежно с помощью некоторых простых алгоритмов.

В действительности эти алгоритмы не генерируют случайных чисел с той неопределенностью, которая присуща вращающемуся колесу. На самом деле они используют простые арифметические операции для *детерминированной* выработки последовательности чисел, которые только *кажутся* случайными. Эти числа можно предсказать *до того,* как они будут сгенерированы. Этот факт, по-видимому, раздражает некоторых, но на самом деле для этого нет оснований. Последовательности *псевдослучайных* чисел, генерируемых этими алгоритмами, оказываются случайными и удовлетворяют ряду сложных статистических тестов, что в результате «узаконивает» их как удовлетворительные.

Один из ранних генераторов случайных чисел, принадлежащий Джону фон Нейману (1946), известен как *метод середины квадрата.*

Algorithm MS *(середина квадрата)* для генерации K -разрядных псевдослучайных чисел. Должно быть задано K -разрядное начальное

число X_0. Для удобства предполагаем, что K четно и нужна последовательность $X(1), X(2), \ldots, X(M)$ из M случайных чисел.
Шаг 0. [Инициализация] **Set** $X \leftarrow X_0$.
Шаг. 1. [Основной цикл] **For** $J \leftarrow 1$ **to** M **do** шаг 2 **od; and** STOP.
Шаг 2. [Генерация нового случайного числа X(J)] **Set** $Y \leftarrow X^2$;
$X(J) \leftarrow$ (средние K разрядов числа Y); **and** $X \leftarrow X(J)$. (Число Y удет иметь $2K$ разрядов, а следующее случайное число $X(J)$ получается, если удалить по $K/2$ разрядов с каждого конца Y. Ясно, что десятичная точка помещается перед первым разрядом числа $X(J)$ до поступления его на выход случайного генератора.)

Для K =4 и X_0=2134 первые 10 псевдослучайных чисел, выработанные алгоритмом MS, будут:

$X_0^2 = 04\ 5539\ 56 \quad X_1 = 0.5539$
$X_1^2 = 30\ 6805\ 21 \quad X_2 = 0.6805$
$X_2^2 = 46\ 3080\ 25 \quad X_3 = 0.3080$
$X_3^2 = 09\ 4864\ 00 \quad X_4 = 0.4864$
$X_4^2 = 23\ 6584\ 96 \quad X_5 = 0.6584$
$X_5^2 = 43\ 3490\ 56 \quad X_6 = 0.3490$
$X_6^2 = 12\ 1801\ 00 \quad X_7 = 0.1801$
$X_7^2 = 03\ 2436\ 01 \quad X_8 = 0.2436$
$X_8^2 = 05\ 9340\ 96 \quad X_9 = 0.9340$
$X_9^2 = 87\ 2356\ 00 \quad X_{10} = 0.2356$

Несмотря на видимую случайность чисел, генерируемых алгоритмом MS, ему свойственны недостатки, которые привели к тому, что им избегают пользоваться в серьезных приложениях. Убедитесь, что если в последовательности когда-нибудь появится 0.0000, то все следующие за ним числа будут также 0.0000. Генерация случайных чисел в алгоритме MS может вырождаться различными путями. Для K =4 и X_0=1357 последовательность оказывается следующей:

$X_1=0.8414$
$X_2=0.7953$
$X_3=0.2502$
$X_4=0.2600$
$X_5=0.7600$
$X_6=0.7600$
$X_7=0.7600$,
.
.
.
$X_n=0.7600$ для $n\geqslant 5$.

Это вряд ли похоже на случайные числа. Даже при более тщательном выборе K и X_o последовательности, генерируемые этим методом, склонны к быстрому «зацикливанию» и работают неудовлетворительно при проверке по сложным статистическим критериям.

Среди различных детерминированных арифметических процедур, генерирующих случайные числа, шире всего используется класс алгоритмов, называемый *линейными конгруэнтными методами.* Общий вид таких алгоритмов приведен ниже.

Algorithm LCM *(линейный конгруэнтный метод)* для генерации последовательности *X(1), X(2), . . ., X(M)* из *M* псевдослучайных чисел. Должны быть заданы следующие входные значения:

X (0) = начальное «необработанное» случайное число *X* (0)≥1; целое

B = множитель *B≥1;* целое

K = шаг *K≥0*; целое

MOD = модуль *MOD>X(0), B, K;* целое

Шаг 1. [Основной цикл] **For** *J*← 1 **to** *M* **do through** шаг 3 **od; and** STOP.

Шаг 2. [Генерация нового необработанного случайного числа] **Set** $X(J) \leftarrow (B*X(J-1)+K) \pmod{\text{MOD}}$

(По определению, если *a* (mod *m)=x* для целых *a* и *m,* то *x* есть (целый) остаток от деления *a* на *m,* Например, 5(mod3)=2.)

(Число *X (J)* должно лежать в интервале 0≤X(J)< MOD.)

Шаг 3. [Генерация следующего случайного числа] **Set** *Y (J)* ←X(J)/MOD. *(Y(J)* будет лежать в интервале 0≤ *Y(J)* <1 и будет обладать желаемым распределением.)

Возьмем $K=0$, $\text{MOD}=2^{10}$, $B=101$ и $X(0)=432$. Первые числа, выработанные алгоритмом LCM, будут

$X_1=624 \quad Y_1=624/1024=0.610$
$X_2=560 \quad Y_2=560/1024=0.546$
$X_3=240 \quad Y_3=240/1024=0.234$
$X_4=688 \quad Y_4=688/1024=0.673$
$X_5=880 \quad Y_5=880/1024=0.859$
$X_6=816 \quad Y_6=816/1024=0.796$
$X_7=496 \quad Y_7=496/1024=0.484$
$X_8=944 \quad Y_8=944/1024=0.923$

Существует теория по вопросу «хорошего» выбора *K, B,* MOD и X(0), которая выходит за рамки этой книги. Для наших целей достаточно просто утверждения, что существует такой выбор этих параметров, при котором алгоритм LCM будет генерировать числа в интервале [0, 1), представляющиеся непредсказуемыми и удовлетворяющие определенным статистическим критериям. Для всех практических целей эти числа оказываются последовательностью наблюдений равномерно распределенной случайной переменной.

Равномерно распределенные случайные переменные можно, в частности, использовать при моделировании случайных переменных с другими распределениями. Начнем с двух относительно простых примеров.

Пусть *X* — дискретная случайная переменная со следующим распределением:

$P(X=1)=0.1$
$P(X=2)=0.2$
$P(X=3)=0.4$
$P(X=4)=0.2$
$P(X=5)=0.1$

Как можно получить от компьютера «выборки» из *X*, такие, что значение X=1 встретится приблизительно в 10% случаев, значение X=2 приблизительно в 20% случаев и т. д.

Так как сумма всех вероятностей в распределении равна единице, проще всего разбить интервал [0, 1) на пять подынтервалов и назначить каждое возможное значение *X* подынтервалу, длина которого равна вероятности того, что *X* реально принимает это значение. В нашем примере мы могли бы установить следующее соответствие:

$X=1 \leftrightarrow [0,\ 0.1)$
$X=2 \leftrightarrow [0.1,\ 0.3)$
$X=3 \leftrightarrow [0.3,\ 0.7)$
$X=4 \leftrightarrow [0.7,\ 0.9)$
$X=5 \leftrightarrow [0.9,\ 1.0)$

Для машинного моделирования случайной переменной *X* генерируют стандартное (равномерно распределенное) случайное число Z. Если $Z \in [0,\ 0.1)$, берем X=1; если $Z \in [0.1,\ 0.3)$, берем *X=2.* После извлечения большого числа выборок наблюдаемое распределение частот *X* должно быть довольно близко к искомому распределению.

Только что описанная процедура может применяться к произвольной дискретной случайной переменной на конечном интервале; ее также можно применять (приближенно) для многих случайных переменных, интервал значений которых представляет собой бесконечное счетное множество. Этот метод широко используется при разработке алгоритмов моделирования.

Теперь рассмотрим использование генератора равномерно распределенных случайных чисел. Хороший простой алгоритм основывается непосредственно на центральной предельной теореме и на преобразовании, с помощью которого можно превратить распределение $N(\mu,\ \sigma^2)$ в *N (0,* 1) и наоборот.

Следующая теорема является простейшей из центральных предельных теорем, и именно ее часто называют центральной предельной теоремой.

Теорема 1*(центральная предельная теорема). Пусть* $X_1, X_2, \ldots, X_n$ — *последовательность независимых, одинаково распределенных случайных переменных со средним* μ *и дисперсией* σ^2. *Распределение вероятности для случайной переменной*

$$Z = \frac{X_1 + X_2 + \ldots + X_n - n\mu}{\sigma\sqrt{n}}$$

стремится к N (0, 1) *при* $n \to \infty$.

Теорема 2. *Если случайная переменная Y имеет распределение* $N(\mu,\ \sigma^2)$, *то переменная*

$$Z = \frac{Y - \mu}{\sigma}$$

имеет распределение N (0, 1). *Обратное также верно.*

Доказательство теоремы 1требует некоторой квалификации и поэтому опущено; доказательство теоремы 2 очевидно и оставлено в качестве упражнения.

Algorithm NRN *(нормальные случайные числа)* для генерации нормально распределенных случайных чисел с использованием генератора равномерно распределенных случайных чисел RANN. Равномерно распределенные случайные числа применяются для выработки случайных чисел с распределением *N (0,* 1) на основе теоремы 1. Случайные числа с распределением *N (0,* 1) преобразуются в случайные числа $N(\mu,\ \sigma^2)$с использованием теоремы 2. Необходимо задать на

входе в алгоритм значения n, μ, σ^2. Предполагается, что нужна последовательность $Y(1), Y(2), \ldots, Y(M)$ из M случайных чисел.

Шаг 1. [Основной цикл] **For** $J \leftarrow 1$ **to** M **do through** шаг 3 **od; and** STOP.

Шаг 2. [Генерация нового случайного числа с распределением $N(0, 1)$] CALL RANN n раз; на выходе процедуры — последовательность $X_1, \ldots, X_n$; **set**

$$Z(J) \leftarrow \frac{X_1 + X_2 + \ldots + X_n - (n/2)}{\sqrt{n/12}}$$

(Так как каждое X_k обладает равномерным распределением со средним 1/2 и дисперсией 1/12, то $Z(J)$ будет обладать распределением $N(0, 1)$ согласно теореме 1. Практически для большинства применений достаточно n=10.)

Шаг 3. [Генерация нового числа с распределением $N(\mu, \sigma^2)$] **Set** $Y(J) \leftarrow \sigma Z(J) + \mu$.

Теперь рассмотрим две более совершенные процедуры. Начнем с общей задачи моделирования произвольного непрерывного распределения с использованием равномерного распределения.

Для любой случайной переменной X *интегральная функция распределения* определяется как

$$F(a) = P(X \leqslant a), \quad -\infty < a < \infty.$$

Легко убедиться, что любая интегральная функция распределения обладает следующими свойствами:

1) $F(a)$ — неубывающая функция a;
2) $\lim\limits_{a \to \infty} F(a) = 1$;
3) $\lim\limits_{a \to -\infty} F(a) = 0$;
4) $P(a < X \leqslant b) = F(b) - F(a)$ для всех $a < b$.

Если X — непрерывная случайная переменная с плотностью $f(x)$, то

$$F(a) = P(X \leqslant a) = \int_{-\infty}^{a} f(x)\, dx$$

и по основной теореме интегрального исчисления

$$\frac{d}{da} F(a) = f(a).$$

Если Y обладает равномерным распределением на [0, 1), то его интегральная функция распределения равна

$$G(a) = P(Y \leqslant a) = \begin{cases} 0 & \text{при } a \leqslant 0; \\ \int_0^a 1\, dy = a & \text{при } 0 < a \leqslant 1; \\ 1 & \text{при } 1 < a. \end{cases}$$

Пусть случайная переменная *X* имеет интегральную функцию распределения *F*, и пусть случайная переменная равномерно распределена на [0, 1). Предположим, что обратная функция F^{-1} ведет себя достаточно хорошо. Мы хотим показать, что случайная переменная, определяемая формулой $X' = F^{-1}(Y)$, имеет интегральную функцию распределения *F*. Тогда, если $\{y_1, y_2, \ldots, y_n\}$— последовательность равномерно распределенных чисел, то последовательность $\{x_1 = F^{-1}(y_1), x_2 = F^{-1}(y_2), \ldots, x_n = F^{-1}(y_n)\}$ будет иметь распределение *F*.
Таким образом, мы хотим показать, что

$$P(X' \leqslant x) = F(x), \quad \text{если} \quad X' = F^{-1}(Y),$$

а *Y* обладает равномерным распределением. Это доказывается так:

$$P(X' \leqslant x) = P(F^{-1}(Y) \leqslant x) = P(Y \leqslant F(x)) = \int_0^{F(x)} 1\, dy = F(x).$$

Мы уже отмечали, что $F(x)$ всегда лежит между 0 и 1; здесь мы также использовали интегральную функцию распределения для *Y*.
Хотя эта процедура в принципе весьма обща и проста, она применяется не так часто, как можно было бы ожидать. В первую очередь это обусловлено тем, что трудно получить явные аналитические выражения для обратных функций многих распределений. Однако мы рассмотрим один важный случай, когда этот метод работает очень хорошо.
Экспоненциальное распределение (с параметром λ) имеет функцию плотности

$$f(x) = \lambda e^{-\lambda x}, \quad 0 < x < \infty;$$
$$= 0 \qquad \text{в остальных случаях.}$$

Интегральная функция распределения для экспоненциально распределенной случайной переменной задается выражением

$$F(x) = 0 \qquad \text{при } x \leqslant 0;$$
$$= \int_0^x \lambda e^{-\lambda x}\, dx = (1 - e^{-\lambda x}) \quad \text{при } x > 0.$$

Это распределение является фундаментальным в теории очередей. Экспоненциально распределенные случайные переменные хорошо подходят для моделирования интервалов между прибытиями. Если *Y* распределена равномерно, то

$$Y = F(X) = 1 - e^{-\lambda X}.$$

Обращая F (т. е. решая относительно X), получим

$$X = -\frac{1}{\lambda} \ln (1 - Y).$$

Таким образом, если $\{y_1, y_2, \ldots, y_n\}$ — равномерно распределенные случайные числа, то

$$\left\{x_1 = -\frac{1}{\lambda} \ln (1 - y_1),\ x_2 = -\frac{1}{\lambda} \ln (1 - y_2),\ \ldots,\ x_n = -\frac{1}{\lambda} \ln (1 - y_n)\right\}$$

будут экспоненциально распределенными случайными числами. Невозможно воспользоваться этой процедурой с обратной функцией для генерации нормально распределенных случайных чисел. Кроме того, алгоритм NRN (нормально распределенные числа) имеет неприятную особенность — он требует n равномерно распределенных случайных чисел для генерации единственного случайного числа с нормальным распределением. Большая важность нормального распределения послужила причиной построения нескольких эффективных алгоритмов генерации случайных чисел с нормальным распределением. Следующий алгоритм демонстрирует один из таких методов.

Algorithm PNRN *(полярный метод генерации случайных чисел с нормальным распределением)* для генерации двух независимых случайных чисел с распределением $N(0, 1)$ из двух независимых, равномерно распределенных случайных чисел. Распределение $N(0, 1)$ преобразуется в $N(\mu, \sigma^2)$ с использованием теоремы 2.

Шаг 1. [Генерация двух равномерно распределенных случайных чисел] Генерируются два независимых случайных числа U_1 и U_2 с распределением $U(0, 1)$; **set** $V_1 \leftarrow 2U_1 - 1$; **and** $V_2 \leftarrow 2U_2 - 1$. (V_1 и V_2 имеют распределение $U(-1, +1)$.)

Шаг 2. [Вычисление и проверка S] **Set** $S \leftarrow V_1^2 + V_2^2$; **if** $S \geqslant 1$ **then go to** шаг 1 **fi.**

Шаг 3. [Вычисление N_1 и N_2] **Set** $N_1 \leftarrow V_1 \sqrt{(-2 \ln S)/S}$; **and** $N_2 \leftarrow V_2 \sqrt{(-2 \ln S)/S}$; **and** STOP. *($N_1$ и N_2 распределены нормально.)*

Алгоритм PNRN имеет важное вычислительное преимущество перед алгоритмом NRN, а именно: он обычно генерирует по одному числу с нормальным распределением на каждое число с равномерным распределением. Компенсируется ли этим дополнительное время, затрачиваемое на вычисление натуральных логарифмов и квадратных корней?

Доказательство правильности алгоритма PNRN существенно опирается на интегральное исчисление и аналитическую геометрию; оно оставлено в качестве упражнения.

3. Статистические решения

Человеческому обществу и всем его частям, вплоть до отдельных личностей, в процессе своей жизнедеятельности приходится постоянно выполнять различные действия, направленные на достижение определенных целей. Человек разумный отличается способностью планирования своих действий, т. е. он сначала принимает решение (желательно, хорошее), выбирая его из множества возможных. Искусство принятия хороших решений приходит с опытом, и немаловажен здесь природный дар. Можно с уверенностью сказать, что все выдающиеся деятели (ученые, изобретатели, политики, военачальники и т. д.) достигли своих успехов именно благодаря умению принимать правильные решения в своей области деятельности.

Задачи обработки МИ тоже могут быть сформулированы как задачи принятия решений. Например, при обнаружении объектов на фоне МИ нужно выбрать одно из решений: «есть объект» или «нет объекта», а при оценке параметров МИ нужно выбрать какой-то набор их возможных значений.

Было бы весьма полезно иметь математический аппарат для нахождения оптимальных решений. Для ряда ситуаций таким аппаратом является теория статистических решений , основные положения и результаты которой приводятся в этом разделе.

3.1. Решения

Решение – это результат целенаправленной обработки имеющейся информации. Приведем несколько примеров. Решение может заключаться в выборе одного из действий в бизнесе, политике, лечении больного и т. д. Оценка параметра – тоже решение, состоящее в выборе какого-то числа из массы возможных значений. В задаче обнаружения выбирается одно из двух решений – есть объект или его нет. **В задаче распознавания принимаются решения о принадлежности исследуемого объекта к одному из возможных классов.**

Решение, таким образом, является очень общим понятием. Любая задача подразумевает принятие некоторого решения.

Отметим некоторые особенности, которые следует учитывать, принимая решения.

1^0. Решение направлено на достижение некоторой **цели** и приводит к некоторым **последствиям**, по которым должно оцениваться качество решения.

2^0. Решение выбирается из **множества альтернатив** (возможных решений), конечного или бесконечного. При построении математической теории множество альтернатив предполагается **определенным**. Это предположение не ограничивает общности теории, так как новые, нестандартные решения можно включить в множество уже имеющихся альтернатив.

3^0. Решение принимается по доступной к моменту его принятия **информации**, состоящей из двух принципиально различных частей. Первая ее часть обобщает весь прошлый опыт, т. е. **априорна**. Вторая часть – совокупность данных наблюдения – получается непосредственно в процессе выработки решений. Эта **апостериорная** совокупность данных является объектом обработки в процессе принятия решения. Например, это может быть наблюдаемым И, набором анализов, состоянием биржи и т. д.

В практических ситуациях соотношение объемов этих двух частей информации может меняться в широких пределах. В частности, одна из них может полностью отсутствовать.

4^0. Очень часто доступная информация и последствия от принятия решения имеют **статистическую природу** из-за ненаблюдаемости скрытых случайных факторов. Поэтому и процедура принятия решений должна иметь статистический характер. Именно такой случай рассматривается в теории статистических решений.

5^0. Решение часто заключается в принятии совокупности частичных решений, т. е. решение может быть векторным или даже многомерным.

Например, в различных задачах обработки И результатом является совокупность частных решений, касающихся отдельных элементов наблюдаемого И.

6^0. Выбор решения из множества альтернатив не всегда однозначно определяется имеющейся информацией. Он может (а иногда и должен) допускать элементы случайности. Это так называемые **рандомизированные решения**. Такие решения используются, например, в теории игр. И в реальной жизни мы иногда принимаем решение, бросая монетку.

Задачей является выбор такого решения, которое приводило бы к наиболее благоприятным последствиям. Соответствующие **правила принятия решений** называются **оптимальными**. Для их нахождения разработан и постоянно совершенствуется мощный математический аппарат (теория игр; линейное, нелинейное и динамическое программирование; теория статистических решений и т. д.), позволяющий рассматривать очень многообразные задачи с общих математических позиций.

Таким образом, решения выбираются из множества альтернатив $U = \{u\}$. Решение выбирается с использованием данных (наблюдений, измерений, выборки) z, всевозможные значения которых составляют множество $Z = \{z\}$. При этом имеются некоторые скрытые, ненаблюдаемые параметры θ, описывающие реальную ситуацию и принимающие значения из множества $\Theta = \{\theta\}$ всех возможных ситуаций.

Множество альтернатив U может быть самым разнообразным (совокупность оценок параметров, управлений, и прочих акций). Математически же можно выделить три случая.

а) $U = \{u_1, u_2, \dots\}$, т. е. пространство решений дискретно (конечно или счетно). Например, в задачах обнаружения и распознавания объектов.

б) Пространство решений непрерывно (ограничено или бесконечно). Например, в задаче оценки параметров.

в) Пространство дискретно–непрерывно: $U = \{U_1, U_2, \ldots, U_K\}$, где все или некоторые из U_i непрерывны. Например, задача обнаружения объектов с оценкой их параметров (координат, скорости и т. д.).

Правила принятия решений могут быть нерандомизированными и рандомизированными.

Нерандомизированные правила определяют для каждого наблюдения z вполне определенное решение u. Такие правила имеют вид $u = u(z)$, т. е. являются функциями, преобразованиями $z \to u$.Совокупность различных преобразований $u(z)$ (любых или из некоторого ограниченного класса) образует множество $U(z)$ всех нерандомизированных решающих правил, из которого и нужно выбирать оптимальное правило.

Отметим, что U и $U(z)$ – разные множества. Например, в задаче обнаружения объекта $U = \{H_0, H_1\}$, где $H_0 =$ (нет объекта) и $H_1 =$ (есть объект), а множество $U(z)$ – это набор всевозможных правил $\boldsymbol{u(z)}$, которые каждому возможному наблюдению z ставят в соответствие определенное решение H_0 или H_1. Геометрически это можно представить в виде рис.3.1, на котором пространство Z условно представлено прямоугольником. Пусть $\boldsymbol{u = u(z)}$ – некоторое решающее правило, которое на одной части Z_0 наблюдений принимает значение H_0, а на остальной части Z_1 – значение H_1.

Таким образом, $U(z)$ геометрически есть набор всевозможных разбиений Z на два подпространства Z_0 и Z_1.

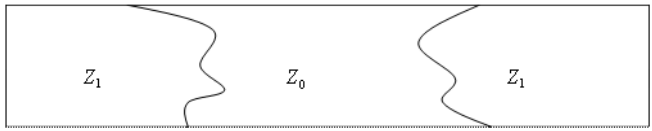


Рис.3.1.

Рандомизированные правила определяются заданием на U условной вероятностной меры $\Phi(u \mid z)$, т. е. при имеющемся наблюдении Z_0 решение u выбирается случайным образом с распределением вероятностей $\Phi(u \mid z_0)$. Нерандомизированное правило $u(z)$ есть частный, вырожденный случай рандомизированного, когда мера $\Phi(u \mid z)$ целиком сосредоточена в точке $u(z)$.

3.2. Потери и риск

Последствия от принятия решений следует оценивать по степени их соответствия поставленной цели. Это соответствие, в принципе, может быть определено с помощью количественной меры, определяющей выигрыш или потери от принятия решения. Эта мера называется **функцией потерь (штрафа)** или **функцией выигрыша (целевой функцией)**. Для определенности будем использовать функцию потерь, которую, естественно, следует **минимизировать**, чтобы решение было оптимальным.

Потери могут зависеть не только от принятого решения u, но и от реальной ситуации, описываемой параметрами θ. Возможно, что

потери зависят и от наблюдений z . В общем случае функция потерь имеет вид $g = g(u, \theta, z)$. В частных случаях может быть $g = g(u, \theta)$, $g = g(u, \theta_0) = g(u)$ и т. д.

Если бы значение θ было известно, т. е. полностью определена реальная ситуация θ_0 , то при получении наблюдения $z = z_0$ оптимальным решением была бы точка минимума u_0 функции $g(u) = g(u, \theta_0, z_0)$, т. е. **оптимальным решающим правилом** была бы просто минимизация известной функции по u при заданных значениях ее параметров θ и z . Полученное таким образом решение является каждый раз наилучшим.

В реальности же θ являются скрытыми параметрами, т. е. действительная обстановка может быть, в лучшем случае, известна только приближенно. В таком случае уже невозможно каждый раз находить наилучшее решение – неизбежны просчеты из-за ошибочной оценки ситуации.

Единственное, что можно сделать в сложившемся положении, это попытаться найти такое решающее правило, при котором минимальными будут **средние потери**, т. е. среднее значение функции потерь, называемое **риском. Правило, минимизирующее риск, будем называть оптимальным.**

Для нахождения среднего значения функции потерь $g = g(u, \theta, z)$ нужно задать распределение вероятностей на пространстве ее аргументов U, Θ, Z , что может быть сделано следующим образом.

Закон распределения вероятностей возможных ситуаций θ описывается ПРВ $W(\theta)$ на Θ .

Наблюдения z должны быть в той или иной мере связаны с θ, иначе в них нет никакой надобности. Эта связь может быть описана условной ПРВ $P(z \mid \boldsymbol{\theta})$ на пространстве наблюдений Z, которая называется **функцией правдоподобия** (ФП).

Решения u принимаются в зависимости от z, это и есть решающее правило. В наиболее общем случае это правило рандомизировано и определяется условной вероятностной мерой, для определенности заданной условной ПРВ $\boldsymbol{\varphi}(u \mid z)$.

Таким образом, совместная ПРВ параметров $u, \boldsymbol{\theta}, z$ есть функция $W(u, \theta, z) = W(\theta) P(z \mid \theta) \varphi(u \mid z)$, определенная на пространстве $U \times \Theta \times Z$.

В зависимости от степени усреднения можно рассматривать различные типы рисков. **Средний риск.** Если известно распределение θ и z (заданы ПРВ $W(\theta)$ и $P(z \mid \theta)$), то для любого решающего правила $\boldsymbol{\varphi} = \boldsymbol{\varphi}(u \mid z)$ можно найти безусловное математическое ожидание функции потерь, зависящее только от этого правила и называемое **средним риском:**

$$R(\boldsymbol{\varphi}) = M[g(u, \boldsymbol{\theta}, z)] = \int_{\Theta} \int_{Z} \int_{U} g(u, \boldsymbol{\theta}, z) \boldsymbol{\varphi}(u \mid z) P(z \mid \boldsymbol{\theta}) W(\boldsymbol{\theta}) du dz d\boldsymbol{\theta} \quad . \qquad (3.1)$$

Оптимизация принятия решений заключается в выборе такого правила (т. е. ПРВ $\boldsymbol{\varphi}(u \mid z)$), при котором средний риск (3.1) минимален.

Для нерандомизированых правил $u = u(z)$ распределение $\boldsymbol{\varphi}(u \mid z)$ сосредоточено в точке $u(z)$, т. е.

$$\varphi(u \mid z) = \delta(u - u(z)),$$

(3.2)

где $\delta(\cdot)$ – функция Дирака, и (3.1) принимает вид

$$R(u(z)) = \iint g(u(z), \theta, z) P(z \mid \theta) W(\theta) dz d\theta$$

. (3.3)

Здесь и в дальнейшем, если не возникает недоразумений, будем опускать обозначения областей интегрирования. Выражение (3.3) определяет средний риск для любого правила $u = u(z)$, а оптимизация принятия решений заключается в выборе такого правила $u(z)$, при котором риск (3.3) минимален.

Априорным риском называется величина

$$G(u) = \iint g(u, \theta, z) P(z \mid \theta) W(\theta) dz d\theta$$

, (3.4)

где u не зависит от z. Если, кроме того, $g(u, \theta, z) = g(u, \theta)$, то

$$G(u) = \iint g(u, \theta) P(z \mid \theta) W(\theta) dz d\theta = \int g(u, \theta) W(\theta) d\theta$$

. (3.5)

Выражения (3.4) и (3.5) определяют **априорную оценку потерь**, связанных с решением u, принимаемым при отсутствии наблюдений или при их игнорировании. Такое решение может быть спланировано заранее, еще до наступления момента, когда решение необходимо принимать. В такой постановке оптимальным решением является u^*, минимизирующее (3.4) или (3.5), оно учитывает только априорную информацию.

Апостериорным риском называется условное математическое ожидание функции потерь для данного решения u при данном значении z. Для этого найдем по формуле Байеса апостериорное условное распределение θ при заданном значении z:

$$W_a(\theta) = W(\theta \mid z) = \frac{P(z \mid \theta) W(\theta)}{\int P(z \mid \theta) W(\theta) d\theta}. \quad (3.6)$$

Апостериорный риск определится усреднением функции потерь по ПРВ (3.6):

$$R(u,z) = \int g(u,\theta,z) W(\theta \mid z) d\theta = \frac{\int g(u,\theta,z) P(z \mid \theta) W(\theta) d\theta}{\int P(z \mid \theta) W(\theta) d\theta}. \quad (3.7)$$

Средний и апостериорный риски связаны очевидным соотношением

$$R(\varphi) = \iint R(u,z) \varphi(u \mid z) P(z) du dz, \quad (3.8)$$

где

$$P(z) = \int P(z \mid \theta) W(\theta) d\theta - \quad (3.9)$$

безусловная ПРВ наблюдений z. В случае нерандомизированных правил соотношение (3.8) принимает вид

$$R(u(z)) = \int R(u(z), z) P(z) dz. \quad (3.10)$$

Условным риском называются средние потери при использовании правила $\varphi = \varphi(u \mid z)$ при заданном значении θ:

$$r(\varphi, \theta) = \iint g(u, \theta, z)\varphi(u \mid z)P(z \mid \theta)dudz, \qquad (3.11)$$

а в случае нерандомизированного правила $u = u(z)$:

$$r(u(z), \theta) = \int g(u(z), \theta, z)P(z \mid \theta)dz. \qquad (3.12)$$

Этот риск определяет потери в среднем по всем возможным наблюдениям для конкретного решающего правила $u = u(z)$ и конкретной ситуации θ.

3.3. Байесовы решающие правила

Если известны распределения $W(\theta)$ и $P(z \mid \theta)$, т. е. если имеется статистическое описание параметров θ и наблюдений z, то, в принципе, задача нахождения оптимального решающего правила легко решается (остаются только некоторые вычислительные проблемы). Само оптимальное решающее правило в этих условиях называется **байесовым**. Выведем это правило в общем случае.

Оптимальное решающее правило $\varphi = \varphi(u \mid z)$ должно минимизировать средний риск (3.8). В силу неотрицательности ПРВ $P(z)$ минимальное значение интеграла в (3.8) достигается, если при любом z минимален интеграл

$$\int R(u, z)\varphi(u \mid z)du.$$

Этот интеграл в силу неотрицательности $\varphi(u \mid z)$ минимален, когда распределение $\varphi(u \mid z)$ целиком сосредоточено в точке $u = u_0 = u_0(z)$ минимума апостериорного риска $R(u,z)$, т. е. при

$$\boldsymbol{\varphi}(u \mid z) = \boldsymbol{\delta}(u - u_0(z)).$$

Следовательно, байесово правило является нерандомизированным и при каждом наблюдении z оптимальным является решение, минимизирующее апостериорный риск. Этого и следовало ожидать, так как апостериорный риск – это риск при известном наблюдении.

Таким образом, задача построения оптимального решающего правила сводится к относительно простой задаче минимизации функции $R(u,z)$ на множестве альтернатив U при заданном наблюдении z.

Байесово решение $u_0(z)$ определяется из соотношения

$$R(u_0(z),z) = \min_u \int g(u,\theta,z)W(\theta \mid z)d\theta, \qquad (3.13)$$

что (с учетом независимости знаменателя в (3.6) и (3.7) от u) эквивалентно соотношению

$$\int g(u_0(z),\theta,z)P(z \mid \theta)W(\theta)d\theta = \min_u \int g(u,\theta,z)P(z \mid \theta)W(\theta)d\theta. \qquad (3.14)$$

Получаемый при этом минимальный средний риск

$$R^* = R(u_0(z)) = \int (\min_u \int g(u,\theta,z)P(z \mid \theta)W(\theta)d\theta)dz \qquad (3.15)$$

называется **байесовым**, его обеспечивает само байесово решение.

3.4. Многоальтернативные решения

Рассмотрим важный частный случай решения, заключающегося в выборе одной из m альтернатив $u = i = 0,1,\ldots,m-1$, т. е. $U = \{0,1,\ldots, m-1\}$. При этом $\Theta = \{j = 0,1,\ldots, n-1\}$, т. е. возможна одна из n ситуаций $\theta = j = 0,1,\ldots,n-1$. Этой схеме соответствуют задачи проверки гипотез, обнаружения и различения сигналов, распознавания образов и т. д. Пусть функция потерь не зависит от z и

$$g(u,\theta,z) = g(u = i,\theta = j) = g_{ij},$$

т. е. функция потерь задана $m \times n$**-матрицей потерь** $\{g_{ij}\}$. Тогда

$$R(u,z) = R(u = i,z) = \sum_j g_{ij} p(\theta = j \mid z) = \sum_j g_{ij} \frac{P(z \mid \theta = j)p(\theta = j)}{P(z)} =$$

$$= \frac{1}{P(z)} \sum_j g_{ij} p(\theta = j) P(z \mid \theta = j) = \frac{1}{P(z)} \sum_j a_{ij} P(z \mid \theta = j) = \frac{1}{P(z)} L_i, \qquad (3.16)$$

где $p(\theta = j)$ – априорные вероятности ситуаций; $P(z \mid \theta = j)$ – ФП; $P(z) = \sum_j p(\theta = j) P(z \mid \theta = j)$ – вероятность наблюдения z; $p(\theta = j \mid z)$ – условная вероятность ситуации j при наблюдении z; $a_{ij} = g_{ij} p(\theta = j)$.

Из (3.16) следует, что при имеющемся наблюдении z нужно выбрать такое решение $u = i$, при котором минимальна линейная

комбинация L_i ФП $P(z \mid \theta = j)$ с коэффициентами a_{ij}, т. е. нужно сравнить между собой m линейных комбинаций $L_0,\ldots,L_{m-1}$.

Отметим, что коэффициенты a_{ij} этих линейных комбинаций зависят от функции потерь g_{ij} и априорных вероятностей $p(\theta = j)$ ситуаций, но от наблюдений не зависят, т. е. концентрируют в себе априорную информацию. Значения же $P(z \mid \theta = j)$, входящие в L_i, напротив, зависят от наблюдений.

В хорошей информационной системе определяющую роль в принятии решения должны играть именно наблюдения. Если это не так, то решение будет приниматься в основном по априорной информации, а сама информационная система окажется практически бесполезной. В этом заключается общая закономерность систем обработки информации: чем более высокими качествами должна обладать информационная система, тем меньшее значение имеют априорные данные о характеристиках потерь и поведении параметров θ.

Для рассматриваемой задачи это означает, что основное значение должен иметь разброс значений $P(z \mid \theta = 0),\ldots,P(z \mid \theta = m-1)$, а не разброс коэффициентов $a_{i_0},\ldots,a_{i_{m-1}}$. А именно, существенно большим должно быть значение $P(z \mid \theta = j_0)$, соответствующее действительно имеющей место ситуации j_0. Другими словами, наблюдения в хорошей информационной системе должны достаточно точно идентифицировать имеющуюся ситуацию. В этом случае априорные сведения имеют очень малое влияние, поэтому их можно выбирать практически произвольно.

Но это все, конечно, только пожелания о качествах системы обработки информации. В действительности приходится работать с той системой,

какая есть. В любом случае оптимальное решение соответствует минимальной из линейных комбинаций L_i.

Двухальтернативные решения

Рассмотрим частный случай двухальтернативных задач, когда $m = n = 2$. Этому случаю соответствует, например, задача обнаружения сигналов или других объектов, в применении к которой и произведем все выкладки.

Итак, возможны две ситуации или гипотезы H_0 – нет сигнала и H_1 – есть сигнал. Решение состоит в выборе одной из этих гипотез. Заданы априорные вероятности $p(H_0)$ и $p(H_1)$ и функция потерь $g_{ij} = g(u = H_i, \theta = H_j)$. При этом g_{01} и g_{10} – потери при неверных решениях, а g_{00} и g_{11} – потери при верных решениях. Задана также ФП: $P(z \mid H_0)$ – распределение вероятностей наблюдений при отсутствии сигнала и $P(z \mid H_1)$ – при его наличии.

Из (3.16) следует, что решение $u = H_1$ принимается при выполнении неравенства $L_1 < L_0$, т. е. если

$$a_{10}P(z \mid H_0) + a_{11}P(z \mid H_1) < a_{00}P(z \mid H_0) + a_{01}P(z \mid H_1),$$

или в эквивалентном виде

$$(a_{01} - a_{11})P(z \mid H_1) > (a_{10} - a_{00})P(z \mid H_0),$$

$$(g_{01} - g_{11})p(H_1)P(z \mid H_1) > (g_{10} - g_{00})p(H_0)P(z \mid H_0).$$

Естественно, что $g_{01} > g_{11}$ и $g_{10} > g_{00}$ (потери при верном решении должны быть меньше, чем при ошибочном). Поэтому решающее правило принимает вид

$$\Lambda(z) = \frac{p(z \mid H_1)}{p(z \mid H_0)} \begin{cases} > \Lambda_0 \Rightarrow H_1, \\ \leq \Lambda_0 \Rightarrow H_0, \end{cases} \qquad (3.17)$$

где

$$\Lambda_0 = \frac{g_{10} - g_{00}}{g_{01} - g_{11}} \frac{p(H_0)}{p(H_1)} \quad . \qquad (3.18)$$

пороговое значение (порог) решающего правила.

Отношение $\Lambda(z) = p(z \mid H_1) / p(z \mid H_0)$ называется **отношением правдоподобия** (ОП). Оказывается, что ОП является **достаточной статистикой** для рассматриваемой задачи, т. е. вся информация, содержащаяся в наблюдениях z, сконцентрирована в единственном числе – значении ОП. Это значение нужно сравнить с порогом Λ_0, который зависит от априорной вероятности появления сигнала $p(H_1)$ (отметим, что $p(H_0) = 1 - p(H_1)$) и от функции потерь g_{ij}, т. е. от критерия оптимальности обнаружения. Если выбрать какой-то другой критерий оптимальности, то сменится только значение порога Λ_0, а правило обнаружения сохранит вид (3.17).

Отметим, что в общем случае (не обязательно для задачи обнаружения) правило (3.17) имеет вид

$$\Lambda(z) = \frac{p(z \mid \theta = 1)}{p(z \mid \theta = 0)} \begin{cases} > \Lambda_0 \Rightarrow u = 1, \\ \leq \Lambda_0 \Rightarrow u = 0. \end{cases} \qquad (3.19)$$

3.5. Оценка параметров. Методы построения оценок

Вторым важнейшим частным случаем статистических решений является оценка параметров.

Пусть искомое решение заключается в построении оценки $u = (u_1, \ldots, u_n)$ векторного параметра $\theta = (\theta_1, \ldots, \theta_n)$ по совокупности наблюдений z, где каждая из компонент имеет неограниченную область значений. Этот случай охватывает многочисленные задачи прогноза, фильтрации, оценки характеристик МИ и т. д.

Оптимальное решающее правило (оптимальная оценка) и в этом случае определяется соотношением (3.13). Однако при некоторых естественных предположениях можно получить правила в более простых формах.

Предположим, что функция потерь не зависит от z и симметрична относительно ошибки оценки: $g(u, \theta, z) = g(|\theta - u|)$. Например, при **квадратичной функции потерь** $g(t) = t^2$ получаем **метод наименьших квадратов** (МНК): в качестве оценки выбирается такое значение u, при котором среднее значение квадрата ошибки оценки $(\theta - u)^2$ минимально (если оценка несмещенная, то минимальна дисперсия ошибки). Если взять $g(t) = |t|$, то получим метод минимального среднего модуля ошибки и т. д. При разных функциях потерь, т. е. при разных понятиях оптимальности, могут получаться и разные оптимальные оценки. Например, при $g(t) = t^2$ оптимальной

оценкой является (условное) математическое ожидание параметра θ, а при $g(t) = |t|$ его (условная) медиана. Это разнообразие оптимальных (каждая в своем смысле) оценок нежелательно.

Предположим дополнительно, что апостериорная ПРВ $W(\theta \mid z)$ хотя бы приблизительно симметрична относительно некоторой точки $\hat{\theta}(z)$, зависящей от z, и что

$$\lim_{\theta \to \infty} g(\theta - \hat{\theta}(z)) W(\theta \mid z) = 0,$$

т. е. что функция потерь на бесконечности возрастает не слишком быстро. При этих условиях из (3.12) следует, что оптимальное решение u, т. е. оптимальная оценка неизвестного θ, определяется как

$$u = u_0(z) = \hat{\theta}(z) \tag{3.20}$$

независимо от конкретного вида функции потерь $g(t)$, функции правдоподобия $P(z \mid \theta)$ и априорного распределения $W(\theta)$.

Точка $\hat{\theta}(z)$ обладает тем свойством, что в ней достигается максимум апостериорной ПРВ $W(\theta \mid z)$, что и дает универсальный способ нахождения оптимальных оценок – **метод максимума апостериорной ПРВ** (МАП): оптимальной оценкой $\hat{\theta}(z)$ параметра θ при наблюдении z является точка максимума апостериорной ПРВ $W(\theta \mid z)$ по θ:

$$W(\hat{\theta}(z) \mid z) = \max_{\theta} W(\theta \mid z) \quad . \qquad (3.21)$$

Как и в п.3.4, при высокой информативности наблюдений априорные сведения должны слабо влиять на вид оптимального решения. При этом предположении можно получить другое решающее правило. Для этого представим (3.21) в эквивалентном виде

$$P(z \mid \hat{\theta}(z))W(\hat{\theta}(z)) = \max_{\theta} P(z \mid \theta)W(\theta) \quad . \qquad (3.22)$$

Если априорная информация (т. е. ПРВ $W(\theta)$) мало влияет на решение, то определяющей должна быть ФП $P(z \mid \theta)$, поэтому, заменяя (3.22) на приближенное уравнение

$$P(z \mid \hat{\theta}(z)) = \max_{\theta} P(z \mid \theta) \quad , \qquad (3.23)$$

получаем так называемый **метод максимального правдоподобия** (ММП): в качестве оценки $\hat{\theta} = \hat{\theta}(z)$ берется точка максимума ФП $P(z \mid \theta)$.

Замечание. ММП – все же приближенный метод, в нем априорная информация полностью игнорируется, поэтому иногда получаемые с помощью этого метода оценки существенно хуже оценок МАП.

3.6. Оценка гауссовских параметров по гауссовским наблюдениям

Во многих приложениях данные имеют гауссовские распределения, поэтому рассмотрим этот случай подробнее.

Совместная ПРВ n гауссовских СВ $y_1,\dots,y_n$, составляющих гауссовский вектор $\bar{y} = (y_1,\dots,y_n)^T$, имеет вид

$$w(y_1,\dots,y_n) = w(\bar{y}) = \frac{1}{(2\pi)^{n/2}\det^{1/2}(V)} \exp\left(-\frac{1}{2}(\bar{y}-\bar{m})^T V^{-1}(\bar{y}-\bar{m})\right),$$

(3.24)

где

$\bar{m} = M[\bar{y}] = (M[y_1], M[y_2],\dots,M[y_n])^T = (m_1,\dots,m_n)^T$

– вектор математических ожиданий (математическое ожидание вектора $\bar{y}$);

$V = M[(\bar{y}-\bar{m})(\bar{y}-\bar{m})^T] = \left(M[(\bar{y}_i-\bar{m}_i)(\bar{y}_j-\bar{m}_j)^T]\right)_{i,j=\overline{1,n}}$

– матрица ковариаций. Отметим, что V – симметричная матрица: VT=V. В частности, если $\bar{m} = 0$, т. е. $M[y_i] = 0$, то

$$w(\bar{y}) = \frac{1}{(2\pi)^{n/2}\det^{1/2}(V)} \exp\left(-\frac{1}{2}\bar{y}^T V^{-1}\bar{y}\right),$$

(3.25)

где $V = M[\bar{y}\bar{y}^T]$.

Оптимальная оценка

Рассмотрим задачу оценки гауссовского вектора $\bar{x} = (x_1,\dots,x_m)^T$, когда имеется гауссовский вектор наблюдений $\bar{z} = (z_1,\dots,z_n)^T$.

Пусть известны все средние значения и все ковариации СВ x_i, $i = \overline{1,m}$, и z_i, $i = \overline{1,n}$. Без потери общности можно считать, что $M[\bar{x}] = 0$ и $M[\bar{z}] = 0$, так как мы можем центрировать все СВ, вычитая из них

их математические ожидания. Будем искать оптимальные оценки в смысле минимума средних квадратов ошибок, т. е. при квадратичной функции потерь

$$M[(\hat{x}_i - x_i)^2] = \min \quad . \qquad (3.26)$$

По общей теории статистических решений (п.3.5), оптимальная оценка $\hat{\bar{x}}$ в рассматриваемом случае есть оценка по методу МАП:

$$w(\hat{\bar{x}}, \bar{z}) = \max_{\bar{x}} w(\bar{x}, \bar{z}) \quad . \qquad (3.27)$$

Представим $w(\hat{\bar{x}}, \bar{z})$ в виде (3.25). Для этого объединим $\bar{x}$ и $\bar{z}$ в один вектор $\bar{y} = \begin{pmatrix} \bar{x} \\ \bar{z} \end{pmatrix}$, тогда

$$V = \begin{pmatrix} V_{xx} & V_{xz} \\ V_{zx} & V_{zz} \end{pmatrix}, \qquad (3.28)$$

где $V_{xx} = M[\overline{\bar{x}\bar{x}}^T]$; $V_{zz} = M[\overline{\bar{z}\bar{z}}^T]$; $V_{xz} = M[\overline{\bar{x}\bar{z}}^T]$; $V_{zx} = M[\overline{\bar{z}\bar{x}}^T]$, $V_{xz}^T = V_{zx}$. Таким образом,

$$w(\bar{x}, \bar{z}) = \frac{1}{(2\pi)^{m+n/2} \det^{1/2}(V)} \exp\left(-\frac{1}{2} (\bar{x}^T, \bar{z}^T) \begin{pmatrix} V_{xx} & V_{xz} \\ V_{zx} & V_{zz} \end{pmatrix}^{-1} \begin{pmatrix} \bar{x} \\ \bar{z} \end{pmatrix} \right) \quad . \qquad (3.29)$$

Максимум (3.29) достигается при минимальном значении выражения

$$\begin{pmatrix} \bar{x}^T, \bar{z}^T \end{pmatrix} \begin{pmatrix} V_{xx} & V_{xz} \\ V_{zx} & V_{zz} \end{pmatrix}^{-1} \begin{pmatrix} \bar{x} \\ \bar{z} \end{pmatrix}. \qquad (3.30)$$

Пусть

$$\begin{pmatrix} V_{xx} & V_{xz} \\ V_{zx} & V_{zz} \end{pmatrix}^{-1} = \begin{pmatrix} K & L \\ M & N \end{pmatrix}, \qquad (3.31)$$

где K, L, M, N – матрицы размеров $m \times m, m \times n, n \times m, n \times n$, соответственно, и

M=LT. Тогда

$$\begin{pmatrix} \bar{x}^T, \bar{z}^T \end{pmatrix} \begin{pmatrix} K & L \\ M & N \end{pmatrix} \begin{pmatrix} \bar{x} \\ \bar{z} \end{pmatrix} = \bar{x}^T K \bar{x} + \bar{x}^T L \bar{z} + \bar{z}^T M \bar{x} + \bar{z}^T N \bar{z} = \bar{x}^T K \bar{x} + 2\bar{x}^T L \bar{z} + \bar{z}^T N \bar{z}. \qquad (3.32)$$

Для нахождения минимума (3.32) возьмем производную по $\bar{x}$ и приравняем ее к нулю: $2K\bar{x} + 2L\bar{z} = 0$, $K\bar{x} + L\bar{z} = 0$, т. е.

$$\hat{\bar{x}} = -K^{-1} L \bar{z}. \qquad (3.33)$$

Отсюда следует очень важный факт: **оптимальная оценка гауссовских параметров по гауссовским наблюдениям линейна (есть линейная функция наблюдений $\bar{z}$).**

Конкретизируем оценку (3.33). Для этого воспользуемся формулой Фробениуса обращения блочных матриц:

$$\begin{pmatrix} A & B \\ C & D \end{pmatrix}^{-1} = \begin{pmatrix} T^{-1} & -T^{-1}BD^{-1} \\ -D^{-1}CT^{-1} & D^{-1}+D^{-1}CT^{-1}BD^{-1} \end{pmatrix},$$

(3.34)

где A и D – квадратные матрицы и $T = A - BD^{-1}C$. Из (3.31) и (3.34) имеем

$T = V_{xx} - V_{xz}V_{zz}^{-1}V_{zx}$, $K = T^{-1}$, $L = -T^{-1}V_{xz}V_{zz}^{-1}$, $-K^{-1}L = TT^{-1}V_{xz}V_{zz}^{-1} = V_{xz}V_{zz}^{-1}$.

Таким образом, оптимальная оценка (3.33) принимает вид

$$\hat{\bar{x}} = V_{xz}V_{zz}^{-1}\bar{z}. \qquad (3.35)$$

Исследуем свойства оптимальной оценки (3.35). Найдем сначала ковариации ошибок этой оценки:

$$M[(\hat{\bar{x}} - \bar{x})(\hat{\bar{x}} - \bar{x})^T] = M[(V_{xz}V_{zz}^{-1}\bar{z} - \bar{x})(\bar{z}^T V_{zz}^{-1}V_{zx} - \bar{x}^T)] =$$

$$= M[V_{xz}V_{zz}^{-1}\bar{z}\bar{z}^T V_{zz}^{-1}V_{zx} - \bar{x}\bar{z}^T V_{zz}^{-1}V_{zx} - V_{xz}V_{zz}^{-1}\bar{z}\bar{x}^T + \bar{x}\bar{x}^T] =$$

$$= V_{xz}V_{zz}^{-1}M[\bar{z}\bar{z}^T]V_{zz}^{-1}V_{zx} - M[\bar{x}\bar{z}^T]V_{zz}^{-1}V_{zx} - V_{xz}V_{zz}^{-1}M[\bar{z}\bar{x}^T] - M[\bar{x}\bar{x}^T]$$

$$= V_{xx} - V_{xz}V_{zz}^{-1}V_{zx},$$

$$M[(\hat{\bar{x}} - \bar{x})(\hat{\bar{x}} - \bar{x})^T] = V_{xx} - V_{xz}V_{zz}^{-1}V_{zx} = T. \qquad (3.36)$$

Итак, матрица T в (3.34) есть матрица ковариаций ошибок оптимальной оценки (3.35). При этом дисперсии ошибок равны диагональным элементам матрицы T.

Найдем ковариации ошибок оценок и наблюдений:

$$M[(\hat{\bar{x}}-\bar{x})\bar{z}^T]=M[(V_{xz}V_{zz}^{-1}\bar{z}-\bar{x})\bar{z}^T]=V_{xz}V_{zz}^{-1}M[\bar{z}\bar{z}^T]-M[\bar{x}\bar{z}^T]=\mathbf{0}.$$

Это очень важное обстоятельство: **ошибки оптимальных оценок не коррелированы с наблюдениями:**

$$M[(\hat{\bar{x}}-\bar{x})\bar{z}^T]=\mathbf{0}. \qquad (3.37)$$

Отметим, что оценка (3.35) есть **оптимальная линейная оценка** $\hat{\bar{x}}$ **для любых центрированных векторов** $\bar{x}$ **и** $\bar{z}$, т. е. это оптимальная оценка среди оценок вида $F\bar{z}$, где F – любая матрица. Но эта оценка не обязательно оптимальна для негауссовских векторов.

Рассмотрим теперь важный случай оценки одного параметра, т. е. пусть $\bar{x}=x$ и $\hat{x}=\bar{\boldsymbol{\alpha}}^T\bar{z}$, где $\bar{\boldsymbol{\alpha}}$ – весовой вектор оценки. Тогда (3.37) принимает вид

$$M[(\bar{\boldsymbol{\alpha}}^T\bar{z}-x)\bar{z}^T]=\mathbf{0},$$

где

$$\bar{\boldsymbol{\alpha}}^T=V_{xz}V_{zz}^{-1}, \quad \bar{\boldsymbol{\alpha}}=V_{zz}^{-1}V_{zx}. \qquad (3.38)$$

Из (3.38) следует, что $\bar{\boldsymbol{\alpha}}$ является решением системы линейных уравнений

$$\boldsymbol{V}_{zz}\bar{\boldsymbol{\alpha}}=\boldsymbol{V}_{zx} \quad \text{или}$$

$$\begin{pmatrix} V_{z_1z_1} & V_{z_1z_2} & \dots & V_{z_1z_n} \\ \dots & \dots & \dots & \dots \\ V_{z_nz_1} & V_{z_nz_2} & \dots & V_{z_nz_n} \end{pmatrix} \begin{pmatrix} \alpha_1 \\ \dots \\ \alpha_n \end{pmatrix} = \begin{pmatrix} V_{z_1x} \\ \dots \\ V_{z_nx} \end{pmatrix}. \qquad (3.39)$$

При этом средний квадрат ошибки оценки определяется из (3.36):

$$M\left[(\hat{x}-x)^2\right] = V_{xx} - V_{xz}V_{zz}^{-1}V_{zx} = \sigma^2 - V_{xz}V_{zz}^{-1}V_{zx}. \qquad (3.40)$$

Этот средний квадрат ошибки можно представить в другом виде. Рассмотрим определитель полной ковариационной матрицы V (наблюдений и оцениваемых параметров):

$$|V| = \begin{vmatrix} V_{xx} & V_{xz} \\ V_{zx} & V_{zz} \end{vmatrix}. \qquad (3.41)$$

Умножим слева матрицы нижнего ряда на $V_{xz}V_{zz}^{-1}$ и вычтем полученные произведения из матриц верхнего ряда (определитель при этом не изменится):

$$|V| = \begin{vmatrix} V_{xx} - V_{xz}V_{zz}^{-1}V_{zx} & V_{xz} - V_{xz}V_{zz}^{-1}V_{zz} \\ V_{xz} & V_{zz} \end{vmatrix} = |V_{zz}|\left|V_{xx} - V_{xz}V_{zz}^{-1}V_{zx}\right|.$$

Из (3.40), (3.41) и последнего равенства следует, что

$$M\left[(\hat{x}-x)^2\right] = |V|/|V_{zz}|, \qquad (3.42)$$

т. е. **средний квадрат ошибки оценки равен отношению определителя полной ковариационной матрицы к определителю ковариационной матрицы наблюдений.**

Пример 1. Рассмотрим оценку гауссовского сигнала x с параметрами (0, s2) по зашумленному наблюдению z=x+q, где q – гауссовский шум с параметрами (0, s2q), независимый от x. В нашем случае, используя (3.35), получаем

$$V_{zz} = M\left[z^2\right] = M\left[(x+\theta)^2\right] = \sigma_x^2 + \sigma_\theta^2$$

$$V_{xz} = M\left[xz\right] = M\left[x(x+\theta)\right] = \sigma_x^2,$$

$$\alpha = V_{xz}V_{zz}^{-1} = \sigma_x^2\left(\sigma_x^2+\sigma_\theta^2\right)^{-1} = \frac{\sigma_x^2}{\sigma_x^2+\sigma_\theta^2}.$$

Итак, оптимальной оценкой является

$$\hat{x} = \frac{\sigma_x^2}{\sigma_x^2+\sigma_\theta^2} z. \qquad (3.43)$$

Дисперсию ошибки этой оценки получим из (3.36):

$$M\left[(\hat{x}-x)^2\right] = V_{xx} - V_{xz}V_{zz}^{-1}V_{zx} = \sigma_x^2 - \sigma_x^2\frac{1}{\sigma_x^2+\sigma_\theta^2}\sigma_x^2 = \sigma_x^2\left(1-\frac{\sigma_x^2}{\sigma_x^2+\sigma_\theta^2}\right) = \frac{\sigma_x^2\sigma_\theta^2}{\sigma_x^2+\sigma_\theta^2},$$

или, по-другому, из (3.42):

$$M\left[(\hat{x}-x)^2\right] = \frac{\begin{vmatrix}\sigma_x^2 & \sigma_x^2\\ \sigma_x^2 & \sigma_x^2+\sigma_\theta^2\end{vmatrix}}{\sigma_x^2+\sigma_\theta^2} = \frac{\sigma_x^2\sigma_\theta^2}{\sigma_x^2+\sigma_\theta^2}.$$

Геометрическая интерпретация оптимальной оценки.

Лемма об ортогональном проектировании

Равенству (3.37) можно придать геометрический смысл. Будем считать центрированные СВ **элементами (векторами) векторного пространства**, в котором введено **скалярное произведение** $(xy) = M[xy]$. **Векторы x и y ортогональны, если (xy)=0, т. е. если x и y не коррелированы.** Множество всех линейных комбинаций вида $\bar{\alpha}^T \bar{z}$ есть **линейное пространство Z, натянутое на пространство наблюдений (коротко – пространство наблюдений).** Оптимальную линейную оценку, таким образом, нужно искать в пространстве наблюдений. Оптимальной является оценка $\hat{x}$, при которой значение $M\left[(\hat{x}-x)^2\right]$ минимально, т. е.

$$M\left[(\hat{x}-x)(\hat{x}-x)\right] = (\hat{x}-x, \hat{x}-x) = \left\|\hat{x}-x\right\|^2$$

принимает минимальное значение. Следовательно, длина вектора $\hat{x}-x$ должна быть **минимальной**. Этот минимум достигается, если вектор $\hat{x}-x$ **ортогонален к пространству** Z, а, следовательно, и к каждому из векторов z1, z2, …, zn, что, собственно, и означает равенство (3.37). Отсюда получаем следующее утверждение.

Лемма об ортогональном проектировании. Оптимальная линейная оценка $\hat{x}$ параметра x по наблюдениям $\bar{z}$ есть **ортогональная проекция x на пространство наблюдений** Z. Она удовлетворяет равенству (3.37), а в общем случае оценки $\hat{\bar{x}} = F\bar{z}$ векторного параметра $\bar{x}$ – равенству

$$M\left[\left(F\bar{z} - \bar{x}\right)\bar{z}^T\right] = \mathbf{0}.$$

Оптимальная оценка как условное среднее

Рассмотрим снова задачу оценки параметра x по наблюдениям $\bar{z}$ при квадратичной функции потерь, не предполагая гауссовость x и $\bar{z}$. Тогда оптимальная оценка $\hat{x}$ находится из условия минимума

$$M\left[(\hat{x}-x)^2\middle|\bar{z}\right], \tag{3.44}$$

где используется условное среднее при заданных наблюдениях. При этом $\hat{x}=\hat{x}(\bar{z})$. Приравнивая производную от (3.44) по $\hat{x}$ к нулю, получаем

$$M\left[x-\hat{x}\middle|\bar{z}\right]=\mathbf{0},\quad M\left[x\middle|\bar{z}\right]-M\left[\hat{x}\middle|\bar{z}\right]=M\left[x\middle|\bar{z}\right]-\hat{x}=\mathbf{0}.$$

Для векторного параметра аналогично:

$$\hat{\bar{x}}=M\left[\bar{x}\middle|\bar{z}\right]. \tag{3.45}$$

Таким образом, **оптимальная оценка в смысле минимума среднего квадрата ошибки есть условное математическое ожидание оцениваемого параметра при заданных значениях наблюдений.** Оценка (3.27) есть частный случай оценки (3.45). Действительно,

$$w\left(\bar{x},\bar{z}\right)=w\left(\bar{x}\middle|\bar{z}\right)w\left(\bar{z}\right), \tag{3.46}$$

поэтому максимум $w\left(\bar{x},\bar{z}\right)$ по $\bar{\mathbf{x}}$ достигается в той же точке, что и максимум $w\left(\bar{x}\middle|\bar{z}\right)$. Следовательно, (3.45) можно заменить на

$$w\left(\hat{\bar{x}}\middle|\bar{z}\right)=\mathbf{max}\ w\left(\bar{x}\middle|\bar{z}\right). \tag{3.47}$$

Но у гауссовских распределений максимум ПРВ приходится на среднее значение, поэтому оптимальная оценка и есть условное среднее.

3.7. Априорная неопределенность и способы неполного статистического описания

Реализация байесова подхода в идеальном виде требует достаточно полного статистического описания наблюдений Z и скрытых параметров q, позволяющего однозначно определить распределения p(q) и P(z|q), которые требуются для нахождения ожидаемой величины потерь (апостериорного риска) при решении u.

В действительности столь полное описание имеется далеко не всегда. Чаще всего имеется некоторая априорная неопределенность, т. е. неполнота описания. Обычно относительно Z и q имеется какая-то информация, которая не позволяет считать поставленную задачу совсем бессмысленной, но в то же время не дает возможности воспользоваться байесовым подходом в идеальном виде.

Рассмотрим несколько возможных способов неполного статистического описания.

Неполное описание распределения скрытых параметров

Случай А1. Крайний случай, когда относительно q ничего не известно, кроме области Θ допустимых значений. В этом случае априорное распределение $p(\theta)$ вообще неизвестно – это может быть любая неотрицательная функция с единственным условием $\int p(\theta)d\theta = 1$. В таких условиях и приходится решать задачу синтеза решающего правила.

Случай А2. О распределении $p(\theta)$ ничего не известно, но компоненты векторного параметра q = (q1,…, qn) связаны функциональными ограничениями F1(q1,…, qn)=0, … , Fk(q1,…, qn)=0, следующими из особенностей решаемой задачи. Используя эти ограничения, компоненты qi можно привести к виду q1 = f1(a), … , qn

= fn(a), т. е. q = q(a), где a = (a1,…, am) – векторный параметр, размерность m которого меньше, чем размерность n параметра q. В результате для статистического описания q достаточно задать распределение $p(\theta)$ на пространстве меньшей размерности, что предпочтительнее.

Случай А3. Распределение параметров q неизвестно, но известны некоторые его статистические характеристики, например, математические ожидания, дисперсии, ковариации и т. п. Тогда о $p(\theta)$ известно не только, что $\int p(\theta)d\theta = 1$, но и что $\int f_k(\theta)p(\theta)d\theta = a_k$, k=1,2,…,K, где fk(q) – некоторые функции. Например, если известна ковариационная матрица R=(rij) параметров q, то (при нулевых средних) $\int \theta_i\theta_j p(\theta)d\theta = r_{ij}$. Подобные данные, естественно, сужают класс возможных распределений $p(\theta)$, т. е. уменьшают априорную неопределенность.

Случай А4. Заданы распределения вероятностей низшего порядка, например, маргинальные pi(qi), i=1,…,n, или условные pi(qi|qi-1), i=2,…,n.

Отметим, что описание становится полным, если в первом случае компоненты независимы (тогда p(q1,…, qn) = p1(q1) p2(q2)…..pn(qn)), а во втором – марковские (тогда p(q1,…, qn) = p1(q1) p2(q2|q1) p3(q3|q2)….. pn(qn|qn-1)).

Случай А5. Могут быть априорные сведения качественного характера, например, что компоненты q независимы и одинаково распределены (тогда p(q1,…, qn)=p0(q1).…p0(qn), где p0(*) – неизвестное распределение).

Случай А6. Известен тип распределения параметров q, например, что они гауссовские, тогда

$$p(\bar{\theta}) = p(\theta_1,\ldots,\theta_n) = \frac{1}{(2\pi)^{n/2}\det^{1/2} R}\exp\left(-\frac{1}{2}(\bar{\theta}-\bar{m})^T R^{-1}(\bar{\theta}-\bar{m})\right),$$

где средние значения $\bar{m}$ и ковариационная матрица R неизвестны.

Общей чертой всех рассмотренных примеров является то, что в условиях априорной неопределенности вместо единственного распределения $p(\theta)$ параметров q можно задать только класс P0 таких распределений.

Таким образом, исходным описанием параметров q в случае априорной неопределенности является задание класса P0 возможных распределений $p(\theta)$ параметров q.

Чем шире класс P0, тем больше априорная неопределенность. В чисто байесовском случае P0 состоит из единственного элемента $p(\theta)$. В другом крайнем случае (A1) P0 – класс всех возможных распределений на Θ. Все остальные случаи – промежуточные между этими двумя крайними.

Неполное описание наблюдений

Описание априорной неопределенности наблюдений Z аналогично описанию априорной неопределенности параметров q. А именно, имеется целый класс P_θ функций правдоподобия p(z|q).

В чисто байесовском случае P_θ состоит из единственного элемента p(z|q). В другом крайнем случае, когда ничего не известно (подобно случаю A1), P_θ состоит из всех неотрицательных функций p(z|q), удовлетворяющих условию $\int p(z \mid \theta)dz = 1$ при всех q.

Параметрическая априорная неопределенность

Параметрический способ является довольно общим и удобным для описания априорной неопределенности. Рассмотрим несколько примеров.

Пример С1. Пусть параметр q дискретен и может принимать значения q=ai, i=1,2,…,n, с вероятностями p(q=ai)=pi. Если распределение q неизвестно вообще, то в качестве неизвестных параметров, описывающих это распределение, можно взять сами вероятности с очевидными ограничениями pi³0 и Spi=1, т. е. имеется n-1 неизвестных независимых параметров. Если же известно, например, математическое ожидание m_θ, то добавляется еще одно ограничение $\sum a_i p_i = m_\theta$, а количество независимых параметров уменьшается еще на единицу.

Пример С2. Пусть q=(q1,…, qn) – последовательность, описывающая, например, случайный процесс, подлежащий фильтрации. При этом известно, что это авторегрессионный процесс, описываемый уравнением $\theta_k = \rho\theta_{k-1} + \sigma\xi_k$, где ξ_k – независимые стандартные гауссовские СВ, а ρ и σ неизвестны. Тогда

$$p(\theta_k \mid \theta_{k-1}) = p(\theta_k \mid \theta_{k-1}, \rho, \sigma) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{1}{2\sigma^2}(\theta_k - \rho\theta_{k-1})^2\right)$$

и

$$p(\theta \mid \rho, \sigma) = p(\theta_1, \ldots, \theta_n \mid \rho, \sigma) = \frac{1}{(2\pi)^{n/2}\sigma^n} \exp\left(-\frac{1}{2\sigma^2}(\theta_1^2 + \sum_{k=2}^{n}(\theta_k - \rho\theta_{k-1})^2)\right)$$

распределение, зависящее от двух неизвестных параметров ρ и σ. Аналогичным образом можно записать распределение q=(qij:

i=1,2,…,m, j=1,2,…,n), когда q – И размеров $m \times n$, заданное моделью Хабиби с неизвестными значениями ρ, r и σ.

В общем случае параметрическую неопределенность описания параметров q **можно представить в виде**

$$p(\theta) = p(\theta \mid \beta), \qquad (3.48)$$

где β – совокупность неизвестных параметров.

Пример С3. Пусть совокупность наблюдений z=(z1,…,zn) представляет последовательность независимых нормальных компонент с неизвестной дисперсией σ^2 и математическими ожиданиями $M[z_i] = m_i(\theta)$, известным образом зависящими от q. Тогда ФП будет

$$P(z \mid \theta) = P(z \mid \theta, \sigma^2) = \frac{1}{(2\pi\sigma^2)^{n/2}} \exp\left(-\frac{1}{2\sigma^2} \sum_{i=1}^{n} (z_i - m_i(\theta))^2 \right), \qquad (3.49)$$

которая содержит один неизвестный параметр σ.

В общем случае параметрическую неопределенность описания наблюдений можно представить в виде

$$P(z \mid \theta) = P(z \mid \theta, \alpha), \qquad (3.50)$$

где a – совокупность неизвестных параметров.

Пример С4. Рассмотрим частный случай примера С3. Пусть q может принимать два значения: q=0 и q=1;

$M[z_i \mid \theta = 0] = m_i(0) = 0$ и

$M[z_i \mid \boldsymbol{\theta} = 1] = m_i(1) = as_i, i = 1,\ldots, n$, где si – известные величины и a – неизвестный коэффициент.

Этот пример можно трактовать как задачу обнаружения сигнала известной формы s=(s1,…,sn), но неизвестной интенсивности a на фоне некоррелированного шума неизвестной интенсивности (дисперсии) $\boldsymbol{\sigma}^2$.

В этом случае (3.49) принимает вид

$$P(z \mid \theta = 0) = P(z \mid \theta = 0, \sigma^2) = \frac{1}{(2\pi\sigma^2)^{n/2}} \exp\left(-\frac{1}{2\sigma^2}\sum_{i=1}^{n} z_i^2\right),$$

$$P(z \mid \theta = 1) = P(z \mid \theta = 1, \sigma^2, a) = \frac{1}{(2\pi\sigma^2)^{n/2}} \exp\left(-\frac{1}{2\sigma^2}\sum_{i=1}^{n} (z_i - as_i)^2\right),$$

причем при q=0 ФП зависит от одного параметра $\boldsymbol{\sigma}^2$, а при q=1 – от двух параметров $\boldsymbol{\sigma}^2$ и a, но общий вид (3.50) сохраняется, если взять $\boldsymbol{\beta} = (\boldsymbol{\sigma}^2, a)$.

Априорная неопределенность задания функции потерь

Функция потерь g(u,q,z) также может в отдельных задачах иметь неполное описание. Это в особенности касается сферы бизнеса, политики и т. п., когда неясно, к каким последствиям может привести то или иное решение.

В рассматриваемых нами задачах обработки И обычно удается подобрать функцию потерь, соответствующюю понятию оптимальности решения задачи. Поэтому мы будем практически всегда считать, что функция потерь определена.

Отметим, тем не менее, что априорная неопределенность задания функции потерь может быть описана стандартным способом –

указанием класса G возможных функций потерь, в том числе, параметрического класса таких функций.

4. Синтез решающих правил условиях априорной неопределенности. Адаптивные алгоритмы

Априорная неопределенность задания модели данных вносит дополнительные трудности при синтезе алгоритмов обработки этих данных. Алгоритмы должны каким-то образом адаптироваться, т. е. приспосабливаться к конкретным обрабатываемым данным. Синтезу адаптивных алгоритмов вообще и адаптивных алгоритмов обработки информации , в частности, уделяется большое внимание исследователей. Разработан ряд общих подходов к решению данной проблемы, а также множество алгоритмов для решения конкретных задач.

4.1. Особенности задачи синтеза при априорной неопределенности

Основная особенность задачи синтеза в условиях априорной неопределенности заключается в невозможности **однозначного** определения распределения вероятностей параметров θ и наблюдений z, а, следовательно, и в невозможности однозначного определения величины среднего риска, который является критерием для выбора оптимального решающего правила.

Как уже отмечалось, при априорной неопределенности можно указать только класс P0 возможных распределений p(θ) параметра θ и класс P_θ возможных функций правдоподобия $P(z \mid \theta)$. Таким образом, возможно только указание класса $P=(P_0,P_\theta)$ возможных пар распределений p=(p(θ),P(z|θ)).

Тогда для каждого $p \in P$ и каждого правила u = u(z) имеется свой средний риск R(u(z)) = R(u(z),p), являющийся функционалом от u(z) и p. Каждой паре $p \in P$ будет соответствовать свое

решающее правило u0p(z) = u0(z,p). При этом неизвестно, какая именно пара p имеет место в действительной ситуации. Выбрать же можно только одно решающее правило.

Задачей синтеза в этих условиях является отыскание такого правила u=u(z), которое в соответствии с некоторым **установленным порядком** обладало бы **наибольшей предпочтительностью** при всех возможных $p \in P$.

Понятия «установленный порядок» и «наибольшая предпочтительность» должны вводиться дополнительно и будут раскрываться в дальнейшем**.**

4.2. Существенная и несущественная априорная неопределенность

Пример 1. Пусть в примере С4 п.3.7 a=1 известно, но σ^2 неизвестно; p(θ=0)=p(θ=1)=0,5; g(0,0)=g(1,1)=0 и g(0,1)=g(1,0)=1, т. е. за неверное решение дается единичный штраф. Тогда при конкретном значении σ^2 оптимальное решающее правило (основанное на ОП) имело бы вид

$$\Lambda(z) = \frac{P(z \mid \theta = 1, \sigma^2)}{P(z \mid \theta = 0, \sigma^2)} \begin{cases} \geq 1 \Rightarrow \theta = 1, \\ < 1 \Rightarrow \theta = 0. \end{cases} \qquad (4.1)$$

Подставляя в это правило найденные в примере С4 плотности, получаем $\exp\left(\frac{1}{2\sigma^2}\left(\sum z_i^2 - \sum (z_i - s_i)^2\right)\right) \gtrless 0$ или $\sum z_i^2 \gtrless \sum (z_i - s_i)^2$ (если «³», то θ=1; если «<», то θ=0).

Оказалось, что, независимо от значения σ^2, решающее правило одно и то же. Поэтому это правило следует признать оптимальным в условиях данного примера. Априорная неопределенность

(неизвестность σ^2) оказалась несущественной в смысле влияния на структуру решающего правила.

Таким образом, могут встречаться ситуации, когда имеющаяся априорная неопределенность никак не сказывается на решающем правиле – оно остается тем же самым, что и при полном описании. Такая априорная неопределенность называется **несущественной**.

Однако при этом может оказаться, что средний риск, характеризующий возможные потери при использовании этого правила, остается неопределенным. В рассмотренном примере 1 очевидно, что при σ^2 = 0 ошибочных решений не будет и средний риск равен нулю. Но этот риск будет возрастать с ростом σ^2 (покажите, что он стремится к 0.5 при σ^2 ® ¥).

Пример 2. Пусть теперь в примере 1 еще и S_i не известны. Тогда при известных σ^2 и S_i оптимальным решающим правилом было бы снова (как и в примере 1)

$$\sum z_i^2 \gtrless \sum (z_i - s_i)^2,$$

которое, хотя и не зависит от σ^2, но существенно зависит от формы сигнала $S = (s_1, \dots s_n)$. Мы фактически не знаем, что надо искать, поэтому и не можем найти, используя традиционный подход к синтезу решающего правила. Постараемся найти какое-нибудь подходящее правило обнаружения. Преобразуем имеющееся нереализуемое правило к виду

$$\sum z_i^2 \gtrless \sum (z_i - s_i)^2 = \sum z_i^2 - 2\sum s_i z_i + \sum s_i^2,$$ т. е. $$\sum s_i z_i \gtrless 0.5 \sum s_i^2.$$

Значение $0.5\sum s_i^2$, хотя и неизвестно, но от наблюдений не зависит, обозначим его через Λ_0. Получаем $\sum s_i z_i \gtrless \Lambda_0$, т. е. правило заключается в сравнении взвешенной суммы $\sum s_i z_i$ с порогом Λ_0. При этом весовыми коэффициентами наблюдений z_i должны быть неизвестные значения $(s_1, \dots s_n)$ сигнала. Где бы их взять? Если сигнал в наблюдениях присутствует, то с некоторым приближением $s_i \approx z_i$ (по крайней мере, при относительно небольшой дисперсии шума σ^2). Отсюда следует правило

$$\sum z_i^2 \gtrless \Lambda_0.$$

Осталось только подобрать подходящий порог Λ_0. Отметим, что задачу этого примера можно трактовать как задачу различения гипотез H_0 (все $M[z_i] = 0$) и H_1 (не все $M[z_i] = 0$).

Таким образом, априорная неопределенность может быть и **существенной**, т. е. влиять на структуру решающего правила.

Рассмотрим теперь общий случай. Пусть имеющаяся априорная неопределенность позволяет только задать класс P пар распределений $p = (p(\theta), P(z \mid \theta)$). Найдем апостериорный риск

$$R(u \mid \theta, p) = \int g(u, \theta, z) \frac{P(z \mid \theta) p(\theta)}{\int P(z \mid \theta) p(\theta) d\theta} d\theta \tag{4.2}$$

для каждой такой пары p. Минимум (3.2) по u есть оптимальное решение для фиксированного p.

Если этот минимум достигается для $u = u(z)$, одинакового при всех $p \in P$, то априорная неопределенность несущественна. Само правило с таким свойством называется **равномерно наилучшим** для заданного класса P.

Существование таких правил является редким случаем. Чаще встречаются приближенно равномерные наилучшие правила.

4.3. Подходы к определению понятия оптимальности в условиях априорной неопределенности

Как уже отмечалось, каждой паре распределений p из класса возможных пар P соответствует, вообще говоря, свое оптимальное решающее правило. При этом неизвестно, какая именно пара p имеет место в действительности в момент принятия решения. Это обстоятельство требует разработки дополнения понятия оптимальности применительно к случаю априорной неопределенности, т. е. установления некоторого порядка предпочтения в множестве решающих правил, учитывающего особенности класса P. Рассмотрим некоторые подходы к определению оптимальности решения в этих новых условиях.

Пусть $U(z)$ – множество всех решающих правил. Для каждого $p \in P$ оптимальное, т. е. байесово, решение $u_0(z) = u_0(z, p)$ определяется из условия минимума среднего риска:

$$\min_{u(z)} R(u(z), p) = R(u_0(z, p), p). \qquad (4.3)$$

Представим эту зависимость решения $u_0(z,p)$ от p в виде условного графика (рис. 4.1), где по оси абсцисс откладывается p, а по оси ординат – соответствующие $u_0(z,p)$. Множество всех $u_0(z,p)$, $p \in P$, составляет некоторое подмножество правил $U_0(z) \subseteq U(z)$, из которых и следует выбирать окончательное решающее правило.

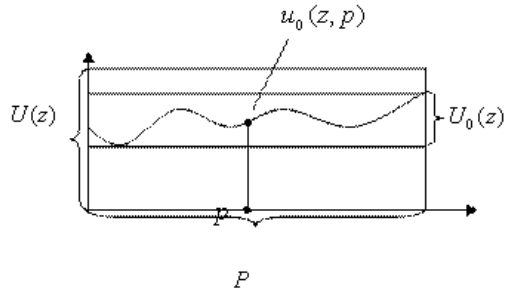


Рис. 4.1.

Равномерно наилучшее решение

Иногда (как в примере 1 из п.4.2) при всех $p \in P$ байесово решение $u_0(z,p)$ – одно и то же, т. е. $u_0(z)$ (рис. 4.2). Тогда это решение является равномерно наилучшим, оно абсолютно оптимально, априорная неопределенность несущественна. Решение $u_0(z)$ может быть найдено с помощью обычного байесова подхода при произвольном p.

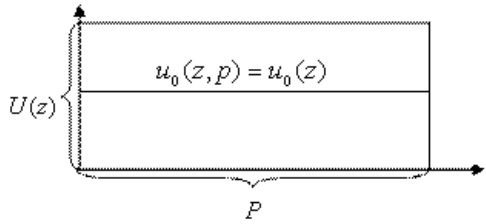


Рис. 4.2.

Отметим, что если ввести произвольную меру $\mu(p)$ на P (не обязательно вероятностную), проинтегрировать средний риск по этой мере:

$$\overline{R}(u(z)) = \int_P R(u(z), p) d\mu(p) \tag{4.4}$$

и найти правило $u^*(z)$, минимизирующее этот функционал, то при существовании равномерно наилучшего решения $u_0(z)$ окажется, что $u_0(z) = u^*(z)$. Это означает, что в рассматриваемом случае можно произвольным образом усреднять средний риск. Такое усреднение может существенно упростить поиск решения, так как усредненный риск (4.4) уже не зависит от p.

К сожалению, равномерно наилучшие правила в реальных задачах встречаются редко. Чаще может встретиться ситуация, когда разброс значений среднего риска $R(u(z), p)$ относительно невелик при всех $p \in P$ и $u(z) \subseteq U_0(z)$. Тогда можно получить приблизительно равномерно наилучшее решение, взяв $u_0(z)$, при

котором $R(u(z), p)$ принимает значения, по возможности наиболее близкие к среднему из значений $R(u(z), p)$.

Принцип минимума усредненного риска

Рассмотрим (3.4) при некоторой мере $\mu(p)$ на P и введем принцип предпочтения решений по **минимуму усредненного среднего риска**:

$$\overline{R}(u^*(z)) = \min_{u(z)} \overline{R}(u(z)) = \min_{u(z)} \int_P R(u(z), p)\, d\mu(p) \quad . \qquad (4.5)$$

Неоднозначность выбора решений $u^*(z)$ будет связана только с неоднозначностью выбора меры усреднения $\mu(p)$. Этой мере может быть дана различная трактовка.

Во-первых, $\mu(p)$ может рассматриваться как некоторое априорное распределение вероятностей на P. Например, можно взять равномерное распределение, если есть основания считать p равновозможными. Отметим, что при задании вероятностной меры $\mu(p)$ задача становится чисто байесовской, так как становится возможным определить распределение на z и θ:

$$p(z, \theta) = \int_P p(\theta) P(z \mid \theta) d\mu(p) \quad . \qquad (4.6)$$

Таким образом, в чистом виде этот случай соответствует полной определенности описания z и λ, поэтому неинтересен. Более интересен случай, когда мера $\mu(p)$ известна приближенно.

Во-вторых, $\mu(p)$ можно рассматривать как меру, характеризующую значимость последствий от принятия решений в условиях $p=(p(\theta),P(z|\theta))$.

Принципы минимакса (крайнего пессимизма), крайнего и умеренного оптимизма

Если не существует точного или приближенного равномерного наилучшего решения, то одним из возможных принципов предпочтения является **принцип минимакса:** выбирается правило $u(z)=u_M(z)$, при котором

$$\min_{u(z)} \max_{p\in P} R(u(z),p) = \max_{p\in P} R(u_M(z),p). \quad (4.7)$$

проиллюстрируем этот принцип рис. 4.3, на котором приведены графики зависимости среднего риска $R(u(z),p)$ от p для различных $u(z)\subseteq U(z)$.

Максимальные значения $R(u(z),p)$ по p для каждого $u_i(z)$ отмечены кружками, минимальные – квадратами, а средние арифметические максимума и минимума – звездочками. Принципу минимакса (4.7) на рис. 4.3 удовлетворяет решение $u_M(z)=u_4(z)$. Это правило при любой ситуации p обеспечивает средние потери, не превышающие R(u4(z),p1), где p1 – точка максимума функции R(u4(z),p) по p. Любое другое решение при некоторых p может привести к большим потерям, хотя и при других p – к меньшим.

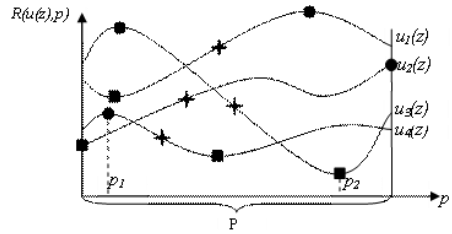


Рис. 4.3.

Таким образом, принцип минимакса обеспечивает минимальные средние потери в наихудшей из возможных ситуаций. Поэтому этот принцип называется **принципом крайнего пессимизма**, поскольку предполагается, что случится самая плохая из возможных ситуаций.

Возможен и другой крайний подход к определению понятия предпочтения решений – **принцип крайнего оптимизма**. А именно, будем предполагать, что случится наиболее благоприятная из возможных ситуаций. Тогда выбирается правило $u(z) = u_m(z)$, при котором

$$\min_{u(z)} \min_{p \in P} R(u(z), p) = \min_{p \in P} R(u_m(z), p) \quad (4.8)$$

На рис. 4.3 этому принципу соответствует правило um(z) = u3(z).

Между этими крайностями располагается **принцип умеренного оптимизма-пессимизма**, исходящий из предпосылки, что обычно случается что-то среднее между наилучшим и наихудшим вариантами. Правило u(z) = us(z) выбирается из условия

$$\min_{u(z)}[\alpha \min_{p \in P} R(u(z), p) + (1-\alpha) \max_{p \in P} R(u(z), p)] =$$
$$= \alpha \min_{p \in P} R(u_s(z), p) + (1-\alpha) \max_{p \in P} R(u_s(z), p), \quad (4.9)$$

где a – константа между 0 и 1. При a = 0 правило (4.9) превращается в (4.7), а при a = 1 – в (4.8). При a = 0.5 в (4.9) будут использоваться средние арифметические максимумов и минимумов средних потерь. На рис. 4.3 эти значения отмечены крестиками, и для этого примера будет выбрано us(z) = u4(z).

Принцип асимптотической оптимальности

В практических задачах с априорной неопределенностью часто имеется большой объем наблюдений, которые в случае полного описания могут быть даже избыточны для решения данной задачи. Но в условиях априорной неопределенности эти избыточные данные могут оказаться полезными.

Пример 3. Рассмотрим задачу двухальтернативного решения при u = 0 или u = 1; θ = 0 или θ = 1; P(θ=0) = P(θ=1) = 0.5; g(0,0) = g(1,1) = 0, g(0,1) = g(1,0) = 1; наблюдение z = (z1, z2,….zn, zn+1) есть совокупность независимых СВ с ПРВ

$$P(z \mid \theta = 1) = P_1(z_{n+1}) \prod_{i=1}^{n} P_0(z_i \mid \alpha),$$

$$P(z \mid \theta = 0) = \prod_{i=1}^{n+1} P_0(z_i \mid \alpha), \qquad (4.10)$$

где a – параметр ПРВ величин z1,…..,zn. Таким образом, P1(t) известна полностью, а P0(t/a) – с точностью до параметра a. Неизвестность a и составляет априорную неопределенность. Этот пример можно трактовать как задачу обнаружения сигнала (θ=1 и u=1), который может проявиться только в последнем наблюдении zn+1. Если сигнал есть, то zn+1 имеет ПРВ P1(zn+1), если же его нет, то ПРВ zn+1 (как и всех остальных наблюдений) есть P0(zi|a).

Если бы параметр a был известен (априорной неопределенности нет), то оптимальное решающее правило имело бы вид

$$\Lambda(z) = \frac{P(z \mid \theta = 1)}{P(z \mid \theta = 0)} = \frac{P_1(z_{n+1})}{P_0(z_{n+1} \mid \alpha)} \begin{cases} \geq 1 \Rightarrow u = 1, \\ < 1 \Rightarrow u = 0. \end{cases}$$

(4.11)

Это правило зависит только от последнего наблюдения zn+1, а все остальные данные z1,….., zn избыточны, их могло бы и не быть вообще.

Если же а неизвестно, то данные z1,….., zn можно попытаться использовать для уменьшения априорной неопределенности. Более того, возможно, что с ростом количества этих данных влияние априорной неопределенности будет уменьшаться (по крайней мере, не возрастать). Можно даже надеяться, что при увеличении количества и качества данных удастся получить столь же хорошее решение, как и при отсутствии априорной неопределенности.

Отсюда вытекает **принцип асимптотической оптимальности**: наиболее предпочтительным является такое правило u = u(z), для которого средний риск R(u(z),p) с увеличением объема данных z стремится к минимальному байесову риску $\min_{u(z)} R(u(z), p)$ для всех $p \in P$ равномерно.

Однако может оказаться, что этот принцип не определяет решения однозначно, так как может быть целый ряд асимптотически оптимальных решений.

4.4. Адаптивный байесов подход

Сущность этого подхода состоит в следующем. Пусть имеется существенная априорная неопределенность в описании параметров θ и/или наблюдений z. Эта неопределенность не позволяет применить обычный байесов формализм: найти для каждого правила u(z) величину среднего риска R(u(z)), величину апостериорного риска R(u,z) и определить положение минимума R(u,z) по u для каждого z, что и дает оптимальное байесово решение u = u0(z).

Отметим, что невозможность применения этого формализма связана именно с незнанием, а не с существованием: на самом деле в любых конкретных условиях существуют вполне определенные (хотя и неизвестные) истинные значения R(u(z))=Rист(u(z)) для всех u(z), а следовательно, существует и оптимальное решение u0(z), обращающее Rист(u(z)) в минимум.

Попытаемся применить этот байесов формализм в условиях априорной неопределенности, используя сведения, содержащиеся в наблюдениях z для оценки истинного значения апостериорного риска, чтобы получить его приближенное значение $\hat{R}(u,z)$. После нахождения оценки $\hat{R}(u,z)$ остается воспользоваться стандартным байесовым подходом, используя $\hat{R}(u,z)$ вместо Rист(u,z).

Таким образом, адаптивный байесов подход в своей основе ничем не отличается от неадаптивного: главным остается минимизация среднего (апостериорного) риска. Различие только в способе достижения этой цели. При наличии априорной неопределенности приходится модифицировать обычный байесов формализм, вводя в него дополнительные процедуры. При этом повышается роль наблюдений z – они уже не просто аргументы известной функции R(u,z), но еще используются для ее восстановления.

Процесс восстановления функции R(u,z) называется **адаптацией**, а полученные таким способом правила называются **адаптивными байесовыми решающими правилами.**

Итак, адаптивный байесов подход основан на замене точной меры ожидаемых потерь ее оценкой на основе имеющихся данных. В этом аспекте различные способы адаптации есть различные способы оценки меры потерь.

Продолжение примера 3. Правило (4.11) нельзя применить, так как неизвестно значение a. Используем наблюдения z1,….,zn для нахождения оценки $\alpha^* = \alpha^*(z_1,\ldots,z_n)$. Можно, в частности, взять оценку ММП a*, определяемую в нашем случае соотношением

$$\max_{\alpha} P(z \mid \alpha) = \max_{\alpha} \prod_{i=1}^{n} P_0(z_i \mid \alpha) = \prod_{i=1}^{n} P_0(z_i \mid \alpha^{**}) .$$

Подставляя это значение a* в (4.11), получим **адаптивное правило**

$$\frac{P_1(z_{n+1})}{P_0(z_{n+1} \mid \alpha^*)} \gtrless 1 . \qquad (4.12)$$

Если ошибка $|\alpha^* - \alpha|$ мала, то можно ожидать, что правило (4.12) будет для подавляющего большинства значений zn+1 давать то же решение, что и (4.11).

Для иллюстрации конкретизируем этот пример, взяв a>0 и нормальные распределения

$$P_0(z_i \mid \alpha) = \frac{1}{\sqrt{2\pi}\sigma} \exp(-(z_i - \alpha)^2 / 2\sigma^2)$$

и

$$P_1(z_{n+1}) = \frac{1}{\sqrt{2\pi}\sigma} \exp(-z_{n+1}^2 / 2\sigma^2)$$

Тогда (4.11) приводится к виду

$$u(z) = \begin{cases} 1, z_{n+1} \le \alpha / 2, \\ 0, z_{n+1} > \alpha / 2, \end{cases} \qquad (4.13)$$

а правило (4.12) – к виду

$$u^*(z) = \begin{cases} 1, z_{n+1} \le \alpha^* / 2, \\ 0, z_{n+1} > \alpha^* / 2. \end{cases} \qquad (4.14)$$

Оценкой ММП параметра a в нашем случае будет $\alpha^* = \sum z_i / n$. При этом ошибка e=a*-a нормальна с нулевым средним и дисперсией s2/n. Таким образом, вместо области $A = (-\infty; \alpha/2]$ принятия решения u=1 в правиле (4.13) имеем область $A^* = (-\infty; \alpha^*/2] = (-\infty; \alpha/2 + \varepsilon/2]$ в правиле (4.14). Эти области различаются на интервал (a/2;a/2+e/2] или (a/2+e/2; a/2]. При этом e/2 имеет СКО $\sigma/2\sqrt{\pi}$, стремящееся к нулю при $n \to \infty$, поэтому разница между правилами тоже сходит на нет.

Найдем средний риск для правил (4.13) и (4.14). Средний риск для (4.13):

$$R^0 = M[g(u(z),\theta)] = 0.5P(z_{n+1} \le \tfrac{\alpha}{2} \mid \theta = 0) + 0.5P(z_{n+1} > \tfrac{\alpha}{2} \mid \theta = 1) =$$
$$= 0.5\Phi((\alpha/2 - \alpha)/\sigma) + 0.5(1 - \Phi((\alpha/2 - 0)/\sigma)) = 1 - \Phi(\alpha/2\sigma).$$

Средний риск для (4.14):

$$R^* = M[g(u^*(z),\theta)] = 0.5\,P(z_{n+1} \le \sum_{i=1}^{n} z_i/2n \mid \theta = 0) + 0.5\,P(z_{n+1} > \sum_{i=1}^{n} z_i/2n \mid \theta = 1) =$$
$$= 0.5\,P(z_{n+1} - \sum_{i=1}^{n} z_i/2n \le 0 \mid \theta = 0) + 0.5\,P(z_{n+1} - \sum_{i=1}^{n} z_i/2n > 0 \mid \theta = 1).$$

Величина $z_{n+1} - \sum_{i-1}^{n} z_i/2n$ нормальна. При θ=0 она имеет среднее значение a/2 и дисперсию (1+1/4n)s2, а при θ=1 имеет среднее -a/2 и ту же дисперсию. Поэтому

$$R^* = 0.5\Phi((0 - \alpha/2)/\sigma\sqrt{1+1/4n}) + 0.5(1 - \Phi((0 + \alpha/2)/\sigma\sqrt{1+1/4n}) = 1 - \Phi(\alpha/2\sigma\sqrt{1+1/4n}).$$

Естественно, что R*>R0, но разница $R^* - R^0 = \Phi(\alpha/2\sigma) - \Phi(\alpha/2\sigma\sqrt{1+1/4n})$ стремится к нулю при $n \to \infty$ и мала уже при относительно небольших n. Таким

образом, адаптивное правило (4.14) **асимптотически оптимально**: при увеличении объема данных средний риск R*стремится к минимальному байесову риску R0 , который мы имели бы при отсутствии априорной неопределенности, т. е. при известном a.

Пример 4. Имеется И Z = {zij}. На нем, возможно, есть области, отличающиеся от своего окружения большей средней яркостью. Требуется обнаружить эти области. Такая задача возникает, в частности, при поиске акваторий, перспективных для рыбного промысла – таковыми являются участки моря с более высокой средней температурой, для такого поиска используются тепловизионные И моря.

Зададимся формой области W и ее окружением R, например, квадрат и рамка, показанные на рис. 4.4. Если mW и mR – математические ожидания отсчетов изображения по W и по R, то нам нужно выбрать одну из двух альтернатив $H_0(m_W \le m_R)$ или $H_1(m_W > m_R)$. При известных mW и mR решающее правило элементарно:

$$m_W - m_R \begin{cases} \le 0 \Rightarrow H_0, \\ > 0 \Rightarrow H_1. \end{cases} \qquad (4.15)$$

Это правило нужно применить ко всевозможным положениям W на Z. Однако mW и mR неизвестны. Построим адаптивное правило. Для этого применим в качестве оценок неизвестных mW и mR средние арифметические наблюдений:

$$\bar{m}_W = \frac{1}{n_W}\sum_W z_{ij} \quad , \qquad \bar{m}_R = \frac{1}{n_R}\sum_R z_{ij} \quad ,$$

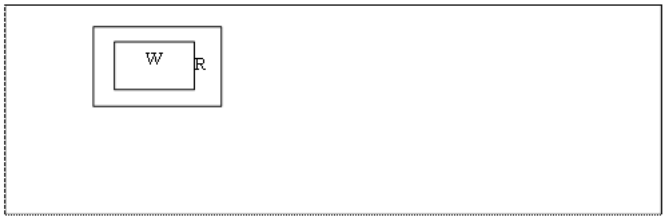


Рис. 4.4.

где nW и nR – количество элементов в W и R. Эти оценки состоятельны. Но теперь $\bar{m}_W > \bar{m}_R$ не обязательно влечет за собой $m_W > m_R$. Поэтому гипотезу H_1 следует принимать только при $m_W - m_R$ >m. Более того, порог m может быть различным для разных положений W, так как изображение Z может быть неоднородным.

Поэтому отнормируем разность $\bar{m}_W - \bar{m}_R$ оценкой СКО в рамке R, т. е. величиной $\bar{\sigma}_R = \sqrt{\frac{n_R}{n_R - 1}\sum_R (z_{ij} - \bar{m}_R)^2}$. В результате получим адаптивное правило

$$\frac{\bar{m}_W - \bar{m}_R}{\bar{\sigma}_R} \begin{cases} \le \Lambda_0 \Rightarrow H_0, \\ > \Lambda_0 \Rightarrow H_1. \end{cases} \quad (4.16)$$

Если все отсчеты изображения имеют ограниченные дисперсии и постоянные средние в W и R, то правило (4.16) сходится к правилу (4.15), когда nW и nR стремятся к бесконечности. При этом порог L0 стремится к нулю.

4.5. Классификация адаптивных алгоритмов

Как уже отмечалось, одним из способов решения задач обработки данных в условиях априорной неопределенности является применение адаптивных алгоритмов (решающих правил). В связи с этим отметим основные классы адаптивных алгоритмов.

Аргументные и критериальные задачи

По цели обработки данных адаптивные алгоритмы можно разделить на **аргументные и критериальные**. Исходной посылкой для синтеза алгоритмов является минимизация средних потерь, формально выражающихся функционалом качества $R(\overline{\alpha}, z) = J(\alpha)$, т. е. критерием, который нужно минимизировать по некоторым параметрам $\overline{\alpha}$. Однако требования к этой минимизации могут быть различными.

В **аргументных задачах** целью является возможно более точное отыскание точки минимума $\overline{\alpha^*}$ (возможно, переменной). К этому типу относятся задачи измерения параметров, фильтрации, прогноза и т. д. Сам критерий $J(\overline{\alpha})$ может вводиться искусственно и играет роль меры рассогласования между оценкой и точным значением параметра. При этом алгоритм обработки часто оказывается одинаковым для широкого класса функций потерь.

В **критериальных задачах** целью является приближение $J(\overline{\alpha})$ к его минимальному значению $J^* = J(\overline{\alpha^*})$, а сами параметры $\overline{\alpha}$ интереса не представляют и, вообще говоря, могут значительно отличаться от $\overline{\alpha^*}$.

Пусть, например, $\overline{\alpha}$ – весовой вектор линейной оценки $€= \overline{\alpha}^T \overline{Z}$ гауссовского параметра x по гауссовским наблюдениям $\overline{Z}$ и $J(\overline{\alpha})$

– средний квадрат ошибки этой оценки. Тогда поверхности $J(\overline{\alpha}) = c$ являются эллипсоидами с центром в $\overline{\alpha^*}$ (рис. 4.5).

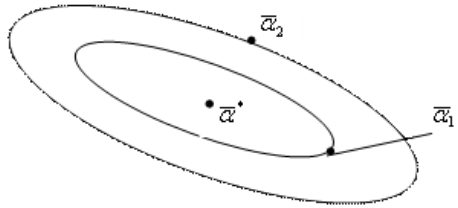


Рис. 4.5.

Может оказаться, что $J(\overline{\alpha_1}) < J(\overline{\alpha_2})$, хотя и $\overline{\alpha}_1$ находится от $\overline{\alpha^*}$ дальше, чем $\overline{\alpha}_2$. Другими словами, «потребительские» качества вектора $\overline{\alpha}_1$ выше, чем у $\overline{\alpha}_2$, несмотря на то, что $\overline{\alpha}_1$ находится дальше от оптимума, чем $\overline{\alpha}_2$.

Идентификационная и безыдентификационная адаптация

По методу нахождения оптимальных параметров $\overline{\alpha^*}$ адаптивные алгоритмы можно разбить на **идентификационные и безыдентификационные.**

В **идентификационных алгоритмах** сначала по всем имеющимся данным оцениваются все недостающие неизвестные характеристики γ. Затем полученные оценки € используются как точные. В результате получаем параметры для алгоритма в виде $\overline{\alpha} = \overline{\alpha}(€)$. В этом суть многочисленных **модифицированных байесовых решающих правил**. Таким способом было получено решающее правило в примере 3 из п. 4.4.

При всех своих положительных качествах идентификационные алгоритмы имеют следующие серьезные недостатки, особенно при обработке многомерных данных.

1) Зависимость данных от γ может быть очень сложной и даже неизвестной (например, зависимость И от межкадровых смещений), поэтому даже при известных γ определение $\bar{\alpha}(\gamma)$ представляет сложную задачу.

2) Получение оценок $\hat{\theta}$ требует дополнительных вычислений и делает обработку двухэтапной: сначала находятся оценки, а потом производится собственно обработка, что требует дополнительных линий задержки данных.

3) Дальнейшие вычисления, в которых используются оценки $\hat{\theta}$, могут быть неустойчивы к ошибкам этих оценок, например, матрицы выборочных корреляций зачастую плохо обусловлены (их детерминанты близки к нулю).

4) Даже точное значение $\bar{\alpha} = \bar{\alpha}(\gamma)$ может отличаться от оптимальных значений параметров алгоритма, так как обрабатываемые данные могут отличаться от используемой для их описания модели.

В алгоритмах **без идентификации** минимизация критерия $J(\bar{\alpha})$ производится по регулируемым параметрам $\bar{\alpha}$ без промежуточных оценок каких-либо характеристик γ исходных данных. При этом $\bar{\alpha}$ могут подбираться итерационно в процессе текущей обработки по наблюдениям за текущими значениями $J(\bar{\alpha})$. Для иллюстрации можно привести пример ручной настройки телевизора во время просмотра телепрограммы.

Для реализации таких алгоритмов необходима оценка текущего значения $J(\bar{\alpha})$, т. е. критерий должен быть наблюдаемым, что

является ограничением на область применения этого подхода. Иногда выход может быть найден с помощью замены $J(\bar{\alpha})$ на другой, наблюдаемый критерий $J_1(\bar{\alpha})$, от которого требуется только, чтобы точки минимума $J(\bar{\alpha})$ и $J_1(\bar{\alpha})$ совпадали в аргументных задачах, а в критериальных – чтобы $J(\bar{\alpha})$ приближалось к $J^* = J(\bar{\alpha}^*)$, когда $J_1(\bar{\alpha})$ приближается к $J_1^* = J_1(\bar{\alpha}_1^*)$. Эта методика не означает отхода от адаптивного байесова принципа, поскольку $J(\bar{\alpha})$ по-прежнему минимизируется.

Отметим, что между этими двумя классами алгоритмов есть много общего. В безыдентификационном алгоритме можно найти признаки идентификации. Действительно, поскольку имеется зависимость $\bar{\alpha} = \bar{\alpha}(\gamma)$, то по найденному $\bar{\alpha}$ можно иногда оценить и γ. Тем не менее, это существенно разные классы алгоритмов.

Квазиоптимальные алгоритмы

Аппроксимация решающего правила

Даже при полном описании данных не всегда удается найти оптимальное решающее правило из-за математических трудностей. Если его и удается найти, оно часто оказывается недопустимо трудоемким. Кроме того, используемая при синтезе модель исходных данных обычно лишь приближенно описывает реальность. В силу этих причин в реальных ситуациях часто не удается найти и применить оптимальное правило.

Поэтому приходится применять **квазиоптимальные**, реализуемые правила, по возможности с меньшим проигрышем в качестве обработки. Для поиска таких правил можно использовать упрощенные модели данных, описывающие лишь их принципиальные свойства. Полученные правила (алгоритмы) содержат некоторые неопределенные параметры $\bar{\alpha}$, которые необходимо выбрать так,

чтобы этот алгоритм давал наилучший результат на конкретных обрабатываемых данных.

Итак, к реальным данным адаптируется готовый алгоритм с неизменной структурой, изменяться могут только подстраиваемые параметры $\overline{\alpha}$. Такой подход к адаптации называется **аппроксимацией решающего правила**. Этот прием применяется, например, когда какая-то готовая аппаратура используется для обработки другого класса данных. Другой пример – поиск для решения данной задачи оптимального алгоритма в классе линейных алгоритмов.

4.6. Псевдоградиентные алгоритмы адаптации

Из п. 4.5 можно сделать вывод, что для обработки больших объемов данных (в частности, И и их последовательностей) целесообразно использовать безыдентификационные алгоритмы, учитывая требования простоты и работоспособности при значительных вариациях реальной ситуации. В значительной степени этим требованиям удовлетворяют **псевдоградиентные** (ПГ) адаптивные алгоритмы.

4.6.1. Структура и общие свойства

Пусть структура процедуры обработки определена, а критерий качества решения задачи сформулирован в терминах минимизации функционала $J(\overline{\alpha})$, который прямо или косвенно отражает средние потери, когда обработка выполняется с параметрами $\overline{\alpha}$. Ввиду априорной неопределенности описания данных нет возможности заранее определить оптимальные параметры $\overline{\alpha^*}$. Поэтому необходима некоторая процедура адаптации, составляющая вместе с процедурой обработки адаптивный алгоритм, в котором параметры $\overline{\alpha}$ определяются на основании конкретной реализации (наблюдения) Z объекта обработки.

Таким образом, **задача адаптации формулируется в виде задачи минимизации функции** $J(\overline{\alpha}) = J(\overline{\alpha}, Z)$ **для конкретной**

реализации Z, и речь идет об аппроксимации решающего правила, т. е. выбранной процедуры обработки. Применим для решения этой задачи безыдентификационную адаптацию.

Существует ряд численных методов поиска экстремумов. Наиболее распространенными являются различные модификации градиентного алгоритма

$$\overline{\alpha_n} = \overline{\alpha_{n-1}} - \mu_n \nabla J(\overline{\alpha_{n-1}}), \qquad (4.17)$$

где $\overline{\alpha_n}$ – следующее за $\overline{\alpha_{n-1}}$ приближение к точке минимума; μ_n – положительная числовая последовательность, определяющая длину шагов; $\nabla J(\overline{\alpha})$ – градиент функции $J(\overline{\alpha})$. Каждый шаг в (4.17) делается в направлении скорейшего убывания $J(\overline{\alpha})$. Хотя и при выполнении некоторых условий сходимость $\overline{\alpha_n} \to \overline{\alpha^*}$ имеет место, она может оказаться очень медленной. Для ее ускорения выбираются направления, отличные от антиградиента (методы Ньютона, сопряженных градиентов и т. д.).

Применению этих методов в обработке И препятствует необходимость многократных и громоздких вычислений $\nabla J(\overline{\alpha_{n-1}}, Z)$, каждое из которых обычно включает в себя всю процедуру обработки Z при параметрах $\overline{\alpha_{n-1}}$. Значительно сократить объем вычислений можно, если вместо $\nabla J(\overline{\alpha_{n-1}}, Z)$ взять усечение $\nabla Q(\overline{\alpha_{n-1}}) = \nabla J(\overline{\alpha_{n-1}}, Z_n)$, т. е. вычислять градиент не по всей реализации Z, а только по некоторой ее части Z_n, например, в скользящем окне на И. Но тогда в (3.17) вместо точного значения градиента будет использоваться его значение со случайной ошибкой $\overline{\delta_n}$, и получается алгоритм

$$\overline{\alpha_n} = \overline{\alpha_{n-1}} - \mu_n(\nabla J(\overline{\alpha_{n-1}}, Z) + \overline{\gamma_n}) = \overline{\alpha_{n-1}} - \mu_n \nabla Q(\overline{\alpha_{n-1}})$$

. (4.18)

Последовательность $\overline{\alpha_n}$ становится случайной, поэтому случаен и сам факт ее сходимости к $\overline{\alpha}^*$.

Случайные ошибки $\overline{\delta_n}$ в (4.18), вообще говоря, не являются серьезным препятствием для сходимости $\overline{\alpha_n} \to \overline{\alpha^*}$. Существует большой класс **методов стохастической аппроксимации**, основанных на том факте, что при центрированности ошибки ($M[\overline{\delta_n}] = 0$) процедура (4.18) сходится к $\overline{\alpha}^*$, как и процедура (4.17). Применяются и так называемые **методы случайного поиска**, в которых ошибка $\overline{\delta_n}$ вводится искусственно. Интересно, что наличие случайной ошибки $\overline{\delta_n}$ может даже ускорять сходимость процедуры.

Смысл центрированности $\overline{\delta_n}$ состоит в том, что шаги процедуры (5.2) **в среднем выполняются точно по антиградиенту**, так сказать «в среднем в правильном направлении». Оказывается, что это условие можно существенно ослабить. Рассмотрим пример, показанный на рис. 4.6, для которого $J(\overline{\alpha})$ есть расстояние между точкой $\overline{\alpha}$ и фиксированной точкой $\overline{\alpha}^*$. Тогда антиградиент $-\nabla J(\overline{\alpha})$ направлен по прямой от $\overline{\alpha}$ к $\overline{\alpha}^*$. Если вместо этого направления все время двигаться под углом, например, 89° к нему, то движущаяся точка $\overline{\alpha}$ по спирали будет стремиться к $\overline{\alpha}^*$. Таким образом, для сходимости (4.18) к $\overline{\alpha}^*$ совсем не обязательно, чтобы $M[\overline{\delta_n}] = 0$. Однако, если

в нашем примере взять угол 91°, то точка $\bar{\alpha}$ будет удаляться от $\bar{\alpha}^*$ по спирали.

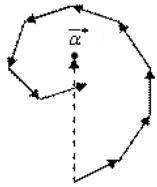


Рис. 4.6

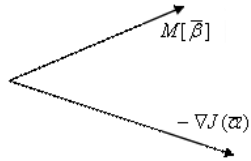


Рис. 4.7

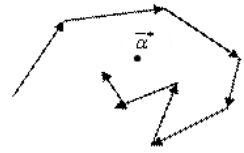


Рис. 4.8

В 1973 г. Я. З. Цыпкиным и Б. Т. Поляком было введено понятие **псевдоградиента** (ПГ), на основе которого разработан единый подход к анализу и синтезу алгоритмов стохастической минимизации функционалов. Класс ПГ алгоритмов очень широк и включает в себя все (или почти все) **алгоритмы адаптации и обучения**. Эти алгоритмы основаны на процедуре

$$\bar{\alpha}_n = \bar{\alpha}_{n-1} - \mu_n \bar{\beta}_n, \qquad (4.19)$$

где $\bar{\beta}_n$ – случайное (в частности, детерминированное) направление, вообще говоря, зависящее от предыдущих значений $\bar{\alpha}_i$ и от номера шага n. Направление $\bar{\beta}_n$ называется **псевдоградиентом** функционала $J(\bar{\alpha})$ в точке $\bar{\alpha}_{n-1}$, если выполнено **условие псевдоградиентности**

$$[J(\bar{\alpha}_{n-1})]^T M[\bar{\beta}_n] \geq 0, \qquad (4.20)$$

где левая часть есть скалярное произведение, поэтому **ПГ в среднем составляет острый угол с точным значением градиента** (рис. 4.7).

Алгоритм (4.19) называется **псевдоградиентным,** если $\overline{\beta}_n$ является ПГ на каждом шаге. В этом случае шаги в (4.19) будут производиться в среднем в сторону уменьшения $J(\overline{\alpha})$ и можно надеяться на сходимость $\overline{\alpha}_n \to \overline{\alpha}^*$ при $n \to \infty$ (рис. 4.8), хотя и некоторые шаги могут быть сделаны в сторону увеличения $J(\overline{\alpha})$. И действительно, выполнение относительно слабых условий оказывается достаточным для сходимости с вероятностью единица при любом начальном приближении $\overline{\alpha}_0$.

Главным из этих условий является строгое неравенство в (4.20) при $\overline{\alpha}_{n-1} \neq \overline{\alpha}^*$, единственность точки минимума, а также $\sum \mu_n = \infty$ (обеспечивает возможность дойти из любой точки до $\overline{\alpha}^*$) и $\sum \mu_n^2 < \infty$ (обеспечивает возможность асимптотического уменьшения дисперсии колебаний последовательности $\overline{\alpha}_n$). Последним двум условиям удовлетворяют, например, последовательности вида $\mu_n = 1/(a+bn)$.

Как уже отмечалось, не всегда целью является приближение $\overline{\alpha}_n$ к $\overline{\alpha}^*$. В критериальных задачах требуется, чтобы $J(\overline{\alpha}_n) \to J(\overline{\alpha}^*)$. Условия такой сходимости даже слабее условий аргументной сходимости.

Скорость сходимости для обоих классов задач имеет обычный для статистических алгоритмов порядок $O(1/\sqrt{n})$, хотя для критериальных задач она иногда выше, чем для аргументных.

Отметим, что алгоритм (4.19) является существенно более общим, чем (4.18), так как в (4.19) не предполагается возможность вычисления

$J(\bar{\alpha})$ или $\nabla J(\bar{\alpha})$, хотя бы и со случайной ошибкой, т. е. $J(\bar{\alpha})$ может быть и **ненаблюдаемым**. Необходимо только наличие **наблюдаемого** ПГ. В частности, в качестве $\bar{\beta}_n$ может быть выбрано (даже зашумленное) значение градиента другого функционала $J_1(\bar{\alpha})$, у которого та же точка минимума, что и у $J(\bar{\alpha})$.

Допустимость зависимости $\bar{\beta}_n$ от предыдущих значений $\bar{\alpha}_i$ дает возможность применения ПГ алгоритмов для обработки не только одномерных данных, но и многомерных в порядке некоторой их развертки. До сих пор предполагалось, что задачей является нахождение точки минимума $\bar{\alpha}^*$ функционала $J(\bar{\alpha}, Z)$, единой для всей реализации Z. Такая точка $\bar{\alpha}^*$ существует, но обработка будет оптимальной, если данные Z **однородны**. Для сходимости $\bar{\alpha}_n \to \bar{\alpha}^*$ при этом требуется сходимость $\mu_n \to 0$. Если же, начиная с некоторого момента, ограничить μ_n снизу (например, взять постоянные $\mu_n = \mu$), то дисперсии ошибок оценок $\bar{\alpha}_n$ параметров $\bar{\alpha}^*$ перестанут уменьшаться и будут иметь порядок μ^2, а сами $\bar{\alpha}_n$ будут колебаться около $\bar{\alpha}^*$.

Таким образом, если обработку однородных данных производить одновременно с оценкой $\bar{\alpha}^*$ (и при $\mu_n = \mu$), то по достижении установившегося режима будет осуществляться некоторая квазиоптимальная обработка.

Если произойдет скачкообразное изменение характеристик Z или переход к обработке других данных, то могут измениться и значения требуемых оптимальных параметров $\bar{\alpha}_n^*$. Если обработка будет

просто продолжена, то непосредственно после этого скачка возможно значительное ухудшение качества обработки, после чего постепенно снова будут достигнуты квазиоптимальные результаты.

При плавном изменении характеристик наблюдений Z (точнее, при плавном изменении оптимальных значений параметров $\overline{\alpha}^*$), соизмеримом со скоростью переходного процесса процедуры (4.19), появляется возможность применения ПГ алгоритмов к обработке неоднородных данных без их сегментации на участки относительно однородной структуры.

В такой постановке алгоритм (4.19) используется для текущей оценки переменных параметров $\overline{\alpha}_n^*$. Вопрос об асимптотической сходимости снимается. Вместо этого требуется как можно более точное приближение $\overline{\alpha}_n$ к изменяющемуся $\overline{\alpha}_n^*$ (или же $J(\overline{\alpha}_n)$ к $J(\overline{\alpha}_n^*)$). При этом возникает проблема компромисса в выборе μ_n: для уменьшения дисперсии ошибки нужно уменьшить μ_n, а для избежания запаздывания оценки следует увеличить μ_n.

Итак, ПГ алгоритмы просты в реализации, применимы к очень широкому классу однородных и неоднородных данных (причём в случае однородных данных сходятся к оптимальным алгоритмам). Адаптация может выполняться непосредственно в процессе обработки, поэтому не требуется линий задержки данных. Отмеченные положительные качества ПГ адаптивных алгоритмов делают их привлекательными для применения в обработке И, а также других больших массивов данных .

4.6.2. Выбор псевдоградиента

Главным в синтезе ПГ алгоритмов вида (4.19) является нахождение ПГ $\overline{\beta}_n$ функционала качества $J(\overline{\alpha}, Z)$. Рассмотрим два важнейших случая.

Случай 1. В большинстве статистических задач (в частности, в обработке И) функционал качества выражается через среднее значение некоторой функции $g(\bar{\alpha}, Z)$:

$$J(\bar{\alpha}, Z) = M[g(\bar{\alpha}, Z)], \tag{4.21}$$

например, через средний квадрат ошибки оценки параметра θ. Тогда

$$g(\bar{\alpha}, Z) = [f(\bar{\alpha}, Z) - \theta]^2 = \Delta^2(\bar{\alpha}, Z), \tag{4.22}$$

где θ – точное значение параметра и $f(\bar{\alpha}, Z) = \hat{\theta}$ – его оценка. К (4.22) приводят задачи прогноза, фильтрации и т. д.

Если реализации $g(\bar{\alpha}, Z)$ наблюдаемы (например, при прогнозе), то можно взять в качестве ПГ $\bar{\beta}_n = \nabla g(\bar{\alpha}_{n-1}, Z)$ или его сужение

$$\bar{\beta}_n = \nabla g(\bar{\alpha}_{n-1}, Z_n) \tag{4.23}$$

$$(\nabla \mathbf{M}[g])^{\mathrm{T}} \mathbf{M}[\bar{\beta}] = (\nabla \mathbf{M}[g])^{\mathrm{T}} \mathbf{M}[\nabla g] = (\mathbf{M}[\nabla g])^{\mathrm{T}} \mathbf{M}[\nabla g] = |\mathbf{M}[\nabla g]|^2 > 0.$$

на часть данных Z_n (например, на скользящее окно на И). Если возможно дифференцирование под знаком математического ожидания, то для направления (4.23) условие псевдоградиентности (4.20) выполняется тривиально:

Если же реализации функции (4.22) не наблюдаемы, то следует ввести вспомогательный наблюдаемый функционал качества J_1, выраженный через среднее значение некоторой функции.

Например, при оценке математического ожидания α случайной величины Z можно взять $J_1(\alpha, Z) = M[(Z-\alpha)^2]$, тогда

$$\overline{\beta}_n = -(z_n - \alpha_{n-1}) \quad \text{и} \quad \alpha_n = \alpha_{n-1} + \mu_n(z_n - \alpha_{n-1}), \tag{4.24}$$

где z_n – очередное наблюдение Z.

При оценке среднего квадрата случайной величины Z можно взять $J_1(\alpha, Z) = M[(Z^2-\alpha)^2]$, тогда

$$\overline{\beta}_n = -(z_n^2 - \alpha_{n-1}) \quad \text{и} \quad \alpha_n = \alpha_{n-1} + \mu_n(z_n^2 - \alpha_{n-1}). \tag{4.25}$$

При оценке коэффициента корреляции центрированных случайных величин Z и Y с одинаковыми дисперсиями можно взять $J_1(\alpha, Z, Y) = M[(\alpha Z - Y)^2]$, тогда

$$\overline{\beta}_n = (\alpha_{n-1} z_n - y_n) z_n \quad \text{и} \quad \alpha_n = \alpha_{n-1} - \mu_n(\alpha_{n-1} z_n - y_n) z_n. \tag{4.26}$$

Обобщением последней задачи является оптимизация линейной оценки $Y = \overline{\alpha}^T \overline{Z}$, например, оптимизация линейного прогноза. В этом случае $J_1(\overline{\alpha}, \overline{Z}, Y) = M[(\overline{\alpha}^T \overline{Z} - Y)^2]$,

$$\overline{\boldsymbol{\beta}}_n = (\overline{\boldsymbol{\alpha}}_{n-1}^T \overline{z}_n - y_n)\overline{z}_n \quad \text{и}$$
$$\overline{\boldsymbol{\alpha}}_n = \overline{\boldsymbol{\alpha}}_{n-1} - \boldsymbol{\mu}_n(\overline{\boldsymbol{\alpha}}_{n-1}^T \overline{z}_n - y_n)\overline{z}_n. \tag{4.27}$$

Оценка квантилей случайных величин также может быть выполнена с помощью ПГ алгоритмов. Этот способ рассмотрен в п. 4.4.

Случай 2. Иногда критерий качества выражается через вероятность события А, например, через вероятность правильного обнаружения. Этот случай можно свести к предыдущему, так как вероятность может быть выражена через математическое ожидание: $P(A) = M[\chi_A]$, где χ_A – индикатор события А ($\chi_A = 1$, если А произошло, и $\chi_A = 0$, если А не произошло). Тогда реализациями являются оценки вероятности Р(А) по частоте события А в каждом отдельном испытании (0 или 1). Для улучшения оценок можно использовать относительные частоты в группах из нескольких испытаний. При ненаблюдаемости события А иногда удается ввести вспомогательный функционал, выражаемый через параметры, влияющие на Р(А). Например, через отношение сигнал/шум в задачах обнаружения.

Отметим, что вместо ПГ (4.23) часто можно взять

$$\bar{\beta}_n = \varphi(\nabla g(\bar{\alpha}_{n-1}, Z_n)), \qquad (4.28)$$

где φ – векторная функция той же размерности, что и ∇g. При этом φ может выбираться из широкого класса функций. Требуется, чтобы условие псевдоградиентности сохранялось. В частности, можно брать симметричные функции. Очень простые и в то же время хорошо сходящиеся алгоритмы получаются при знаковой функции φ

$$\varphi(\bar{a}) = \varphi(a_1,\ldots,a_m) = sign(a_1,\ldots,a_m) = (sign(a_1),\ldots,sign(a_m))^T, \qquad (4.29)$$

при которой компоненты $\bar{\alpha}_n$ в (4.19) отличаются от компонент $\bar{\alpha}_{n-1}$ на $\pm\mu_n$.

4.7. Адаптивные псевдоградиентные алгоритмы фильтрации изображений

Методы ПГ адаптации могут быть с успехом применены и к задачам фильтрации И. Основная возникающая при этом трудность состоит в ненаблюдаемости качества фильтрации, поскольку ненаблюдаема ее ошибка. Поэтому качество фильтрации приходится оценивать с помощью вспомогательного наблюдаемого функционала, от которого требуется только, чтобы его точка минимума по параметрам процедуры фильтрации совпадала с точкой минимума основного функционала качества. Продемонстрируем эту методику на примере фильтрации плоского изображения.

Пусть наблюдаемое изображение $Z = \{z_{ij}\}$ представляет собой аддитивную смесь информативного сигнала $X = \{x_{ij}\}$, определяемого авторегрессионной моделью Хабиби

$$x_{ij} = \rho_{ij} x_{i,j-1} + r_{ij} x_{i-1,j} - \rho_{ij}\, r_{ij}\, x_{i-1,j-1} + \sqrt{(1-\rho_{ij}^2)(1-r_{ij}^2)}\,\sigma_x \xi_{ij}, \tag{4.30}$$

и белого гауссовского шума $\Theta = \{\theta_{ij}\}$:

$$z_{ij} = x_{ij} + \theta_{ij}. \tag{4.31}$$

При этом параметры модели сообщения (4.30) и дисперсия шума $\sigma^2_{\theta_{ij}}$ в модели наблюдения (4.31) неизвестны и, возможно, варьируются по полю кадра. В последнем случае вариация предполагается достаточно плавной.

Требуется по наблюдениям Z оценить информативное изображение X. Применим для решения этой задачи так называемый адаптивный псевдоградиентный аппроксимированный

фильтр Калмана, являющийся адаптивным вариантом аппроксимированного фильтра Калмана.

Рассмотрим сначала неадаптивный фильтр, когда параметры моделей (4.30) и (4.31) известны и постоянны. Оценки X находятся построчно.

Первая строка $x_1 = \{x_{1j} : j = \overline{1,N}\}$ оценивается по первой строке наблюдений $z_1 = \{z_{1,j} : j = \overline{1,N}\}$ с помощью уравнения фильтра Калмана в установившемся режиме:

$$\tilde{x}_{1j} = a\tilde{x}_{1,j-1} + b(z_{1j} - a\tilde{x}_{1,j-1}) = c\tilde{x}_{1,j-1} + bz_{1j},$$
$$c = a(1-b). \qquad (4.32)$$

Далее производится сглаживание обратным ходом:

$$\hat{x}_{1j} = \tilde{x}_{1j} + b(\hat{x}_{1,j+1} - a\tilde{x}_{1j}). \qquad (4.33)$$

Процедуры (4.32) и (4.33) отличаются от оптимальных постоянством коэффициентов, что приводит к ухудшению оценок в начале строки.

Пусть уже получена оценка $\hat{x}_{i-1}$ строки с номером i-1. Следующая i-я строка представляется в виде

$$x_i = (x_i - rx_{i-1}) + rx_{i-1} = y_i + rx_{i-1}, \qquad (4.34)$$

где $r = r_{ij}$ – (постоянный) параметр модели (4.31) – коэффициент корреляции между соседними строками. Оценка

$$\hat{x}_i = \hat{y}_i + r\hat{x}_{i-1} \qquad (4.35)$$

находится по наблюдениям

$$z_i' = z_i - r\hat{x}_{i-1}, \qquad (4.36)$$

полученным вычитанием прогноза $r\hat{x}_{i-1}$ строки x_i из наблюдений z_i этой строки. Сглаженные оценки $\hat{y}_i$ формируются с помощью процедур, аналогичных (4.32) и (4.33).

Описанный фильтр является приближенным, аппроксимированным вариантом векторного фильтра Калмана, когда изображение рассматривается как последовательность векторов (строк). Использование установившегося варианта приводит, помимо ухудшения оценок в начале строк, еще и к ухудшению оценок первых строк изображения.

Рассмотрим теперь адаптивный вариант описанного выше алгоритма. Этот алгоритм включает в себя процедуры (4.32)-(4.36) с переменными коэффициентами a, b, c, r и процедуру подстройки этих параметров непосредственно в процессе обработки.

Рассмотрим сначала процедуру (4.36). Она включает в себя формирование прогнозов $x_{ij}^{*} = r\hat{x}_{i-1,j}$ элементов x_{ij} по уже полученным сглаженным оценкам $\hat{x}_{i-1,j}$ предыдущей строки. Прогнозы эти должны быть оптимальными в смысле минимума дисперсии ошибок прогноза $s_{ij} = x_{ij} - x_{ij}^{*}$. Наблюдения $z_{ij} = x_{ij} + \theta_{ij}$ отличаются от x_{ij} некоррелированным с x_{ij} аддитивным шумом θ_{ij}, поэтому оптимальный прогноз $rx_{i-1,j}$ минимизирует не только дисперсию остатков s_{ij}, но и дисперсию остатков (4.36) прогноза наблюдений z_{ij}. Эти остатки наблюдаемы,

что позволяет применить адаптивные псевдоградиентные методы подстройки коэффициента прогноза r.

Построение адаптивного варианта процедуры (4.32) основано на том, что если вектор параметров $\bar{\alpha} = (a,b)^T$ оптимален в смысле минимума средних квадратов ошибок оценок x_{ij}, то он же оптимален и в смысле минимума средних квадратов ошибок прогнозов

$$\Delta_{ij} = z_{ij} - a\tilde{x}_{i,j-1}, \qquad (4.37)$$

и наоборот. Поэтому подстройка $\bar{\alpha}$ может быть осуществлена по наблюдаемым Δ_{ij}. Остатки в (4.37) зависят от параметра b через $\tilde{x}_{i,j-1}$:

$$\Delta_{ij+1} = z_{i,j+1} - a^2\tilde{x}_{i,j-1} - ab\Delta_{ij}. \qquad (4.38)$$

Для минимизации остатков (4.38) применяется алгоритм ПГ адаптации. Рассчитанные на очередном шаге коэффициенты $a_{i,j+1}$ и $b_{i,j+1}$ используются для вычисления очередного прогноза $\tilde{x}_{i,j+1}$ и уточнения значения $\Delta_{i,j+1}$.

Отметим, что в случае неоднородных изображений может быть скачок характеристик изображения при переходе к обработке очередной строки, так как конец предыдущей строки находится на большом расстоянии от начала очередной. Это обстоятельство требует резкого изменения параметров фильтра, что в алгоритме не предусмотрено. Сгладить это явление позволяет треугольная развертка изображения – смена направления обработки на противоположное при переходе к очередной строке. При такой развертке очередная обрабатываемая точка всегда находится рядом с предыдущей, поэтому

резких скачков характеристик изображения (при сделанном предположении о их плавном изменении) не происходит.

На рис. 4.9 приведен пример применения описанного алгоритма. В верхней части рисунка находится неискаженное И девочки.

Рис. 4.9.

Это И искажалось аддитивным белым шумом, в правой нижней части рисунка показана часть этого искаженного И. Часть отфильтрованного И показана в левой нижней части рисунка. Заметно значительное улучшение изображения после фильтрации. Однако можно заметить, что отфильтрованное изображение несколько хуже оригинала – оно как бы расфокусированное. В этом проявление неизбежных ошибок фильтрации. И многочисленные усилия исследователей направлены на улучшение качества фильтрации.

4.8. Коды Хаффмана

Работа алгоритма начинается с составления списка символов (чисел) алфавита в порядке убывания их частоты (вероятности). Лучше всего продемонстрировать этот алгоритм на простом примере. Пусть имеется пять символов: $a_1, a_2, \ldots, a_5$ с известными вероятностями: $p_1 = 0,4$, $p_2 = 0,2$, $p_3 = 0,2$, $p_4 = 0,1$ и $p_5 = 0,1$. Для построения кодов, выбираем сначала пару символов с наименьшими вероятностями – это символы a_4 и a_5. Наименее вероятному символу ставим в соответствие битовый ноль, а более вероятному – битовую единицу:

$$a_4 \to 1$$
$$a_5 \to 0$$

Символы a_4 и a_5 условно объединяются в единый символ a_{45} с вероятностью появления 0,2. Затем берется третий символ из упорядоченного списка – это символ a_3 с вероятностью 0,2. Поэтому код символа a_3 будет начинаться с битовой 1, а к кодам символов a_4 и a_5 дописывается битовый ноль:

$$a_3 \to 1$$
$$a_4 \to 10$$
$$a_5 \to 00$$

Далее, условно объединяем все три символа, в символ a_{345} с вероятностью появления 0,4 и рассматриваем его со следующим символом списка - a_2, вероятность которого 0,2. Так как вероятность появления символа a_2 меньше вероятности появления условного

символа a_{345}, то код символа a_2 будет начинаться с нуля, а кодам символов a_3, a_4 и a_5 приписывается битовая единица:

$$a_2 \to 0$$
$$a_3 \to 11$$
$$a_4 \to 101$$
$$a_5 \to 001$$

Аналогично, получаем для последнего символа a_1:

$$a_1 \to 0$$
$$a_2 \to 01$$
$$a_3 \to 111$$
$$a_4 \to 1011$$
$$a_5 \to 0011$$

В итоге получаем коды Хаффмана, но записанные в обратном порядке, т.е. для кодирования символа a_1 имеем код 0, для a_2 - код 10, для a_3 - 111, a_4 - 1101 и a_5 - 1100. Заметим, что данные коды могут быть корректно декодированы, например, последовательность символов a_1, a_2, a_3, a_4, a_5 будет представлена последовательностью бит

01011111011100

В этой последовательности первым встречается битовый ноль, но с нуля начинается только один код – код символа a_1, поэтому он так и декодируется. Затем видим код, начинающийся с битовой единицы. Таких кодов несколько, поэтому необходимо прочитать следующий бит, который равен 0. Код 10 – единственный, соответствующий символу a_2. После этого видим код с тремя единицами подряд (111).

Это говорит о том, что это символ a_3. Наконец последние два кода точно декодируют символы a_4 и a_5.

Построенные таким образом однозначные коды будут в среднем тратить 2,2 бита на символ:

$$0,4 \cdot 1 + 0,2 \cdot 2 + 0,2 \cdot 3 + 0,1 \cdot 4 + 0,1 \cdot 4 = 2,2 \text{ бит/символ}$$

Для сравнения, обычный двоичный код потребовал бы 3 бита для представления одного символа, т.е. в этом случае последовательность можно было бы сжать в 3/2,2=1,36 раза.

Однако чтобы произвести декодирование данных необходимо знать построенные коды. Самый лучший способ передать декодеру частоты появления символов в потоке, т.е. в данном случае дополнительно пять чисел. На основе этих частот декодер строит коды, а затем применяет их для декодирования последовательности. Поэтому, чем длиннее кодированная последовательность, тем меньше удельный вес служебной информации переданной декодеру.

И здесь возникает вопрос: а как можно численно определить степень статистической избыточности в заданной цифровой последовательности? Ведь тогда, зная эту характеристику мы могли бы определять максимально возможную степень сжатия этой последовательности и сравнивать конкретный алгоритм с этой нижней границей, т.е. мы могли бы объективно оценивать качество работы алгоритма сжатия. Оказывается, что такой величиной является энтропия, которая определяет минимальное число бит, необходимое для представления заданной последовательности чисел с последующей возможностью полного восстановления информации.

В 1948 г. сотрудник лаборатории Bell Labs Клод Шеннон показал, что минимальное число бит, которое необходимо затратить для представления одного символа той или иной информации можно найти с помощью формулы

$$H = -\sum_{i=1}^{N} p_i \log_2 p_i ,$$

где p_i - частота (вероятность) появления i-го числа в последовательности; N - число уникальных чисел в последовательности. Например, применяя данную формулу к нашей последовательности чисел, определяем, что число уникальных символов равно 5 – это a_1, a_2, a_3, a_4 и a_5. Частота появления цифр равна $p_1 = 0,4$, $p_2 = 0,2$, $p_3 = 0,2$, $p_4 = 0,1$ и $p_5 = 0,1$. В результате, минимальное число бит для представления одного символа в такой последовательности равно

$$H = -0,4\log_2 0,4 - 2 \cdot 0,2\log_2 0,2 - 2 \cdot 0,1\log_2 0,1 \approx 2,1$$

бит/символ.

Сравнивая полученный результат с ранее полученным для кодов Хаффмана (2,2 бит/символ), видим, что коды Хаффмана оказываются более близкими к нижней границе, чем равномерный код, которому необходимо 3 бита/символ.

Также можно посчитать и степень сжатия последовательности. Если равномерный код будет расходовать 3 бита на символ, то потенциальное сжатие будет равно

$$k = 3 / 2,1 = 1.4 \text{ раз,}$$

а с помощью неравномерных кодов Хаффмана получим

$$k = 3 / 2,2 = 1,36 \text{ раз.}$$

Следовательно, представленный алгоритм сжатия не является лучшим. Кроме того, с помощью энтропии можно показать, что если последовательность содержит случайный набор чисел (не имеет закономерностей), то вероятности $p_i \to 1 / N$, а возможность сжатия

$k \to 1$, т.е. случайный набор данных сжать невозможно. Это положение в частности говорит о том, что если один раз к данным был применен хороший алгоритм сжатия, например, zip, то повторное сжатие этим или другим алгоритмом не приведет к лучшим результатам, скорее наоборот. Таким образом, на выходе любого хорошего алгоритма сжатия будет получаться почти случайный набор данных.

4.9. Адаптивные коды Хаффмана

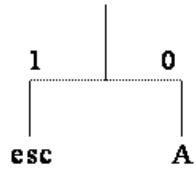
Работа рассмотренного выше алгоритма предполагает, что нам известны частоты появления символов. Для этого обычно просматривают кодируемую последовательность и подсчитывают число появлений того или иного символа, затем строят коды Хаффмана, а потом кодируют заданную последовательность. Такой подход имеет два недостатка: во-первых, приходится дважды просматривать кодируемую последовательность, что приводит к снижению скорости работы алгоритма, и, во-вторых, найденные частоты необходимо передавать декодеру для корректного декодирования данных. Поэтому на практике применяют другой метод формирования кодов переменной длины, который позволяет «на лету», по мере кодирования данных уточнять коды Хаффмана. В этом случае нет необходимости дважды просматривать последовательность и передавать коды приемной стороне, но за это приходится платить несколько худшим качеством сжатия. Такой алгоритм, например, лежит в основе программы compact операционной системы UNIX.

Основная идея адаптивного кодирования заключается в том, что компрессор и декомпрессор начинают работать с «пустого» дерева Хаффмана, а потом модифицируют его по мере чтения и обработки символов. Соответственно, и кодер и декодер должны модифицировать дерево одинаково, чтобы все время использовать один и тот же код, который может меняться по ходу процесса. Итак, в начале кодер строит пустое дерево Хаффмана, т.е. никакому символу коды еще не присвоены. Поэтому первый символ просто записывается в выходной поток в незакодированной форме, что обычно соответствует 8 битному коду ASCII. Затем, этот символ помещается в дерево и ему присваивается код, например, 0. После этого кодируется следующий символ во входном потоке, и если этот символ встретился впервые, то он также записывается в выходной поток в виде 8 битного ASCII символа, и помещается в дерево Хаффмана, где ему присваивается

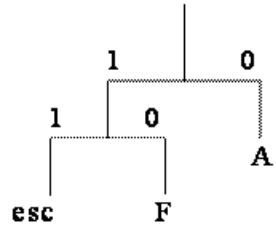
определенный код в соответствии с его текущей частотой появления, равной 1. По мере того как поступают символы на вход кодера, происходит подсчет числа их появления и их количества и в соответствии с этой информацией выполняется перестройка дерева Хаффмана. Но здесь есть один нюанс: как отличить 8 битный ASCII символ от кода переменной длины в момент декодирования последовательности? Чтобы разрешить эту коллизию используют специальный esc (escape) символ, который показывает, что за ним следует незакодированный символ. Соответственно код самого esc символа должен находиться в дереве Хаффмана и будет меняться каждый раз по мере кодирования информации (перестройки дерева).

Рассмотрим пример реализации адаптивных кодов Хаффмана на последовательности символов AFFAADB. Так как вначале кодирования дерево Хаффмана пустое, то в выходной поток записывается 8-битовый код символа A (01000001), который затем добавляется в дерево Хаффмана (рис. 10, а). Следующий символ входного потока F также отсутствует в дереве Хаффмана, поэтому он записывается в выходной поток как код ASCII-символа, но перед ним ставится еще и esc-символ (рис. 10, б). Следующий, третий символ, снова F и для него имеется код в дереве Хаффмана, поэтому этот код записывается в выходной поток, счетчик для символа F увеличивается на единицу и становится равный двум и дерево Хаффмана принимает вид (рис. 10, в). Четвертый символ входного потока A также имеется в дереве Хаффмана, поэтому его код заносится в выходной поток, а счетчик принимает значение 2 (рис. 10, г). Аналогичные действия выполняются и при считывании следующего символа A (рис. 10, д).

Выходной поток: 01000001
Счетчик для A: 1

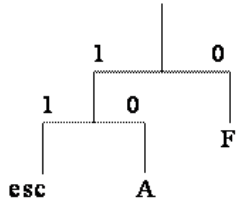


Выходной поток: 01000001 1 01000110
Счетчик для A: 1
Счетчик для F: 1



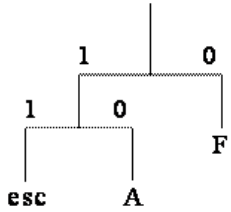
а) б)

Выходной поток: 01000001 1 01000110 10
Счетчик для A: 1
Счетчик для F: 2



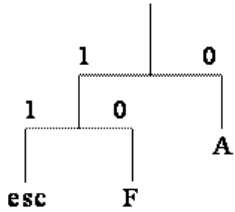
в)

Выходной поток: 01000001 1 01000110 10 10
Счетчик для A: 2
Счетчик для F: 2



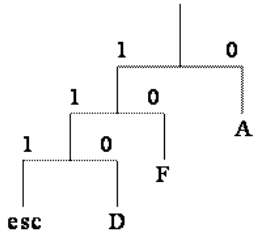
г)

Выходной поток: 01000001 1 01000110 10 10 10
Счетчик для A: 3
Счетчик для F: 2



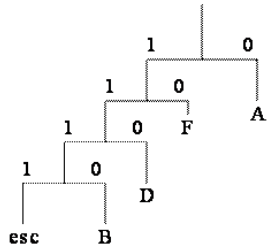
д)

Выходной поток: 01000001 1 01000110 10 10 10 11 01000100
Счетчик для A: 3
Счетчик для F: 2
Счетчик для D: 1



е)

Выходной поток: 01000001 1 01000110 10 10 10 11 01000100 111 01000010
Счетчик для A: 3
Счетчик для F: 2
Счетчик для D: 1
Счетчик для B: 1



ж)

Рис. 10

Следующий символ D является новым, поэтому он должен быть записан в выходной поток как ASCII-символ, перед которым следует поставить esc-символ (рис. 10, е). Здесь в дереве Хаффмана появляется третий уровень, т.к. сумма счетчиков для esc-символа и D равна 1 (счетчик esc всегда 0) и это меньше суммы значений счетчиков F и A. Наконец, последний символ B записывается в незакодированном виде в выходной поток и добавляется в дерево Хаффмана (рис. 10, ж).

В результате проведенного кодирования выходной поток составил 44 бит вместо 56, что соответствует сжатию в 1,27 раза. При кодировании более длинных последовательностей это значение будет увеличиваться, т.к. дерево Хаффмана будет содержать все больше и больше символов, которые будут представлены в выходном потоке как коды переменной длины, а не парами esc и ASCII символов.

Декодер при восстановлении исходной последовательности работает подобно кодеру, т.е. он также последовательно строит дерево Хаффмана по мере декодирования последовательности, что позволяет корректно определять исходные символы.

Адаптивное кодирование Хаффмана имеет ряд особенностей при своей работе. Первая связана с хранением значений счетчиков символов, при кодировании длинных последовательностей. Дело в том, что языки

высокого уровня, обычно, оперируют с числами максимум в 32 бит, в которых можно записывать числа от 0 до 4294967296, т.е. можно работать с файлами размером 4Гб. Но иногда этих значений недостаточно и приходится вырабатывать дополнительные меры по предотвращению переполнения счетчиков символов. Вторая особенность связана с постоянным перестроением дерева Хаффмана, что влияет на скорость работы алгоритма. Вместе с тем можно заметить, что при считывании очередного символа из входного потока, дерево Хаффмана может остаться прежним и его перестраивать не имеет смысла. Это бывает, когда новое значение счетчика символа не влияет на его местоположении в дереве Хаффмана. Следовательно, при кодировании и декодировании необходимо определять: нужно или нет пересчитывать дерево Хаффмана и в зависимости от результата выполнять нужные действия. Вот две основные особенности работы адаптивного алгоритма кодирования Хаффмана, но, несмотря на указанные сложности, он часто применяется на практике, например, в известном протоколе V.32 передачи данных по модему со скоростью 14400 бод.

4.10. Арифметическое кодирование

Метод Хаффмана является простым, но эффективным только в том случае, когда вероятности появления символов равны числам $1/2^n$, где n - любое целое положительное число. Это связано с тем, что код Хаффмана присваивает каждому символу алфавита код с целым числом бит. Вместе с тем в теории информации известно, что, например, при вероятности появления символа равной 0,4, ему в идеале следует поставить код длиной $-\log_2 0,4 \approx 1,32$ бит. Понятно, что при построении кодов Хаффмана нельзя задать длину кода в 1,32 бита, а только лишь в 1 или 2 бита, что приведет в результате к ухудшению сжатия данных. Арифметическое кодирование решает эту проблему путем присвоения кода всему, обычно, большому передаваемому файлу вместо кодирования отдельных символов.

Идею арифметического кодирования лучше всего рассмотреть на простом примере. Предположим, что необходимо закодировать три символа входного потока, для определенности – это строка SWISS_MISS с заданными частотами появления символов: S – 0,5, W – 0,1, I – 0,2, M – 0,1 и _ - 0,1. В арифметическом кодере каждый символ представляется интервалом в диапазоне чисел [0, 1) в соответствии с

частотой его появления. В данном примере, для символов нашего алфавита получим следующие наборы интервалов:

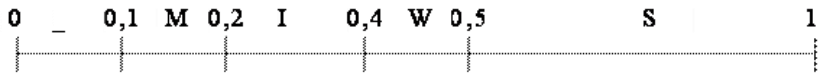


Рис. 11. Распределение интервалов представление символов

Процесс кодирования начинается со считывания первого символа входного потока и присвоения ему интервала из начального диапазона [0, 1). В данном случае для первого символа S получаем диапазон [0,5, 1). Затем, считывается второй символ – W, которому соответствует диапазон [0,4, 0,5). Но исходный диапазон [0, 1) уже сократился до [0,5, 1), поэтому символ W необходимо представить в этом новом диапазоне. Для этого достаточно вычислить новые нижнюю и верхнюю границы. Значение 0,4 будет соответствовать значению 0,7, а значение 0,5 – значению 0,75 (рис. 12).

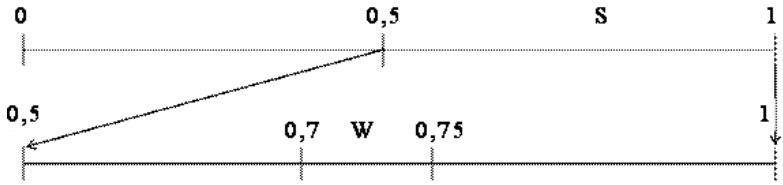


Рис. 12. Схема представления новых границ символа W

Данные границы можно вычислить по формулам:

NewHigh = OldLow + (OldHigh-OldLow)*HighRange(X),

NewLow = OldLow + (OldHigh-OldLow)*LowRange(X),

где OldLow – нижняя граница интервала, в котором представляется текущий символ; OldHigh – верхняя граница интервала; HighRange(X) – исходная верхняя граница кодируемого символа; LowRange(X) – исходная нижняя граница кодируемого символа. Применяя данные формулы к вычислению границ символа W, получаем:

OldLow = 0,5, OldHigh = 1,

HighRange(W) = 0,5, LowRange(W) = 0,4,

NewHigh = 0,5 + (1-0,5)*0,5 = 0,75,

NewLow = 0,5 + (1-0,5)*0,4 = 0,7.

Аналогичным образом выполняется кодирование символа I, для которого новые интервалы также можно вычислить по приведенной формуле:

OldLow = 0,7, OldHigh = 0,75,

HighRange(I) = 0,4, LowRange(I) = 0,2,

NewHigh = 0,7 + (0,75-0,7)*0,4 = 0,72,

NewLow = 0,7 + (0,75-0,7)*0,2 = 0,71.

Ниже, в табл. 1 представлены значения границ при кодировании строки SWISS_MISS.

Символ		Границы
S	L	0.0+(1.0-0.0)*0.5 = 0.5
	H	0.0+(1.0-0.0)*1.0 = 1.0
W	L	0.5+(1.0-0.5)*0.4=0.70
	H	0.5+(1.0-0.5)*0.5=0.75
I	L	0.7+(0.75-0.7)*0.2=0.71
	H	0.7+(0.75-0.7)*0.4=0.72

S	L	0.71+(0.72-0.71)*0.5=0.715
	H	0.71+(0.72-0.71)*1.0=0.72
S	L	0.715+(0.72-0.715)*0.5=0.7175
	H	0.715+(0.72-0.715)*1.0=0.72
_	L	0.7175+(0.72-0.7175)*0.0=0.7175
	H	0.7175+(0.72-0.7175)*0.1=0.71775
M	L	0.7175+(0.71775-0.7175)*0.1=0.717525
	H	0.7175+(0.71775-0.7175)*0.2=0.717550
I	L	0.717525+(0.717550-0.717525)*0.2=0.717530
	H	0.717525+(0.717550-0.717525)*0.4=0.717535
S	L	0.717530+(0.717535-0.717530)*0.5=0.7175325
	H	0.717530+(0.717535-0.717530)*1.0=0.717535
S	L	0.7175325+(0.717535-0.7175325)*0.5=0.71753375
	H	0.7175325+(0.717535-0.7175325)*1.0=0.717535

Конечный выходной код – это последнее значение переменной Low, равное 0.71753375, из которого следует взять лишь восемь цифр 71753375 для записи в файл.

Теперь рассмотрим возможность восстановления закодированной информации по восьми цифрам 71753375 и известным интервалам символов. Первая из восьми цифр – это 7, т.е. 0,7. Она принадлежит одному из заданных интервалов [0.5, 1), который соответствует символу S. Поэтому первый декодированный символ – это S. Теперь вернемся к рис. 3 и заметим, что второй символ был представлен в интервале символа S, т.е. [0.5, 1). Но для удобства декодирования его лучше представить в исходном интервале [0, 1). Для этого достаточно интервал [0.5, 1) увеличить до начального, т.е. умножить на два и границы сдвинуть на величину 0.5*2=1 (рис. 13).

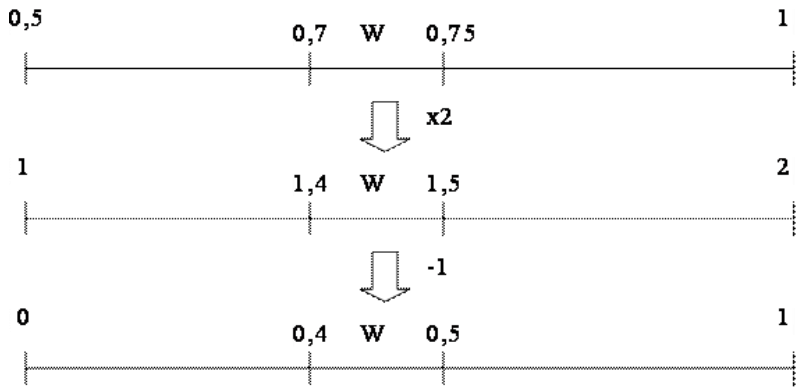


Рис. 13. Схема восстановления исходных интервалов символа W

Применяя данную схему к числу 0.71753375, получаем нижнюю границу следующего закодированного символа как будто он был начальным при кодировании:

$$0.71753375*2-1 = 0.4350675.$$

Полученное значение принадлежит диапазону [0.4, 0.5), который соответствует символу W. Затем, также полученное число 0.4350675 следует нормировать, что в общем случае выполняется по формуле:

$$Code = (Code-LowRange(X))/(HighRange(X)-LowRange(X)),$$

где Code – текущее значение кода. Например, пользуясь этой формулой применительно к коду 0.71753375, получаем значение

$$Code = (0.71753375-0.5)/(1-0.5)= 0.4350675,$$

которое в точности совпадает с предыдущей схемой вычисления. Аналогичным образом выполняется декодирование всех символов строки. В табл. 2 представлены коды, вычисляемые при декодировании символов. Здесь можно заметить, что процесс декодирования в данном случае можно остановить, если значение кода равно нулю.

Таблица 2. Вычисление кодов при декодировании

Символ	Code-Low	Область	
S	0.71753375 – 0.5	= 0.21753375	/ 0.5 = 0.4350675
W	0.4350675 – 0.4	= 0.0350675	/ 0.1 = 0.350675
I	0.350675 – 0.2	= 0.150675	/ 0.2 = 0.753375
S	0.753375 – 0.5	= 0.253375	/ 0.5 = 0.50675
S	0.50675 – 0.5	= 0.00675	/ 0.5 = 0.0135
_	0.0135 – 0	= 0.0135	/ 0.1 = 0.135
M	0.135 – 0.1	= 0.035	/ 0.1 = 0.35
I	0.35 – 0.2	= 0.15	/ 0.2 = 0.75
S	0.75 – 0.5	= 0.25	/ 0.5 = 0.5
S	0.5 – 0.5	= 0	/ 0.5 = 0

Однако это не всегда так. Бывают случаи, когда ноль содержит в себе код очередного символа, а не означает конец процедуры декодирования. Здесь возникает проблема завершения декодирования. Для этого используют специальный символ eof, говорящий о том, что он является последним и декодирование последовательности можно завершить. При этом частота этого символа очевидно должна быть маленькой по сравнению с частотой символов алфавита последовательности, например, [0,999999 1).

Описанный выше процесс кодирования невозможно реализовать на практике, т.к. в нем предполагается, что в переменных Low и High хранятся числа с неограниченной точностью. По существу, результат кодирования – это вещественное число с очень большой точностью. Например, файл объемом 1Мб будет сжиматься, скажем, до 500 КБ, в котором будет записано одно число. Арифметические операции с такими числами реализовать сложно и долго. Поэтому любая практическая реализация арифметического кодера должна основываться на операциях с целыми числами, которые не должны быть слишком длинными. Рассмотрим такую реализацию, в которой переменные Low и High будут целыми числами длиной 16 или 32 бита. Эти переменные будут хранить верхние и нижние концы текущего подинтервала, но мы не будем им позволять неограниченно расти. Анализ табл. 1 показывает, что как только самые левые цифры переменных Low и High становятся одинаковыми, они уже не меняются в дальнейшем. Следовательно, эти цифры можно выдвинуть за скобки и работать с оставшейся дробной частью. После сдвига цифр

мы будем справа дописывать 0 в переменную Low, а в переменную High – цифру 9. Для того, чтобы лучше понять весь процесс, можно представлять себе эти переменные как левый конец бесконечно длинного числа. Число Low имеет вид xxxx00… а число High = yyyy99…

Проблема состоит в том, что переменная High в начале должна равняться 1, однако мы интерпретируем Low и High как десятичные дроби меньшие 1. Решение заключается в присвоении переменной High значения 9999…, которое соответствует бесконечной дроби 0,9999…, равной 1.

Как это все работает? Закодируем этим способом ту же строку SWISS_MISS. Первая буква S определена диапазоном [0,5 1) и формально границы равны

L=0.0+(1.0-0.0)*0.5=0.5

H=0.0+(1.0-0.0)*1.0=1.0

но в нашем случае данные формулы следует преобразовать, чтобы переменные Low и High были целыми. Поэтому запишем их в таком виде:

Low=0+(10000-0)*0.5=5000

High=0+(10000-0)*1.0=10000

но по условию граница High является открытой, т.е. не включает число 10000, поэтому от конечного значения нужно отнять 1:

High=0+(10000-0)*1.0-1=9999

Таким образом, получили формулы для вычисления целочисленных границ Low и High, и процесс кодирования будет выглядеть так (табл. 3).

Табл. 3. Кодирование сообщения сдвигами

1		2			4	5
S	L=	0+(10000-0)*0.5	=	5000		5000
	H=	0+(10000-0)*1.0-1	=	9999		9999
W	L=	5000+(10000-5000)*0.4	=	7000	7	0000
	H=	5000+(10000-5000)*0.5	=	7499		4999
I	L=	0+(5000-0)*0.2	=	1000	1	0000
	H=	0+(5000-0)*0.4-1	=	1999		9999
S	L=	0+(10000-0)*0.5	=	5000		5000
	H=	0+(10000-0)*1.0-1	=	9999		9999
S	L=	5000+(10000-5000)*0.5	=	7500		7500
	H=	5000+(10000-5000)*1.0-1	=	9999		9999
_	L=	7500+(10000-7500)*0.0	=	7500	7	5000
	H=	7500+(10000-7500)*0.1-1	=	7749		7499
M	L=	5000+(7500-5000)*0.1	=	5250	5	2500
	H=	5000+(7500-5000)*0.2	=	5499		4999
I	L=	2500+(5000-2500)*0.2	=	3000	3	0000
	H=	2500+(5000-2500)*0.4-1	=	3499		4999
S	L=	0+(5000-0)*0.5	=	2500		2500
	H=	0+(5000-0)*1.0-1	=	4999		4999
S	L=	2500+(5000-2500)*0.5	=	3750	3750	3750

H=	2500+(5000-2500)*1.0-1	=	4999	4999

На последнем шаге операции кодирования записываются все 4 цифры и полученная выходная последовательность имеет вид: 717533750.

Декодер работает в обратном порядке. В начале переменным Low и High присваиваются значения 0000 и 9999 соответственно, а переменной Code значение 7175. На основе этой информации требуется определить первый закодированный символ. Для этого число переменной Code нужно корректно представить в интервале от 0 до 1. В самом начале интервал такой и есть, поэтому частота символа, соответствующая значению Code будет равна

index = 7175/10000=0,7175.

Эта частота попадает в диапазон [0,5 1) и соответствует символу S. Теперь границы Low и High пересчитываются, так как это делалось в кодере и принимают значения 5000 и 9999 соответственно. Так как значащие цифры этих переменных отличаются, то переменная Code остается прежней и величина

index = (7175-5000)/(10000-5000)=0,4350.

Это значение попадает в диапазон [0,4 0,5) и соответствует символу W. После этого величины Low и High принимают значения 7000 и 7499 и после отбрасывания значащей цифры переходят в 0000 и 4999 соответственно, а переменная Code преобразуется в 1753. Таким образом раскодируется вся последовательность.

На практике обычно значение index принимает целочисленные значения, которые вычисляются по формуле

index = ((Code-Low)*10-1)/(High-Low+1)

и округляют до ближайшего целого.

Может показаться, что приведенный выше пример не производит никакого сжатия. Для того чтобы выяснить степень сжатия результаты

кодирования нужно перевести в двоичную форму. Так как из конечного интервала

[0.71753375 0.717535)

можно выбрать любое число, выберем наименьшее для хранения – это 717534, которому соответствует битовое представление 10101111001011011110 и составляет 20 бит. И строка из 10 символов сжимается в 20 бит. Хорошее ли это сжатие? Для этого нужно найти энтропию кодируемой последовательности и она будет равна

$$H(X) = -0,5\log_2 0,5 - 0,2\log_2 0,2 - 0,1\log_2 0,1 - 0,1\log_2 0,1 - 0,1\log_2 0,1 \approx 1,96$$

бит/сим

и составит величину

$$I = H(X) \cdot 10 \approx 19,6 \text{ бит,}$$

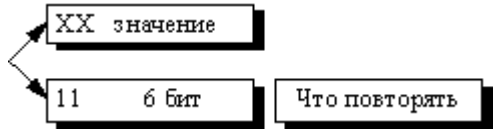
что в целых значениях равно 20 бит. Следовательно, полученный код достиг минимально возможного значения и является оптимальным. В общем случае можно показать, что при достаточно большой последовательности арифметический кодер всегда приводит к оптимальным результатам сжатия, т.е. является наилучшим среди всех энтропийных кодеров.

4.11. Алгоритм RLE

Первый вариант алгоритма

Данный алгоритм необычайно прост в реализации. Групповое кодирование — от английского Run Length Encoding (RLE) — один из самых старых и самых простых алгоритмов архивации графики. Изображение в нем (как и в нескольких алгоритмах, описанных ниже) вытягивается в цепочку байт по строкам растра. Само сжатие в RLE происходит **за счет того, что в исходном изображении встречаются цепочки одинаковых байт**. Замена их на пары <счетчик повторений, значение> уменьшает избыточность данных.

В данном алгоритме признаком счетчика служат единицы в двух верхних битах считанного файла:



Соответственно оставшиеся 6 бит расходуются на счетчик, который может принимать значения от 1 до 64. Строку из 64 повторяющихся байтов мы превращаем в два байта, т.е. сожмем в 32 раза.

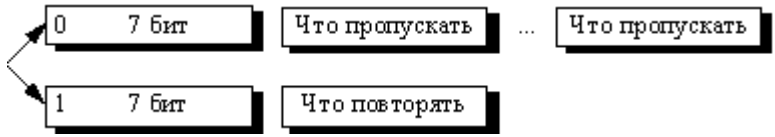
Алгоритм рассчитан на деловую графику — изображения с большими областями повторяющегося цвета. Ситуация, когда файл увеличивается, для этого простого алгоритма не так уж редка. Ее можно легко получить, применяя групповое кодирование к обработанным цветным фотографиям. Для того, чтобы увеличить изображение в два раза, его надо применить к изображению, в котором значения всех пикселов больше двоичного 11000000 и подряд попарно не повторяются.

Данный алгоритм реализован в формате PCX.

Второй вариант алгоритма

Второй вариант этого алгоритма имеет больший максимальный коэффициент архивации и меньше увеличивает в размерах исходный файл.

Признаком повтора в данном алгоритме является единица в старшем разряде соответствующего байта:



Как можно легко подсчитать, в лучшем случае этот алгоритм сжимает файл в 64 раза (а не в 32 раза, как в предыдущем варианте), в худшем

увеличивает на 1/128. Средние показатели степени компрессии данного алгоритма находятся на уровне показателей первого варианта.

Похожие схемы компрессии использованы в качестве одного из алгоритмов, поддерживаемых форматом TIFF, а также в формате TGA.

4.12. Алгоритм LZW

Название алгоритм получил по первым буквам фамилий его разработчиков — Lempel, Ziv и Welch. Сжатие в нем, в отличие от RLE, осуществляется уже за счет одинаковых цепочек байт.

Процесс сжатия выглядит достаточно просто. Мы считываем последовательно символы входного потока и проверяем, есть ли в созданной нами таблице строк такая строка. Если строка есть, то мы считываем следующий символ, а если строки нет, то мы заносим в поток код для предыдущей найденной строки, заносим строку в таблицу и начинаем поиск снова. LZW реализован в форматах GIF и TIFF.

LZW - это способ сжатия данных, который извлекает преимущества при повторяющихся цепочках данных. Поскольку растровые данные обычно содержат довольно много таких повторений, LZW является хорошим методом для их сжатия и раскрытия.

В данный момент давайте рассмотрим обычное кодирование и декодирование с помощью LZW-алгоритма. В GIF используется вариация этого алгоритма.

Первой вещью, которую мы делаем при LZW-сжатии является инициализация нашей цепочки символов. Чтобы сделать это, нам необходимо выбрать код размера (количество бит) и знать сколько возможных значений могут принимать наши символы. Давайте положим код размера равным 12 битам, что означает возможность запоминания 0FFF, или 4096, элементов в нашей таблице цепочек. Давайте также предположим, что мы имеем 32 возможных различных символа. (Это соответствует, например, картинке с 32 возможными цветами для каждого пиксела.) Чтобы инициализировать таблицу, мы установим соответствие кода #0 символу #0, кода #1 символу #1, и

т.д., до кода #31 и символа #31. На самом деле мы указали, что каждый код от 0 до 31 является корневым. Больше в таблице не будет других кодов, обладающих этим свойством.

Процесс сжатия выглядит достаточно просто. Мы считываем последовательно символы входного потока и проверяем, есть ли в созданной нами таблице строк такая строка. Если строка есть, то мы считываем следующий символ, а если строки нет, то мы заносим в поток код для предыдущей найденной строки, заносим строку в таблицу и начинаем поиск снова.

Пусть мы сжимаем последовательность 45, 55, 55, 151, 55, 55, 55. Тогда, согласно изложенному выше алгоритму, мы поместим в выходной поток сначала код очистки <256>, потом добавим к изначально пустой строке “45” и проверим, есть ли строка “45” в таблице. Поскольку мы при инициализации занесли в таблицу все строки из одного символа, то строка “45” есть в таблице. Далее мы читаем следующий символ 55 из входного потока и проверяем, есть ли строка “45, 55” в таблице. Такой строки в таблице пока нет. Мы заносим в таблицу строку “45, 55” (с первым свободным кодом 258) и записываем в поток код <45>. Можно коротко представить архивацию так: “45” — есть в таблице; “45, 55” — нет. Добавляем в таблицу <258>“45, 55”. В поток: <45>; “55, 55” — нет. В таблицу: <259>“55, 55”. В поток: <55>; “55, 151” — нет. В таблицу: <260>“55, 151”. В поток: <55>; “151, 55” — нет. В таблицу: <261>“151, 55”. В поток: <151>; “55, 55” — есть в таблице; “55, 55, 55” — нет. В таблицу: “55, 55, 55” <262>. В поток: <259>;

Последовательность кодов для данного примера, попадающих в выходной поток: <256>, <45>, <55>, <55>, <151>, <259>.

Особенность LZW заключается в том, что для декомпрессии нам не надо сохранять таблицу строк в файл для распаковки. Алгоритм построен таким образом, что мы в состоянии восстановить таблицу строк, пользуясь только потоком кодов.

Мы знаем, что для каждого кода надо добавлять в таблицу строку, состоящую из уже присутствующей там строки и символа, с которого начинается следующая строка в потоке.

Следует отметить, что в целях экономии памяти ЭВМ таблицу цепочек можно эффективно представлять, учитывая следующую ее особенность: если, например, цепочка ASDFK входит в таблицу цепочек, то ASDF тоже входит в нее. Это обстоятельство наводит на мысль, что вместо запоминания в таблице всей цепочки, достаточно запомнить код для ASDF, за которым следует замыкающий символ K.

Для повышения степени сжатия изображений данным методом часто используется одна «хитрость» реализации этого алгоритма. Некоторые изображения, подвергаемые сжатию с помощью LZW, имеют часто встречающиеся цепочки одинаковых значений, например 12 12 12 … или 32 32 32 … и т. п. Их непосредственное сжатие будет генерировать выходной код 12 12 258 и т.д. Спрашивается, можно ли в этом частном случае повысить степень сжатия? Оказывается да, если мы оговорим следующие действия по кодированию и декодированию. Кодирование таких цепочек будем осуществлять следующим образом. Первый цвет 12 записывает с его же кодом из таблицы 12. Следующий цвет тоже 12, тогда добавляем в таблицу строку 12 12 с кодом 258 и в выходной поток сразу заносим этот код, т.е. 258. Смотрим входной поток дальше. Если на вход опять поступает цвет 12, он есть в таблице, смотрим следующий – 12, последовательность 12 12 тоже есть в таблице, смотрим дальше – 12, последовательности 12 12 12 присваиваем код 259 и заносим его в выходной поток. В результате на выходе получаем последовательность 12 258 259 …, которая гораздо короче прямого кодирования стандартным методом LZW.

Можно показать, что такая последовательность будет корректно восстановлена. Декодировщик сначала читает первый код – это 12, которому соответствует цвет 12. Затем читает код 258, но этого кода в его таблице нет. Но мы уже знаем, что такая ситуация возможна только в том случае, когда добавляемый символ равен первому символу только что считанной последовательности, т.е. 12. Поэтому он добавит в свою таблицу строку 12 12 с кодом 258, а в выходной поток поместит 12 12. И так может быть раскодирована вся цепочка кодов.

Мало того, описанное выше правило кодирования мы можем применять в общем случае не только к подряд идущим одинаковым цветам точек, но и к последовательностям, у которых очередной добавляемый символ равен первому символу цепочки**.**

4.13. LZW в GIF

Алгоритм LZW имеет несколько особенностей своей реализации в формате сжатия изображений gif. Первая особенность – это переменный размер кода таблицы цепочек, который не может превышать 12 бит, т.е. не превышать числа 4095. Вторая особенность состоит в использовании двух специальных кодов – это код обновления (реинициализации) таблицы цепочек , и код завершения потока символов .

В самом начале своей работы алгоритм определяет количество цветов, используемых в изображении. В случае GIF их максимум может быть 256, т.к. любое изображение, даже с большим набором цветов преобразуется в 256 цветовое пространство. Минимум может быть 2 цвета. Если используется только два цвета, то начальный размер кодов в таблице равен 3 битам. Причем коду 0 ставится цвет 0, а коду 1 – цвет 1. Коды 4 и 5 соответствуют коду очистки таблицы и коду . При большем количестве цветов размер кода таблицы равен числу бит N, приходящихся на один пиксел. При этом специальные коды равны и . Начальный размер кодов в таблице записывается в заголовок GIF файла.

Кодирование пикселей изображения начинается кодами размером бит. По мере накопления таблицы будут увеличиваться значения кодов и как только очередной код достигает значения , то это значит, что значение необходимо увеличить на 1, иначе значение кода превысит прежний размер в бит.

Разработчики формата GIF ограничили максимальный размер кодов в таблице 12 битами. Это значит, что когда код достигает значения , то размер увеличивать уже нельзя. Но в то же время и размер кодов становится больше 12 бит. Как разрешить данную ситуацию? Простым решением является выполнить перегенерирование таблицы последовательностей, после чего она будет содержать только цепочки длины, равной 1, т.е. значения пикселей и плюс специальные коды. Таким образом, она перейдет в то же состояние, в котором была на

момент начала кодирования изображения. А для того, чтобы декодировщик знал, что в определенный момент произошло перегенерирование таблицы, кодировщик в выходной поток записывает код управляющего символа .

4.14. Адаптивный алгоритм Хаффмана

Адаптивное кодирование Хаффмана (также называемое **динамическое кодирование Хаффмана**) — адаптивный метод, основанный на кодировании Хаффмана. Он позволяет строить кодовую схему в поточном режиме (без предварительного сканирования данных), не имея никаких начальных знаний из исходного распределения, что позволяет за один проход сжать данные. Преимуществом этого способа является возможность кодировать на лету.

Существует несколько реализаций этого метода, наиболее примечательными являются **«FGK»** (ФГК: Фоллер, Галлагер и Кнут) и алгоритм Виттера.

ФГК алгоритм

Он позволяет динамически регулировать дерево Хаффмана, не имея начальных частот. В ФГК дереве Хаффмана есть особый внешний узел, называемый *0-узел*, используемый для идентификации входящих символов. То есть, всякий раз, когда встречается новый символ — его путь в дереве начинается с нулевого узла. Самое важное — то, что нужно усекать и балансировать ФГК дерево Хаффмана при необходимости, и обновлять частоту связанных узлов. Как только частота символа увеличивается, частота всех его родителей должна быть тоже увеличена. Это достигается путём последовательной перестановки узлов, поддеревьев или и тех и других.

Важной особенностью ФГК дерева является принцип братства (или соперничества): каждый узел имеет два потомка (узлы без потомков называются листами) и веса идут в порядке убывания. Благодаря этому свойству дерево можно хранить в обычном массиве, что увеличивает производительность.

Алгоритм Виттера

Код представляется в виде древовидной структуры, в которой каждый узел имеет соответствующий вес и уникальный номер.

Цифры идут вниз, и справа налево.

Веса должны удовлетворять принципу братства. Таким образом, если A является родительским узлом B и C является потомком B, то W(A)> W(B)> W(C).

Вес — это всего лишь количество символов, встреченных ранее.

Набор узлов с одинаковыми весами представляют собой **блок.**

Чтобы получить код для каждого узла, в случае двоичного дерева мы могли бы просто пройти все пути от корня к узлу, записывая, например, «1» если мы идем направо, и «0» если мы пойдем налево.

Также в этом алгоритме используется специальный лист (узел без потомков), NYT (от англ. not yet transmitted — ещё не переданный символ), из которого «растут» новые, ранее не встречавшиеся, символы.

Кодер и декодер начинают только с корневого узла, который имеет максимальный вес. В начале это и есть наш NYT узел.

Когда мы передаем NYT символ, мы должны передать вначале код самого узла, а затем данные.

Для каждого символа, который уже находится в дереве, мы должны только передавать код конечных узлов (листов).

Для каждого передающегося символа передатчик и приёмник выполняют процедуру обновления:

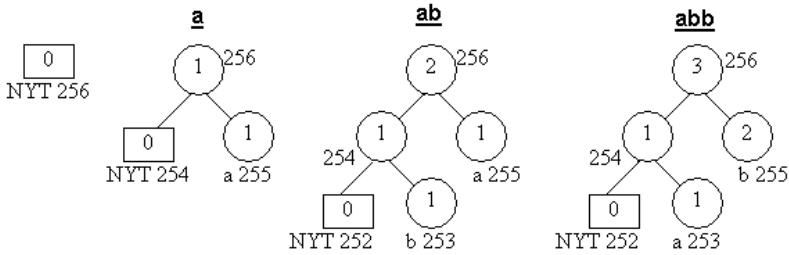
1. Если текущий символ является не встречавшимся — добавить к NYT два дочерних узла: один для следующего NYT, другой для символа. Увеличить вес нового листа и старого NYT и

переходить к шагу 4. Если текущий символ является не NYT, перейти к листу символа.

2. Если этот узел не имеет наибольший вес в блоке, поменять его с узлом, имеющим наибольшее число, за исключением, если этот узел является родительским элементом[3]
3. Увеличение веса для текущего узла
4. Если это не корневой узел зайти в родительский узел затем перейдите к шагу 2. Если это корень, окончание.

Примечание: замена узлов означает замену весов и соответствующих символов, но не чисел.

Пример



Начинаем с пустого дерева.

Для **«a»** передаём его двоичный код.

NYT порождает два дочерних узла: 254 и 255. Увеличиваем вес корня. Код **«a»**, связанный с узлом 255, становится 1.

Для **«b»** передавать 0 (код NYT узла), затем его двоичный код.

NYT порождает два дочерних узла: 252 для NYT и 253 для **b.** Увеличиваем веса 253, 254 и корня. Код для **«b»** равен 01.

Для следующего «b» передаётся 01.

Идём в лист 253. У нас есть блок весов в 1 и наибольшее число в блоке 255, так что меняем веса и символы узлов 253 и 255, увеличиваем вес, идём в корень и увеличиваем вес корня.

В будущем код «b» — это 1, а для «a» — это теперь 01, который отражает их частоту.

4.15. Адаптивный алгоритм Хаффмена с упорядоченным деревом

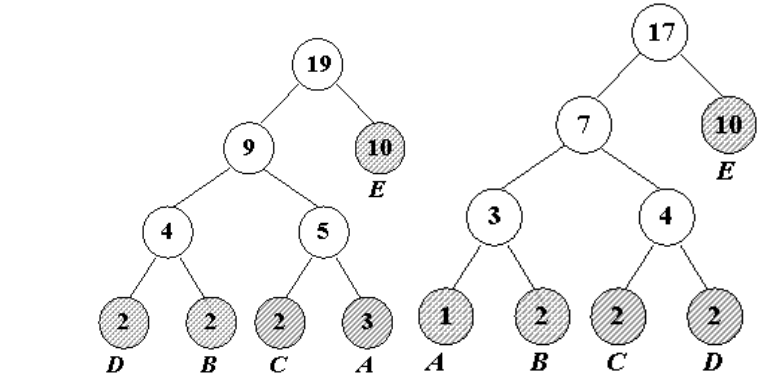
В отличие от рассмотренных ранее методов сжатия адаптивный (динамический) алгоритм Хаффмена ***является более практичным, однопроходным, не требующим передачи таблицы кодов.***

В зависимости от поступающих на вход символов алгоритм перестраивает кодовое дерево, корректируя его в соответствии с изменением статистики входного потока. При декодировании происходит аналогичный процесс.

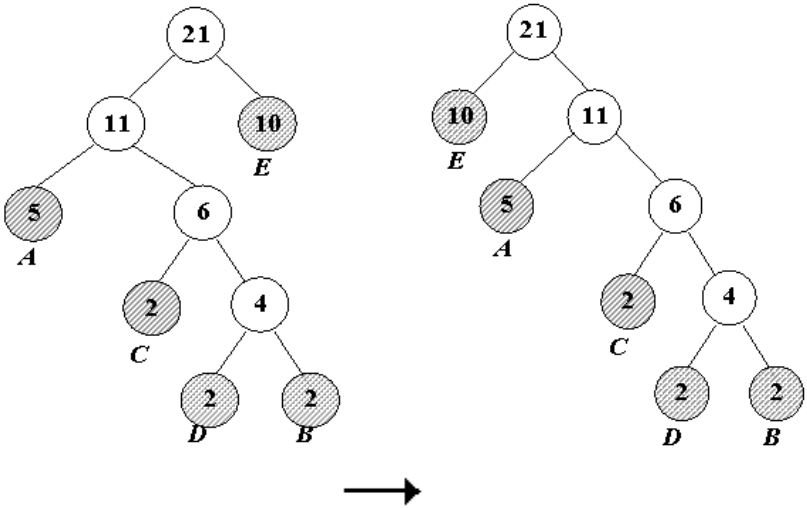
Для обеспечения возможности правильного кодирования/ декодирования используется ***упорядоченная структура кодового дерева.***

Упорядоченным деревом Хаффмена *называется бинарное дерево, узлы которого могут быть перечислены в порядке неубывания их веса слева-направо на каждом уровне и снизу-вверх по уровням, причем в этом перечне узлы, имеющие общего родителя, находятся на одном уровне. Если кодовое дерево упорядочено таким образом, то при изменении веса существующего узла в дереве достаточно поменять местами два узла: узел, вес которого нарушил упорядоченность, и последний из следующих за ним узлов меньшего веса. После перемены узлов местами необходимо пересчитать веса всех узлов-предков. В начале работы алгоритма дерево содержит только один специальный символ* ***<ESC>****, всегда имеющий частоту 0. Он необходим для занесения в дерево нового символа, который передается непосредственно после <ESC>. При появлении нового символа кодовое дерево перестраивается: справа от узла <ESC> добавляется новый лист, и затем, если необходимо, дерево упорядочивается. Левые ветви кодового дерева помечаются 0, а правые – 1.*

Приведем пример упорядоченного таким образом дерева (рис.4.14 *а*). Добавим в это дерево две буквы «***A***», тогда узлы *«**A**»* и «***D***» должны поменяться местами (рис.4.14 *б*). Если добавить еще две буквы «***A***», то необходимо будет поменять местами сначала узел «***A***» и узел, родительский для «***D***» и «***B***», а затем узел «***E***» и узел-брат «***E***» (рис.4.14 *в-г*).



а) ***б)***



в) ***г)***

Рисунок 9.14

Пример 1 Построим адаптивный код Хаффмена для сообщения ***ACCBCAAABC***.

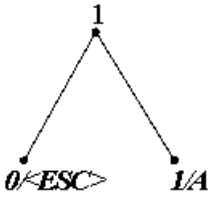
Таблица кодирования данного сообщения представлена *таблицей 4.11*, соответствующие изменения кодового дерева показаны на рис.4.14.

Условимся в дальнейшем символом в кавычках обозначать двоичный восьмибитный код символа по таблице ASCII+.

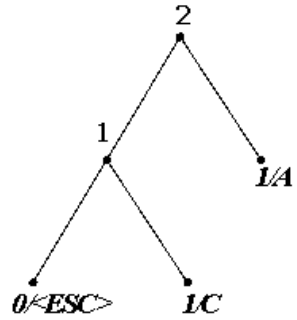
Таблица 4.11

Входные данные	**Код**	**Длина кода**	**Номер дерева**
A	‘A’	8	1

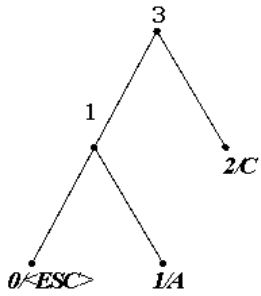
C	0'C'	9	2
C	01	2	3
B	00'B'	10	4
C	1	1	5
A	01	2	6
A	01	2	7
A	11	2	8
B	101	3	9
C	11	2	
		å=***41***	



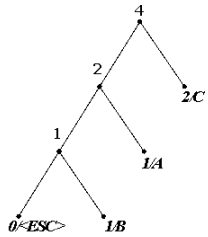
1)



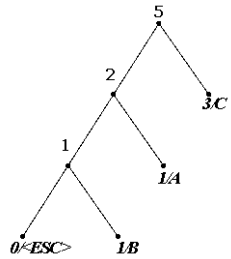
2)



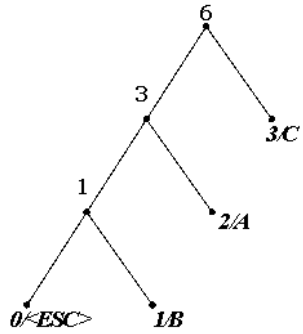
3)



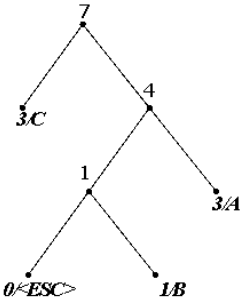
4)



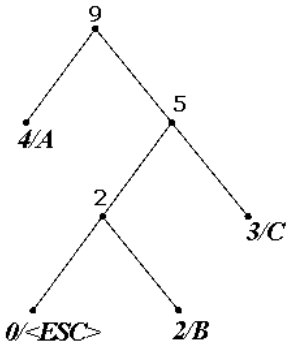
5)



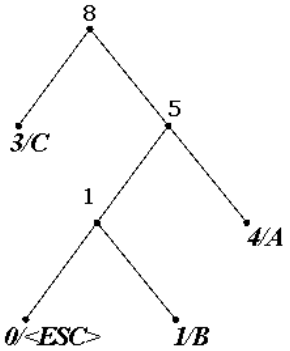
6)



7)



8)



9)

Рисунок 4.15

Длина сжатого сообщения $L^{Адапт.Хаффмена} = 41$ *(бит)*.

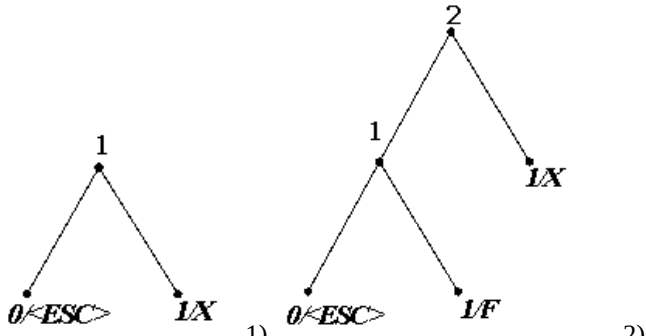
Длина несжатого сообщения в коде *ASCII+* $L^{ASCII+} = 80$ *(бит)*.

Пример 2 Распаковать сообщение ‘***X***’0‘***F***’00‘***Z***’1111101100‘***A***’111010, закодированное по адаптивному алгоритму Хаффмена. Вычислить длину кода сжатого сообщения и несжатого сообщения в коде *ASCII+*.

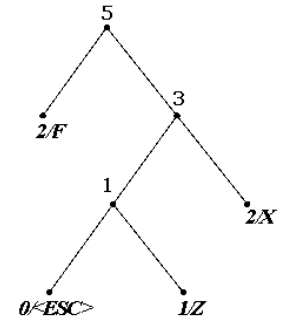
Процесс декодирования иллюстрируется *таблицей 4.12* и рис. 4.16.

Таблица 4.12

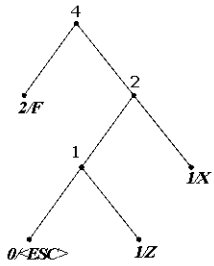
Входной код	**Символ**	**Длина кода**	**Номер дерева**
‘*X*’	***X***	8	1
0‘*F*’	***F***	9	2
00‘*Z*’	***Z***	10	3
11	***F***	2	4
11	***X***	2	5
101	***Z***	3	6
100‘*A*’	***A***	11	7
11	***X***	2	8
10	***F***	2	9
10	***F***	2	
		å=***51***	



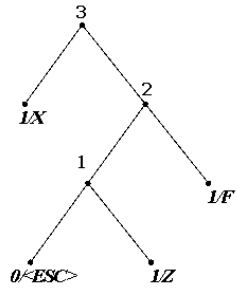
1) 2)


5
3
2/F
1
2/X
0/<ESC>
1/Z

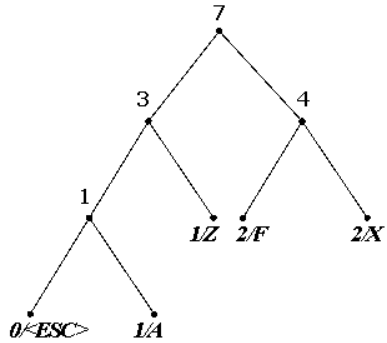
3)


4
2
2/F
1
1/X
0/<ESC>
1/Z

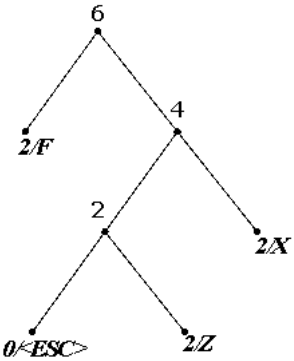
4)


3
2
1/X
1
1/F
0/<ESC>
1/Z

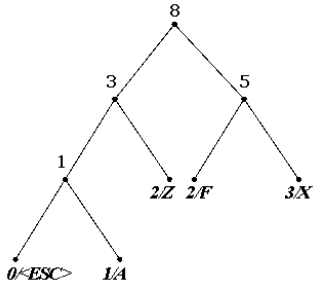
5)


7
3
4
1
1/Z
2/F
2/X
0/<ESC>
1/A

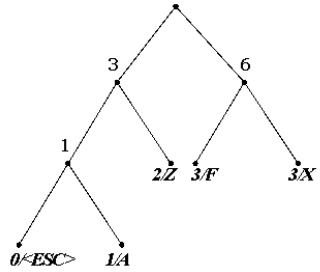
6)


6
4
2/F
2
2/X
0/<ESC>
2/Z

7)



8)



9)

Рисунок 4.16

Длина сжатого сообщения $\boldsymbol{L}^{Адапт.Хаффмена} = \boldsymbol{51}$ *(бит).*

Длина сообщения в коде *ASCII+* $\boldsymbol{L}^{ASCII+} = \boldsymbol{80}$ *(бит).*

Достоинством алгоритма *адаптивного кодирования* информации является достижение высокой степени сжатия, поскольку в нем учитываются локальные изменения вероятностей символов входного потока.

Кодирование *Хаффмена* характеризуется минимальной избыточностью при условии кодирования каждого символа сообщения кодовой комбинацией из алфавита {**0**, **1**} и обеспечивает достаточно высокое качество сжатия информации.

К недостаткам данного способа кодирования относится зависимость степени сжатия от близости вероятностей символов значениям отрицательной степени 2, что связано с необходимостью округления длины каждой кодовой комбинации до большего целого, так как каждый символ кодируется целым числом битов. Поэтому по степени сжатия *адаптивный алгоритм Хаффмена* в большинстве случаев уступает методу *арифметического кодирования*.

4.16. Теория приближенных рассуждений

Под приближенными рассуждениями понимается процесс, при котором из нечетких посылок получают некоторые следствия, возможно, тоже нечеткие. Приближенные рассуждения лежат в основе способности человека понимать естественный язык, разбирать почерк, играть в игры, требующие умственных усилий, в общем, принимать решения в сложной и не полностью определенной среде. Эта способность рассуждений в качественных, неточных терминах отличает интеллект человека от интеллекта вычислительной машины.

Основным правилом вывода в традиционной логике является правило modus ponens, согласно которому мы судим об истинности высказывания B по истинности высказываний A и $A \to B$. Например, если A — высказывание "Джон в больнице", B — высказывание "Джон болен", то если истинны высказывания "Джон в больнице" и "Если Джон в больнице, то он болен", то истинно и высказывание "Джон болен".

Во многих привычных рассуждениях, однако, правило modus ponens используется не в точной, а в приближенной форме. Так, обычно мы знаем, что A истинно и что $A^* \to B$, где A^* есть, в некотором смысле, приближение A. Тогда из $A^* \to B$ мы можем сделать вывод о том, что B приближенно истинно.

Далее мы обсудим способ формализации приближенных рассуждений, основанный на понятиях, введенных нами на предыдущей лекции. Однако, в отличие от традиционной логики, нашим главным инструментом будет не правило modus ponens, а так называемое композиционное правило вывода, весьма частным случаем которого является правило modus ponens.

4.16.1. Композиционное правило вывода

Композиционное правило вывода — это всего лишь обобщение следующей знакомой процедуры. Предположим, что имеется кривая $y = f(x)$ (см. рис. 4.17(А)) и задано значение $x = a$. Тогда из того, что $y = f(x)$ и $x = a$, мы можем заключить, что $y = b = f(a)$.

Обобщим теперь этот процесс, предположив, что a — интервал, а $f(x)$ — функция, значения которой суть интервалы, как на рисунке 4.17(Б). В этом случае, чтобы найти интервал $y = b$, соответствующий интервалу a, мы сначала построим цилиндрическое множество $\bar{a}$ с основанием a и найдем его пересечение I с кривой, значения которой суть интервалы. Затем спроектируем это пересечение на ось OY и получим желаемое значение y в виде интервала b.

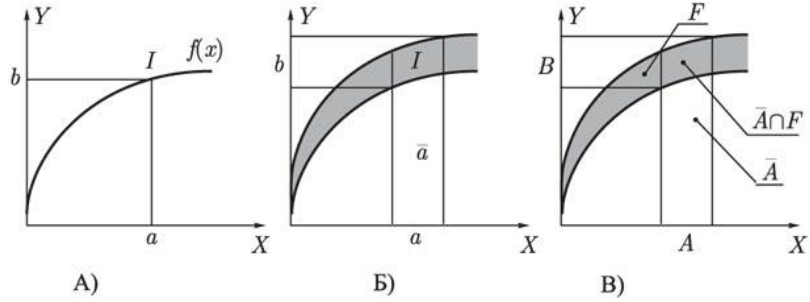


Рис. 4.17.

Чтобы продвинуться еще на один шаг по пути обобщения, предположим, что A — нечеткое подмножество оси OX, а F — нечеткое отношение в $OX \times OY$ (см. рис. 4.17(В)). Вновь образуя цилиндрическое нечеткое множество $\bar{A}$ с основанием A и его пересечение с нечетким отношением F, мы получим нечеткое множество $\bar{A} \cap F$, которое является аналогом точки пересечения I на рис. 4.17(А). Таким образом, из того, что $y = f(x)$ и $x = A$ — нечеткое подмножество оси OX, мы получаем значение y в виде нечеткого подмножества B оси OY.

Правило. Пусть U и V — два универсальных множества с базовыми переменными u и v, соответственно. Пусть A и F — нечеткие подмножества множеств U и $U \times V$. **Тогда композиционное правило вывода утверждает, что из нечетких множеств A и F следует нечеткое множество $B = A \circ F$. Согласно определению композиции нечетких множеств, получим**

$$\mu_B(v) = \bigvee_{u \in U} \left(\mu_A(u) \wedge \mu_F(u, v)\right).$$

Пример. Пусть $U = V = \{1, 2, 3, 4\}$,

А = МАЛЫЙ $= \{\langle 1|1\rangle, \langle 0,6|2\rangle, \langle 0,2|3\rangle, \langle 0|4\rangle\}$,

F=ПРИМЕРНО РАВНЫ $=$

	1	2	3	4
1	1	0,5	0	0
2	0,5	1	0,5	0
3	0	0,5	1	0,5
4	0	0	0,5	1

Тогда получим

$$B = [1 \quad 0,6 \quad 0,2 \quad 0] \circ \begin{bmatrix} 1 & 0,5 & 0 & 0 \\ 0,5 & 1 & 0,5 & 0 \\ 0 & 0,5 & 1 & 0,5 \\ 0 & 0 & 0,5 & 1 \end{bmatrix} = [1 \quad 0,6 \quad 0,5 \quad 0,2],$$

что можно проинтерпретировать следующим образом:

B = БОЛЕЕ ИЛИ МЕНЕЕ МАЛЫЙ,

если терм БОЛЕЕ ИЛИ МЕНЕЕ определяется как оператор увеличения нечеткости.

Словами этот приближенный вывод можно записать в виде

	u – МАЛЫЙ	предпосылка
	u, v – ПРИМЕРНО РАВНЫ	предпосылка
v	– БОЛЕЕ ИЛИ МЕНЕЕ МАЛЫЙ	приближенный вывод

4.16.2. Правило modus ponens как частный случай композиционного правила вывода

Как мы увидим ниже, правило modus ponens можно рассматривать как частный случай композиционного правила вывода. Чтобы установить эту связь, мы сперва обобщим понятие материальной импликации с пропозициональными переменными на нечеткие множества.

Пусть A и B — нечеткие высказывания и μ_A, μ_B — соответствующие им функции принадлежности. Тогда импликации $A \to B$ будет соответствовать некоторая функция принадлежности $\mu_{A \to B}$. По аналогии с традиционной логикой, можно предположить, что

$$A \to B \equiv \neg A \vee B.$$

Тогда

$$\mu_{A \to B}(x, y) = \max\{1 - \mu_A(x), \mu_B(y)\}.$$

Однако, это не единственное обобщение оператора импликации. В следующей таблице показаны различные интерпретации этого понятия.

Larsen	$\mu_{A\to B}(x,y)=\mu_A(x)\mu_B(y)$
Lukasiewicz	$\mu_{A\to B}(x,y)=\min\{1,1-\mu_A(x)+\mu_B(y)\}$
Mamdani	$\mu_{A\to B}(x,y)=\min\{\mu_A(x),\mu_B(y)\}$
Standard Strict	$\mu_{A\to B}(x,y)=\begin{cases}1, & \text{если } \mu_A(x)\leqslant\mu_B(y);\\ 0, & \text{в противном случае.}\end{cases}$
Godel	$\mu_{A\to B}(x,y)=\begin{cases}1, & \text{если } \mu_A(x)\leqslant\mu_B(y);\\ \mu_B(y), & \text{в противном случае.}\end{cases}$
Gaines	$\mu_{A\to B}(x,y)=\begin{cases}1, & \text{если } \mu_A(x)\leqslant\mu_B(y);\\ \frac{\mu_B(y)}{\mu_A(x)}, & \text{в противном случае.}\end{cases}$
Kleene-Dienes	$\mu_{A\to B}(x,y)=\max\{1-\mu_A(x),\ \mu_B(y)\}$
Kleene-Dienes-Lu	$\mu_{A\to B}(x,y)=1-\mu_A(x)+\mu_A(x)\mu_B(y)$

Определим теперь **обобщенное правило** modus ponens (generalized modus ponens).

Предпосылка	$A\to B$
Событие	A^*
Вывод	$A*\circ(A\to B)$

Приведенная формулировка имеет два отличия от традиционной формулировки правила modus ponens : во-первых, здесь допускается, что A, A^*, B — нечеткие множества, и, во-вторых, A^* необязательно идентично A .

4.16.3. Нечеткие экспертные системы

Логико-лингвистические методы описания систем основаны на том, что поведение исследуемой системы описывается в естественном (или близком к естественному) языке в терминах лингвистических переменных. Входные и выходные параметры системы рассматриваются как лингвистические переменные, а качественное описание процесса задается совокупностью высказываний следующего вида:

L_1 если A_{11} и/или A_2 и/или ... и/или A_{1m}, то B_{11} и/или ... и/или B_{1n},

L_2 если A_{21} и/или A_{22} и/или ... и/или A_{2m}, то B_{21} и/или ... и/или B_{2n},

.....

L_k если A_{k1} и/или A_{k2} и/или ... и/или A_{km}, то B_{k1} и/или ... и/или B_{kn},

где $A_{ij}\ i = 1, 2 \ldots, k\ j = 1, 2, \ldots, m$ — нечеткие высказывания, определенные на значениях входных лингвистических переменных, а $B_{ij}\ i = 1, 2, \ldots, k\ j = 1, 2, \ldots, n$ — нечеткие высказывания, определенные на значениях выходных лингвистических переменных. Эта совокупность правил носит название нечеткой базы знаний.

Подобные вычисления составляют основу нечетких экспертных систем. Каждая нечеткая экспертная система использует нечеткие утверждения и правила.

Затем с помощью операторов вычисления дизъюнкции и конъюнкции описание системы можно привести к виду

L_1: если A_1, то B_1,

L_2: если A_2, то B_2,

.....

L_k: если A_k, то B_k,

где $A_1, A_2, \ldots, A_k$ — нечеткие множества, заданные на декартовом произведении X универсальных множеств входных лингвистических переменных, а $B_1, B_2, \ldots, B_k$ — нечеткие множества, заданные на декартовом произведении Y универсальных множеств выходных лингвистических переменных.

В основе построения логико-лингвистических систем лежит рассмотренное выше композиционное правило вывода Заде.

Преимущество данной модели - в ее универсальности. Нам неважно, что именно на входе — конкретные числовые значения или некоторая неопределенность, описываемая нечетким множеством. Но за данную универсальность приходится расплачиваться сложностью системы — нам приходится работать в пространстве размерности $m \times n$. Поэтому этой общей моделью на практике пользуются довольно редко. Обычно же используют ее упрощенный вариант, называемый нечетким выводом. Он основывается на предположении, что все входные лингвистические переменные имеют известные нам числовые значения (как и бывает довольно часто на практике). Также обычно не используют более одной выходной лингвистической переменной.

Нечетким логическим выводом (fuzzy logic inference) называется аппроксимация зависимости $Y = f(X_1, X_2, \ldots, X_n)$ каждой выходной лингвистической переменной от входных лингвистических переменных и получение заключения в виде нечеткого множества, соответствующего текущим значениям входов, с использованием нечеткой базы знаний и нечетких операций. Основу нечеткого логического вывода составляет **композиционное правило Заде**.

В общем случае нечеткий вывод решения происходит за три (или четыре) шага:

1) **этап фаззификации**. С помощью функций принадлежности всех термов входных лингвистических переменных и на основании задаваемых четких значений из универсумов входных лингвистических

переменных определяются степени уверенности в том, что выходная лингвистическая переменная принимает конкретное значение. Эта степень уверенности есть ордината точки пересечения графика функции принадлежности терма и прямой $x =$ четкое значение ЛП.

2) **этап непосредственного нечеткого вывода**. На основании набора правил — нечеткой базы знаний — вычисляется значение истинности для предпосылки каждого правила на основании конкретных нечетких операций, соответствующих конъюнкции или дизъюнкции термов в левой части правил. В большинстве случаев это либо максимум, либо минимум из степеней уверенности термов, вычисленных на этапе фаззификации, который применяется к заключению каждого правила. Используя один из способов построения нечеткой импликации, мы получим нечеткую переменную, соответствующую вычисленному значению степени уверенности в левой части правила и нечеткому множеству в правой части правила.

Обычно в качестве вывода используется минимизация или правила продукции. При минимизирующем логическом выводе выходная функция принадлежности ограничена сверху в соответствии с вычисленной степенью истинности посылки правила (нечеткое логическое И). В логическом выводе с использованием продукции выходная функция принадлежности масштабируется с помощью вычисленной степени истинности предпосылки правила.

3) **этап композиции (агрегации, аккумуляции)**. Все нечеткие множества, назначенные для каждого терма каждой выходной лингвистической переменной, объединяются вместе, и формируется единственное нечеткое множество — значение для каждой выводимой лингвистической переменной. Обычно используются функции **MAX** или **SUM**.

4) **этап дефаззификации (необязательный)**. Используется тогда, когда полезно преобразовать нечеткий набор значений выводимых лингвистических переменных к точным. Имеется достаточно большое количество методов перехода к точным значениям (по крайней мере, 30). Два примера общих методов — "методы полной интерпретации" и "по максимуму". В методе полной интерпретации точное значение выводимой переменной вычисляется как значение "центра тяжести" функции принадлежности для нечеткого значения. В методе

максимума в качестве точного значения выводимой переменной принимается максимальное значение функции принадлежности.

В теории нечетких множеств процедура дефаззификации аналогична нахождению характеристик положения (математического ожидания, моды, медианы) случайных величин в теории вероятности. Простейшим способом выполнения процедуры дефаззификации является выбор четкого числа, соответствующего максимуму функции принадлежности. Однако пригодность этого способа распространяется лишь на одноэкстремальные функции принадлежности. Для многоэкстремальных функций принадлежности часто используются следующие методы дефаззификации:

1) **COG** (Center Of Gravity) — "центр тяжести". Физическим аналогом этой формулы является нахождение центра тяжести плоской фигуры, ограниченной осями координат и графиком функции принадлежности нечеткого множества.

2) **MOM** (Mean Of Maximums) — "центр максимумов". При использовании метода центра максимумов требуется найти среднее арифметическое элементов универсального множества, имеющих максимальные степени принадлежностей.

3) **First Maximum** — "первый максимум" — максимум функции принадлежности с наименьшей абсциссой.

Функциональная схема процесса нечеткого вывода в упрощенном виде представлена на рис. 4.18. На этой схеме выполнение первого этапа нечеткого вывода — фаззификации — осуществляет фаззификатор. За процедуру непосредственно нечеткого вывода ответственна машина нечеткого логического вывода, которая производит второй этап процесса вывода на основании задаваемой нечеткой базы знаний (набора правил), а также этап композиции. Дефаззификатор выполняет последний этап нечеткого вывода — дефаззификацию.

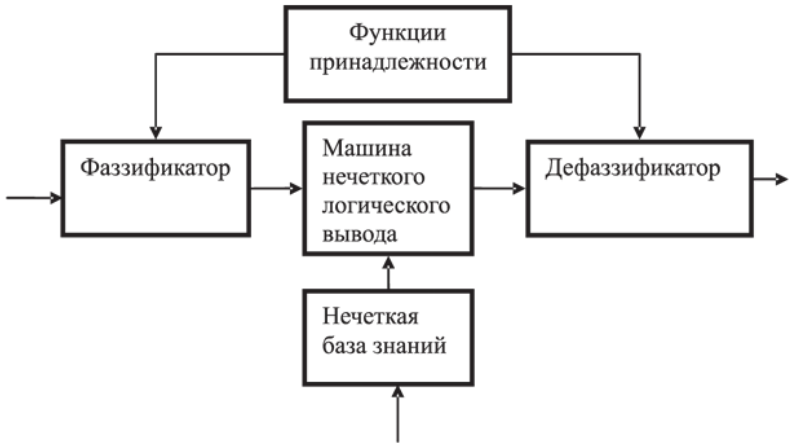


Рис.4.18.

Рассмотрим алгоритм нечеткого вывода на конкретном примере.

Пусть у нас есть некоторая система, например, реактор, описываемая тремя параметрами: температура, давление и расход рабочего вещества. Все показатели измеримы, и множество возможных значений известно. Также из опыта работы с системой известны некоторые правила, связывающие значения этих параметров. Предположим, что сломался датчик, измеряющий значение одного из параметров системы, но знать его показания необходимо хотя бы приблизительно. Тогда встает задача об отыскании этого неизвестного значения (пусть это будет давление) при известных показателях двух других параметров (температуры и расхода) и связи этих величин в виде следующих правил:

- если Температура низкая и Расход малый, то Давление низкое;
- если Температура средняя, то Давление среднее;
- если Температура высокая или Расход большой, то Давление высокое.

В нашем случае Температура, Давление и Расход — лингвистические переменные. Опишем каждую из них.

Температура. Универсум (множество возможных значений) — отрезок $[0, 150]$. Начальное множество термов {Высокая, Средняя, Низкая}. Функции принадлежности термов имеют следующий вид:

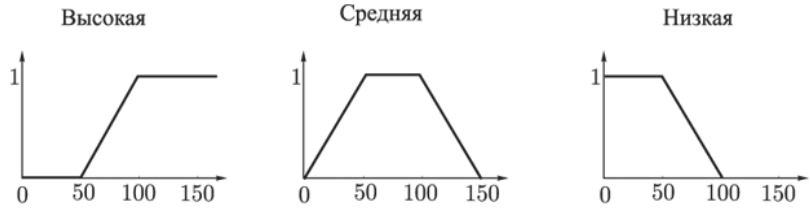


Рис.4.19.

Давление. Универсум — отрезок $[0, 100]$. Начальное множество термов {Высокое, Среднее, Низкое}. Функции принадлежности термов имеют следующий вид:

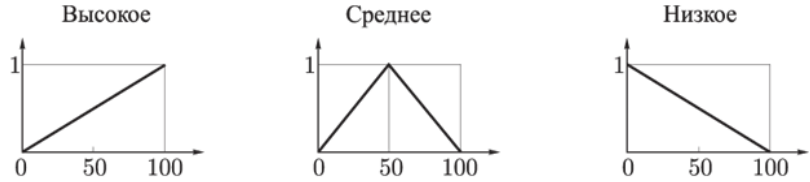


Рис.4.20.

Расход. Универсум — отрезок $[0, 8]$. Начальное множество термов {Большой, Средний, Малый}. Функции принадлежности термов имеют следующий вид:

Рис.4.21.

Пусть известны значения Температура 85 и Расход 3,5 . Произведем расчет значения давления.

Последовательно рассмотрим этапы нечеткого вывода:

Сначала по заданным значениям входных параметров найдем степени уверенности простейших утверждений вида "Лингв. переменная A есть Терм Лингв. переменной A ". Этот этап называется фаззификацией, т.е. переходом от заданных четких значений к степеням уверенности. Получаем следующие степени уверенности:

- Температура Высокая — 0,7;
- Температура Средняя — 1;
- Температура Низкая — 0,3;
- Расход Большой — 0;
- Расход Средний — 0,75;
- Расход Малый — 0,25.

Затем вычислим степени уверенности посылок правил:

- Температура низкая и Расход малый: min(Темп. Низкая, Расход Малый)= min(0.3, 0.25)=0.25;
- Температура Средняя: 1;
- Температура Высокая или Расход Большой: max(Темп. Высокая, Расход Большой)= max(0.7,0)=0,7.

Следует отметить также тот факт, что с помощью преобразований нечетких множеств любое правило, содержащее в левой части как конъюнкции, так и дизъюнкции, можно привести к системе правил, в левой части каждого будут либо только конъюнкции, либо только дизъюнкции. Таким образом, не уменьшая общности, можно рассматривать правила, содержащие в левой части либо только конъюнкции, либо только дизъюнкции.

Каждое из правил представляет из себя нечеткую импликацию. Степень уверенности посылки мы вычислили, а степень уверенности заключения задается функцией принадлежности соответствующего терма. Поэтому, используя один из способов построения нечеткой импликации, мы получим новую нечеткую переменную,

соответствующую степени уверенности в значении выходных данных при применении к заданным входным соответствующего правила. Используя определение нечеткой импликации как минимума левой и правой частей (определение Mamdani), имеем:

Рис.4.22.

Теперь необходимо объединить результаты применения всех правил.

Этот этап называется аккумуляцией. Один из основных способов аккумуляции — построение максимума полученных функций принадлежности. Получаем:

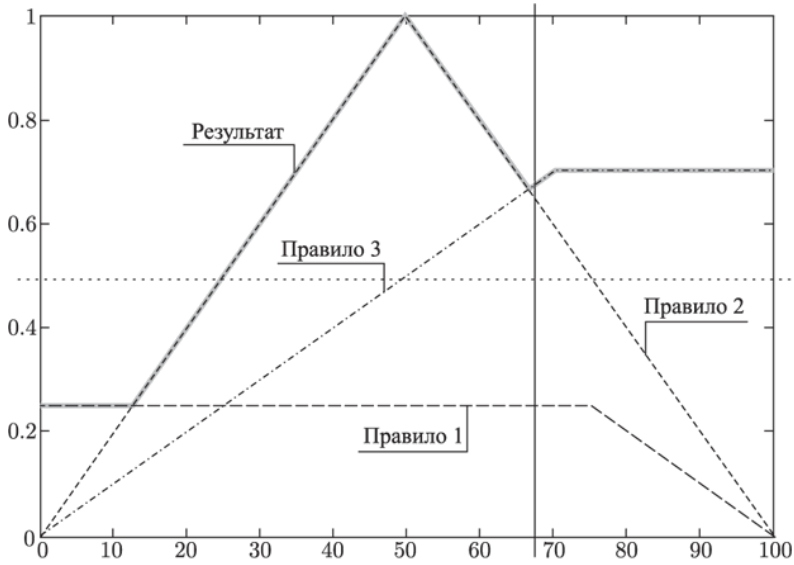


Рис. 4.23

Полученную функцию принадлежности уже можно считать результатом. Это новый терм выходной переменной Давление. Его функция принадлежности говорит о степени уверенности в значении давления при заданных значениях входных параметров и использовании правил, определяющих соотношение входных и выходных переменных. Но обычно все-таки необходимо какое-то конкретное числовое значение. Для его получения используется этап дефаззификации, т.е. получения конкретного значения из универса по заданной на нем функции принадлежности.

Существует множество методов дефаззификации, но в нашем случае достаточно метода первого максимума. Применяя его к полученной функции принадлежности, получаем, что значение давления — 50**.** Таким образом, если мы знаем, что температура равна 85, а расход рабочего вещества — 3,5, то можем сделать вывод, что давление в реакторе равно примерно 50.

5. Нечеткие алгоритмы

Нечеткие алгоритмы реализуют сложные зависимости между нечеткими множествами, нечеткими высказываниями, а также и между так называемыми нечеткими числами.

Нечеткий алгоритм – это последовательность нечетких инструкций. Инструкции классифицируются следующим образом:

1) **назначающие,** например, «x – высокий», «y – маленький»;

2) **нечеткие высказывания**, например, «Если X, то Y иначе Z»;

3) **безусловные активные**, например, «немного увеличить x», «чуть левее z».

Здесь могут добавляться и обычные «четкие» инструкции.

Вначале может производиться фаззификация для получения нечетких множеств по точечным значениям.

После получения результата может производиться дефаззификация – получение точечных значений выходных переменных. Разработаны специальные микропроцессоры, реализующие нечеткие алгоритмы. Управляющие устройства, основанные на нечеткой логике называют контроллерами нечеткой логики. В них вместо точных зависимостей используют формализованные с помощью лингвистических переменных знания экспертов.

Разработаны специальные пакеты программного обеспечения, реализующие нечеткие алгоритмы, например, FuzzyCalc.

5.1.Формализация понятия нечеткого алгоритма

Различные понятия, нечеткие по своей природе, могут быть формально описаны посредством нечетких множеств. Нечеткая логика, например, позволяет формализовать простые логические связки нечетких переменных с помощью нечетких высказываний. Для описания же сложных соотношений между переменными удобно использовать нечеткие алгоритмы.

Под алгоритмом понимается точно определенное правило действий (программа), для которого задано указание, как и в какой последовательности это правило необходимо применять к исходным данным задачи, чтобы получить ее решение. Характеристиками алгоритма являются:

а) детерминированность — однозначность результата процесса при неизменных исходных данных;

б) дискретность определяемого алгоритмом процесса — расчлененность его на отдельные элементарные акты, возможность выполнения которых человеком или машиной не вызывает сомнения;

в) массовость — исходные данные для алгоритма можно выбрать из некоторого множества данных, т.е. алгоритм должен обеспечить решение любой задачи из класса однотипных задач.

Нечеткий же алгоритм **, упрощенно говоря, определяется упорядоченным множеством** нечетких инструкций **(нечетких высказываний), которые содержат понятия, формализуемые нечеткими множествами**.

Под нечеткими инструкциями понимаются инструкции, содержащие нечеткое понятие, например, "пройти около 100 метров", а под машинными — инструкции, не содержащие никаких нечетких понятий: "пройти 100 метров". Здесь и далее четкие инструкции мы будем называть машинными, чтобы подчеркнуть возможность моделирования нечетких алгоритмов на ЭВМ, воспринимающих только чтение инструкций.

Приведем точное определение нечеткого алгоритма. Для формулировки необходимо ввести ряд первоначальных определений и обозначений.

Во-первых, вместо интервала $[0, 1]$, общепринятого множества значений функции принадлежности, рассматривается непустое множество W с отношением частичного порядка $\succ$ и операциями $\otimes$, $\oplus$, удовлетворяющими свойствам коммутативности, ассоциативности и дистрибутивности, а также содержащие нулевой (0) и единичный (1) элементы.

Во-вторых, рассматриваются инструкции следующего вида:

Start: go to L	(инструкция начала);
L : do F , go to L_1	(инструкция операции);
L : if P then go to ($L_1, \ldots, L_n$)	(инструкция условия);
L : halt	(инструкция окончания)

где $L_1, \ldots, L_n \in L$ — множество символов меток инструкций, $f \in F$ — символ оператора или функции, $P \in P$ — символ предикатов или условий.

Введение понятия инструкции позволяет определить понятие **программы**. Под программой понимается конечное множество инструкций π, содержащее единственную инструкцию начала. Никакие инструкции из π не имеют одинаковых меток.

В-третьих, определяется понятие W**-машины**. W-машина есть функция M, определенная на множестве символов $\{O\} \cup \{I\} \cup F \cup P$, для которых существуют множество входов X, множество состояний памяти M и множество выходов Y, а также выполнены следующие условия:

1. $M(I)\colon X \times M \to W$ (функция входов);
2. $\forall F \in F\, M(F)\colon M \times M \to W$ (функция операции);
3. $\forall P \in P, n > 0$ $M(P)\colon M \times \{1, \ldots, n\} \to W$ (функция условий);
4. $M(O)\colon M \times Y \to W$ (функция выхода).

Символы I и O обозначают вход и выход. Наконец, в-четвертых, программа π вместе с W-машиной, которая **допускает** π (т.е. машина определена на всех операциях F и условиях P, содержащихся в инструкциях операции и условия программы π), называется нечеткой программой. Следовательно, последовательностью инструкций, составляющих нечеткую программу, определяется нечеткий алгоритм.

Конкретные типы алгоритмов могут быть получены посредством выбора множеств $\{W, M, X, Y\}$, функций (входов, действий, условий, выходов), операций $\{\otimes,\ \oplus\}$, отношения $\succ$.

Рассмотрим некоторые случаи выбора множеств, функций, операций, отношений. Пусть W, U, V — непустые множества, тогда функцию f из $U \times V$ в W будем называть W-функцией f из U в V; $f(v|u)$ есть степень, с которой значение функции в точке u есть v.

W-функция является вероятностной, если для любого $u \in U$ существует $f(v|u)$ и $\sum_{v \in V} f(v|u) = 1$.

W-функция является детерминированной, если для любого $u \in U$ существует $v_0 \in V : f(v_0|u) = 1$ и для любого $v \neq v_0$ $F(v|u) = 0$.

Если множество W с определенными на нем операциями и отношениями записать в виде четверки $(W, \otimes, \oplus, \succ)$, то:

- $W_X = \{[0,1], \max, \min, \leqslant\}$ — определяет максиминную машину;
- $W_n = \{\Re^+, +, \cdot, \leqslant\}$ — определяет взвешенную машину;
- $W_I = \{[0,1], \min, \max, \leqslant\}$ — минимаксную машину;
- $W_T = \{[0,1], \max, \cdot, \leqslant\}$ — максимально взвешенную машину;
- $W_N = \{\{0,1\}, \max, \min, \leqslant\}$ — недетерминированную машину.

Взвешенная машина является вероятностной, если функции входа, действий, условий, выхода являются вероятностными. Любая же

машина, в которой перечисленные функции являются детерминированными, называется детерминированной.

Рассмотрим программу π, которую допускает W-машина M. Для каждой пары меток $L', L'' \in L$ и пары состояний $m_1, m_2 \in M$ будем писать $(L', m_1) \xrightarrow{w} (L'', m_2)$, если в программе π либо имеется инструкция вида $L'\colon doF;\ go\ to\ L''$, где $w = M_F(m_2|m_1)$ есть степень, с которой осуществляется переход из состояния m_1 в состояние m_2, либо имеется инструкция вида $L'\colon if\ P\ then\ go\ to\ (L_1, \ldots, L_n)$, где $m_1 = m_2,\ \ L'' = L_k$ для некоторого $k\colon 1 \leqslant k \leqslant n$ и $w = M_P(k|m_1)$ есть степень, с которой осуществляется переход на метку L_k.

Выполнением программы π на W-машине M, допускающей π, называется конечная последовательность $xL_0m_0 \ldots L_nm_ny$. Выполнение возможно тогда и только тогда, если $w = w_0 \otimes w_1 \otimes \ldots . \otimes w_{n+1} \neq 0$, где $w_0 = M_I(m_0|x)$, $w_{n+1} = M_O(y|m_n)$, $w_i\colon (L_{i-1}, m_{i-1}) \xrightarrow{w_i} (L_i, m_i)$.

Таким образом, возможное выполнение определяет последовательность инструкций программы π, которая может быть реализована на W-машине. Таких последовательностей может быть несколько.

Приведем другую формулировку нечеткой программы, которая является частным случаем данного выше определения нечеткой программы, так как здесь рассматриваются машины с конечным

множеством состояний, которые моделируются конечными автоматами.

Для определения нечеткого алгоритма первоначально вводится понятие **обобщенной машины**, на основе которого формулируется понятие обобщенной нечеткой машины, которое позволяет формализовать понятие нечеткого алгоритма.

Обобщенная машина есть шестерка $A = (K, S, \Psi, s_0, T, W)$, где K и S — конечные непустые множества машинных инструкций и внутренних состояний соответственно, W — непустое множество с отношением частичного порядка $\succ$ и операциями $\otimes$, $\oplus$, удовлетворяющими свойствам коммутативности, ассоциативности и дистрибутивности, а также содержащие нулевой и единичный элементы; Ψ — W-функция переходов из состояния в состояние; $\Psi: K \times S \to S$; s_0 и T — начальное состояние и множество финальных состояний.

Для цепочки инструкций $k^* = k_1 k_2 \ldots k_n \in K^*$ (K^* — множество всевозможных цепочек инструкций) переходов из состояния s_0 в s определяется степенью $\Psi(s_0,\, k_1,\, s_1) \otimes \Psi(s_1,\, k_2,\, s_2) \otimes \ldots \otimes \Psi(s_{n-1},\, k_n,\, s_n)$.

Если l — пустая цепочка инструкций, то задается расширенная W-функция Ψ следующим образом:

$$\Psi(s, l, s') = \begin{cases} 0, & \text{если } s \neq s', \\ 1, & \text{если } s = s'. \end{cases}$$

Обобщенная нечеткая машина определяется парой (A, Σ), где A — обобщенная машина, Σ — конечное множество нечетких

инструкций и каждая нечеткая инструкция σ из Σ есть W - функция из S в K .

В литературе рассматриваются случаи, когда у обобщенной и обобщенной нечеткой маншп множества *W* различны. Возможность такого введения проиллюстрирована на рис. 5.1 выполнением обобщенной нечеткой машиной нечеткой инструкции о, когда машина находилась в состоянии *s*.

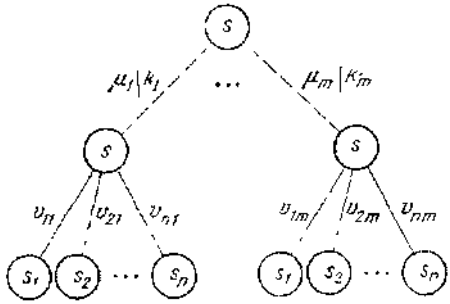


Рис. 5.1. Выполнение нечеткой инструкции σ

Величина μ указывает степень выбора для выполнения машиной инструкции k_i, при этом μ_i может быть, например, из [0, 1] × {0, 1}. Величина v_{ij} — степень перехода из состояния *s* в состояние s_j при получении инструкции k_i. Очевидно, что множество весов (степеней), определяющих переходы из состояния в состояние, не обязательно должно совпадать с множеством степеней *W*, определяющих выбор машинных инструкций.

Рассмотрим различные типы нечетких машин, полученные при различных *V* и *W*.

Пример. Если в определении обобщенной машины конкретизировать различным образом *W*, то получим различные типы обобщенных машин. Если $W = V = W_x = \{[0, 1], \max, \min, \leqslant\}$, то это нечеткая машина. Если $W = V = W_N$, то это недетерминированная машина. Если $W = V = W_n = \{\{0, 1\}, \max, \min, \leqslant\}$ и существует единственное s': $\Psi(s, k, s') = 1$ для любых *s*, *k*, то это деторминирванная машина. Если $W = V = W_p = \{[0, 1], +, \cdot, \leqslant\}$ и $\sum_{s' \in S} \Psi(s, k_1, s') = 1$, то это вероятностная машина.

Различные типы обобщенных нечетких машин можно получить в результате выбора различных W-функций (см. табл. 5.1, где числа указывают номера типов машин).

Таблица 5.1

Функция выбора \ Обобщенная машина	Нечеткая	Детерминированная	Недетерминированная	Вероятностная
Нечеткая	1	2	3	4
Детерминированная	5	6	7	8
Недетерминированная	9	10	11	12
Вероятностная	13	14	15	16

Определение 1. Пусть *(A,* ∑)—обобщеппая нечеткая машипа, где $A = (K, S, \Psi, s_0, T, W)$— обобщенная машина, ∑-конечное множество нечетких инструкций. Любой элемент

$\sigma = \sigma_1\sigma_2 \ldots \sigma_n \in \Sigma^*, \quad \sigma_i \in \Sigma,$ называется *элементарной* нечеткой программой, а любая функция из ∑* в *W* называется *нечеткой программой,* т. е. нечетким алгоритмом, например, регулярное выражение на множестве ∑.

Пусть задана некоторая обобщенная нечеткая машина (A, σ). Выполнение последовательности $\sigma = \sigma_1\sigma_2 \ldots \sigma_n$ на обобщенной машине A есть последовательность

$$s_0k_1s_1k_2 \ldots k_ns_n, \text{ где } s_i \in S,\ k_i \in K,\ s_n \in T.$$

Весом, соответствующим выполнению, является элемент

$$w \in W: \quad w = w_1 \otimes w_1' \otimes w_2 \otimes w_2' \ldots w_n \otimes w_n',$$

где

$$w_i = \sigma_i(k_i|s_{i-1}), \quad w_i' = \psi(s_i|k_i, s_{i-1}).$$

Выполнение возможно тогда и только тогда, если $w \neq 0$. Если σ_i и W принимают значения из различных множеств W и V, то вес, соответствующий выполнению, будет определяться парой $(w, v) = (w_1 \otimes \ \ldots \ \otimes w_n\,,\ v_1 \otimes \ \ldots \otimes v_n)$. В

этом случае говорят, что программа σ выполнима с весом (w, v), если $(w, v) > (0, 0)$.

Пример. Пусть $\otimes$ на W=[0, 1] означает min, $\otimes$ на $V = \{0, 1\}$ означает min или произведение $\times$, тогда обобщенная нечеткая машина будет нечетко-детерминированной и соответствующий вес будет вычисляться следующим образом:
$(w, v) = (w_1 \wedge w_2 \wedge \ldots \wedge w_n, 1)$ при этом выполнение возможно только тогда, когда
$\Psi(s_0, k_1, s_1) = \ldots = \Psi(s_{n-1}, k_n, s_n) = 1$.

Пример. Пусть имеется последовательность инструкций для водителя автомобиля и карта местности. Водителю предлагается найти место назначения, используя карту и последовательность нечетких инструкций, описывающих маршрут. Для простоты изложения предположим, что все точки на плоскости имеют только целочисленные координаты. Типичные инструкции для водителя: "двигаться прямо около L метров", "повернуть налево", "повернуть направо", "двигаться прямо до тех пор, пока не увидишь ..." (станцию, светофор).

Сконструируем соответствующую W-машину M. W-машина имеет множество состояний памяти M в виде упорядоченных троек $(a, b, \bar{v})$, где (a, b) — точка на плоскости, соответствующая местонахождению автомобиля, $\bar{v}$ — единичный вектор направления движения автомобиля. Множество входов $X = M$ и множество выходов Y состоят из упорядоченных пар (a, b); M_I — функция входов, соответствует тождественной функции; M_O — функция выходов, соответствует функции, отображающей каждую тройку $(a, b, \bar{v})$ в (a, b).

Машина M не имеет ни одной функции условия. Каждой инструкции, приведенной выше, соответствует функция операции. При

этом i -я инструкция в последовательности инструкций может быть преобразована в инструкцию операции вида do F_i; go to L_i. Совокупность таких инструкций и инструкций start: go to L_0 и L_n: halt, где n — длина последовательности, составляет программу π. Процесс выполнения программы π на машине M определяется последовательностью инструкций и картой местности. Краткости ради приведем только функцию операции для инструкции типа "двигаться прямо около L метров":

$$M_{F_L} = ((a_2, b_2, \bar{v}_2)|(a_1, b_1, \bar{v}_1)) =$$
$$= f_L\left(\sqrt{(a_2 - a_1)^2 + (b_2 - b_1)^2}\right) \times G\left((a_2, b_2, \bar{v}_2)|(a_1, b_1, \bar{v}_1)\right),$$

где $f_L(d)$ — степень (вес), соответствующая расстоянию d, $G((a_2, b_2, \bar{v}_2)_| (a_1, b_1, \bar{v}_1))$ — вес, соответствующий утверждению: "точка (a_2, b_2) и направление v_2 достижимы при движении прямо из точки (a_1, b_1) по направлению $v_1|$ ".

Примеры функций f_L и $G :: f_L(d) = [1 + ((L - d)/c)^2]^{-1}$, где c — параметр: $G\left((a_2, b_2, \bar{v}_2)|(a_1, b_1, \bar{v}_1)\right) = 1$ тогда и только тогда, если $\bar{v}_1 = \bar{v}_2$ вектор из (a_1, b_1) в (a_2, b_2) параллелен $\bar{v}_1$ и каждая точка на отрезке линии, проходящей через (a_1, b_1) и (a_2, b_2), имеющая целые координаты, есть точка на карте. Очевидно, что f_L зависит только от L, а G зависит только от карты. Другие функции операций могут быть построены аналогично. Нечеткий алгоритм, описывающий движение автомобиля к месту назначения, определяется конкретной последовательностью инструкций приведенного вида, которая реализуется на рассмотренной W-машине.

Пример. Рассматрим нечетко-детерминированную машину (в табл. 5.1 тип 2). Функция переходов из состояния в состояние считается детерминированной, а степень выбора машинной инструкции k_i, когда машина находится в состоянии s_i и получает нечетную инструкцию σ_i, определяется функцией $\sigma_i(s_i, k_i) = \min(f(s_i, \sigma_i, k_i), \lambda(s_i, \sigma_i, k_i))$, где f—функция осуществимости, а λ— функция выполнимости. Функция f дает объективные ограничения на выполнение определенных машинных инструкций. Например, если $f(s, \sigma, k) = 1$, когда машина в состоянии s получает нечеткую инструкцию σ, то машинная инструкция k может быть выполнена. Аналогично, если $f(s, \sigma, k) = 0$, то ипструкция k не может быть выполпепа. С другой сторопы, функция λ даст субъективную оцепку выполнимости машинной инструкции в рассматриваемой ситуации. Например, если $\lambda(s, \sigma, k_1) = 0{,}9$; $\lambda(s, \sigma, k_2) = 0{,}5$, то, когда машина находится в состоянии s и получена нечеткая инструкция σ, субъективно машинная инструкция k_1 дает лучшее выполнение, чем k_2. Функции λ и f рассматриваются для независимого анализ а субъективных и объективных ограничений.

Приведем другие примеры применения нечетких алгоритмов.

- Алгоритмы определения сложного нечеткого понятия A через более простые понятия, которые легко описать нечеткими множествами; результатом применения таких алгоритмов к некоторому элементу u области рассуждений U будет степень принадлежности u понятию A (степень, с которой элемент u может характеризоваться понятием A);
- Алгоритмы порождения, в результате выполнения которых порождается один из элементов нечеткого множества, которое описывает интересующее нас понятие (например, алгоритм порождения образцов почерка, рецептов приготовления пищи, сочинения музыки, предложений в естественном языке);
- Алгоритмы описания отношений между нечеткими переменными, например, в виде последовательности нечетких инструкций типа: "если x мало и x увеличить слегка, то y увеличится слабо"; такие алгоритмы позволяют приближенно описывать поведение систем, входные и выходные сигналы которых являются нечеткими подмножествами;

- Алгоритмы принятия решения, позволяющие приближенно описывать стратегию или важнейшее правило, например, алгоритм проезда перекрестка, содержащий последовательность действий, которые необходимо выполнить, при этом описания этих действий состоят из нечетких понятий типа: нормальная скорость, несколько секунд, медленно приближаться.

5.2. Способы выполнения нечетких алгоритмов

Для реализации поиска какого-либо выполнения нечеткого алгоритма $\sigma = \sigma_1\sigma_2 \ldots \sigma_n$ необходимо определить правила выбора машинной инструкции на каждом шаге. Правила выбора машинной инструкции и переходов из состояния в состояние зависят от типа нечеткой машины.

Определение обобщенной нечеткой машины позволяет различным образом определить не только выбор машинных инструкций, но и переходы из состянпя в состояние. Например, если целевое состояние не достигнуто (водитель не достиг заданного местоположения в последнем примере), то необходимо либо выполнить другую машинную инструкцию, либо осуществить другой переход из состояния в состояние. Если не существует пригодной машинной инструкции для данной нечеткой инструкции σ_i, то реализуется процедура возврата (шаг назад) к нечеткой инструкции σ_{i-1}.

Правила выбора машинной инструкции и переходов из состояния в состояние зависят от типа обобщенной нечеткой машины.

Выбор машинной инструкции:

а. **Нечеткий выбор**. Машина выбирает машинную инструкцию $k_i \in K(i, s_{i-1})$ с наивысшей степенью на каждом шаге $\sigma_i\colon \sigma_i(s_{i-1}, k) \geqslant \sigma_i(s_{i-1}, k')$ для любой инструкции $k' \in K$.

b. **Вероятностный выбор**. Машина на каждом шаге нечеткой инструкции σ_i выбирает инструкцию $k \in K(i, s_{i-1})$ с вероятностью p, пропорциональной нечеткой степени $\sigma_i(s_{i-1}, k)$

$$p = \sigma_i(s_{i-1}, k) / \sum_{k'} \sigma_i(s_{i-1}, k'), \qquad k' \in K(i, s_{i-1}).$$

c. **Недетерминированный выбор**. Машинная инструкция $k \in K(i, s_{i-1})$ выбирается недетерминированным образом.

Определение перехода из состояния в состояние:

a. **Нечеткий переход**. Машина переходит из состояния s_i в состояние $s\colon \Psi(s_i, k_i, s) \geqslant \Psi(s_i, k_i, s')$ для любого состояния $s' \in K(i, s_{i-1}, k_i)$.

b. **Вероятностный переход**. Машина переходит из состояния s_i в состояние s с вероятностью

$$p = \frac{\Psi(s_{i-1}, k_i, s)}{\sum\limits_{s' \in K(i, s_{i-1}, k_i)} \Psi(s_{i-1}, k_i, s')}.$$

c. В случае **детерминированного перехода** состояние, пригодное для машины, единственным образом определяется функцией переходов Ψ

Процедура возврата:

a. Вернуться на предыдущую нечеткую инструкцию.

b. Вернуться на нечеткую инструкцию, соответствующую машинной инструкции с наивысшей функцией принадлежности в ряде таких инструкций, просмотренных последовательно до выбранной нечеткой инструкции.

c. Осуществить возврат так же, как описано в пункте (b), но при этом машинная инструкция выбирается со степенью более высокой, чем выбранная перед этим.

5.3. Представление нечеткого алгоритма в виде графа

Во многих случаях нечеткий алгоритм удобно представлять в виде ориентированного графа. Каждой дуге ставят в соответствие инструкцию условия или инструкцию операции. Входные, выходные, внутренние переменные в нечетком алгоритме представляются нечеткими множествами. Выполнение алгоритма эквивалентно поиску в графе путей, связывающих помеченные вершины: начальные и конечные. Приведем необходимые для дальнейшего изложения известные определения графа и путей в графе.

Определение. Графом G называется тройка (V, U, φ), где $V = \{v\}$ — множество элементов, называемых вершинами графа; $U = \{u\}$ множество элементов, называемых ребрами графа, причем $V \cap U = \varnothing$; φ — функция, ставящая в соответствие каждому ребру $u \in U$ упорядоченную или неупорядоченную пару вершин (v_1, v_2), v_1 и v_2 называются концами ребра u. Если множество $U \cup V$ конечно, то граф называется конечным. Если $\varphi(u) = (v_1, v_2)$ — упорядоченная пара (т.е. $(v_1, v_2) \neq (v_2, v_1)$), то ребро u называется ориентированным ребром или дугой, исходящей из вершины v_1 и входящей в вершину v_2; v_1 называется началом, v_2 — концом дуги u. Граф, все ребра которого ориентированные, называется ориентированным графом.

Определение. Последовательность вершин и ребер графа $Gv_{i_0}u_1v_{i_2}u_2\ldots v_{i_{n-1}}u_nv_{i_n}$ называется путем $[v_{i_0}, v_{i_n}]$ из вершины v_{i_0} в вершину v_{i_n}, если $\varphi(u_k) = (v_{i_{k-1}}, v_{i_k})$ для $k = 1, 2, \ldots, n$. Вершина v_{i_0} называется началом, а v_{i_n} — концом пути; число n называется длиной пути.

Определение. Нечеткая программа есть четверка (X, Y, Z, G), где $X = (x_1, \ldots, x_l)$ — вектор входа, $Y = (y_1, \ldots, y_n)$ — вектор программы (внутренние переменные), $Z = (z_1, \ldots, z_m)$ — вектор выхода, G — ориентированный граф:

- x_i, y_i, z_i — нечеткие переменные, определяющие нечеткие множества на U, V, W;
- В графе G существует точно одна вершина, называемая начальной (стартовой), которая не является конечной вершиной никакой дуги, и существует точно одна вершина, называемая конечной (финальной), которая не является начальной вершиной никакой дуги: любая вершина графа находится на некотором пути из стартовой вершины S в финальную вершину H;
- В графе G любая дуга a, не ведущая в H, связана с нечетким отношением $R_a(X, Y)$ и нечеткой инструкцией $Y = f_a(X, Y)$; каждая дуга a, ведущая в H, связана с нечетким отношением $R_a(X, Y)$ и инструкцией $z = f_a(z, x, y)$, где R — нечеткое отношение, и f — нечеткая операция типа пересечения, объединения,

отрицания нечеткой арифметики, оператор размывания, оператор типа модификаторов и т.д.

Далее понадобятся следующие свойства нечетких множеств. Пусть *A* и *B* нечеткие подмножества соответственно на *U* и *V*, *R* — нечеткое отношение в $U \times V$, $A^* = R \circ B$ — нечеткое множество в *V*, индуцированное *B*, $B^* = R \circ A$ — нечеткое множество в *V*, индуцированное *A;*

ARB: $\mu_{ARB}(u, v) = \mu_A(u) \wedge \mu_R(u, v) \wedge \mu_B(v)$,

П*(R, U)*—проекция *R* на *U,* тогда справедливы следующие свойства:

Свойство 1. $\Pi(ARB, U) = A \cap A^*$, $\Pi(ARB, V) = B \cap B^*$.

Свойство 2. Если $A' = \Pi(ARB, U)$, то $A'RB = ARB$.

Свойство 3. *ARB** имеет наивысшую степень истинности среди всех *ARC* для любого нечеткого множества *C* из *V.*

В графе, описывающем алгоритм, выделяются дуги с условием и без условия (далее вместо знака равенства, обозначающего пересылку значепия, будем использовать стрелку ←*):*

а) без услошш: $Y \leftarrow f_a(X, Y)$;

б) с условием: если $R_a(X, Y)$, то $Y \leftarrow f_a(X, Y)$;

в) с условием:

$X \leftarrow \Pi(R_a(X, Y), U) = X \cap X^*$;

$Y \leftarrow \Pi(R_a(X, Y), V) = Y \cap Y^*$;

$Y \leftarrow f_a(X, Y)$.

Заметим, что условие б) эквивалентно условию в форме в) (свойство 1). Такую замену можно пояснить следующим образом: Y* представляет собой ту величину, которая удовлетворяет отношению R_a при наличии *X,* а пересечение $Y^* \cap Y$ есть та часть Y, которая удовлетворяет отношению R_a.

Возможны следующие условия завершения работы.

1. Если *Z* имеет не нулевую степепь принадлежности.
2. Если существуют элементы пз *Z,* чьи степени принаднежности выше, чем некоторый заранее заданный порог.
3. Если осуществлено заранее заданное число шагов или минимальное число шагов для достижения определенного заранее результата.
4. Если пользователь, оценивая по своему критерию результаты очередного цикла, останавливает работу.

Пример . Рассмотрим алгоритм, изображенный на рис. 5.2, где *несколько* = {0,8|4, 1|5, 0,8|6};

$$(\approx) = \{\mu(z, y) \mid (z, y):\ \mu(y, z) = 1 - |y - z|,\ \text{при}\ |y - z| \leqslant 1\};$$

p_1: $Y_1 \leftarrow \{1 \mid 10{,}5\}$; p_2: $Y_2 \leftarrow \{1 \mid 23\}$;

p_3: $Y_1 \leftarrow Y_1 +$ *несколько*; y_1: $Y_1 (\approx) Y_2$;

p_4: $Y_2 \leftarrow Y_2 -$ *несколько*; y_2: $Y_1 \urcorner (\approx) Y_2$;

p_5: $z \leftarrow Y_1 + Y_2$.

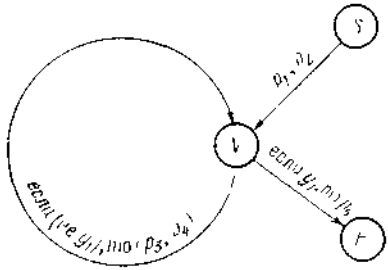


Рис. 5.2. Граф нечеткого алгоритма
[start: p_1; p_2; L: if y_2 then (p_3, p_4, go to L); if y_1 then p_5; halt]

Проследим последовательность выполнения.
На дуге *(SV)* присваиваются начальные значения (вторая строка табл. 2).

Таблица 2

	Последовательность вершин	Y_1	Y_2	Z
1	S	—	—	—
2	*SV*	1/10,5	1/23	—
3	*SVH*	∅	∅	∅
4	*SVV*	0,8/14,5, 1/15,5 0,8/16,5	0,8/19, 1/18 0,8/17	—
5	*SVVH*	0,5/16.5	0,5/17	0,5/33,5

Далее возможны последовательно следующие переходы

$$SVH:\ Y_1 \leftarrow Y_1 \cap Y_1^* \leftarrow \{1 \mid 10{,}5\} \cap \varnothing \leftarrow \varnothing,$$

$$Y_2 \leftarrow Y_2 \cap Y_2^* \leftarrow \{1 \mid 23\} \cap \varnothing \leftarrow \varnothing,$$

где

$$Y_1 = 1|10{,}5, \quad Y_2 = 1|23; \quad (\approx) = 0|(10{,}5,\ 23) \quad Y_1^* = (\approx)\circ Y_2 = \varnothing, \quad Y_2^* = (\approx)\circ Y_1 = \varnothing,$$

$$SVV:\ Z \leftarrow Y_1 + Y_2 \leftarrow \varnothing + \varnothing \leftarrow \varnothing,$$

$$Y_1 \leftarrow Y_1 \cap Y_1^* \leftarrow \{0{,}8\,|\,10{,}5\} \cap \{1\,|\,10{,}5\} \leftarrow \{1\,|\,10{,}5\},$$

$$Y_2 \leftarrow Y_2 \cap Y_2^* \leftarrow \{1\,|\,23\} \cap \{1\,|\,23\} \leftarrow \{1\,|\,23\},$$

где

$$Y_1^* = (\neg(\approx))\circ Y_2, \quad Y_2^* = Y_2 \circ (\neg(\approx)).$$

Далее вычисляем:

$$Y_1 \leftarrow Y_1 + \textit{несколько} \leftarrow \{1|10{,}5\} + \{0{,}8|4,\ 1|5,\ 0{,}8|6\} \leftarrow \leftarrow \{0{,}8|14{,}5,\ 1|15{,}5,\ 0{,}8|16\},$$

$$Y_2 \leftarrow Y_2 + \textit{несколько} \leftarrow \leftarrow \{1|23\} - \{0{,}8|4,\ 1|5,\ 0{,}8|6\} \leftarrow \{0{,}8|19,\ 1|18,\ 0{,}8|17\},$$

$$SVVH:\ Y_1 \leftarrow Y_1 \cap Y_1^* \leftarrow \{0{,}8\,|\,14{,}5,\ 1\,|\,15{,}5,\ 0{,}8\,|\,16{,}5\} \cap \cap \{0{,}5\,|\,16{,}5\} \leftarrow \{0{,}5\,|\,16{,}5\},$$

$$Y_2 \leftarrow Y_2 \cap Y_2^* \leftarrow \{0{,}8\,|\,19,\ 1\,|\,18,\ 0{,}8\,|\,17\} \cap \{0{,}5\,|\,17\} \leftarrow \leftarrow \{0{,}5\,|\,17\},$$

где

$$Y_1^* = (\approx)\circ Y_2 = \begin{matrix} \\ 14{,}5 \\ 15{,}5 \\ 16{,}5 \end{matrix} \begin{matrix} \begin{matrix} 19 & 18 & 17 \end{matrix} \\ \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \end{matrix} \circ \begin{matrix} \begin{matrix} 14{,}5 & 15{,}5 & 16{,}5 \end{matrix} \\ (0{,}8 \quad 1 \quad 0{,}8) \end{matrix} = 0{,}5\,|\,16{,}5;$$

$$Y_2^* = Y_1 \circ (\approx) = \begin{matrix} \\ 17 \\ 18 \\ 19 \end{matrix} \begin{matrix} \begin{matrix} 0{,}8 \end{matrix} \\ \begin{pmatrix} 0{,}8 \\ 1 \\ 0{,}8 \end{pmatrix} \end{matrix} \circ \begin{matrix} \begin{matrix} 19 & 18 & 17 \end{matrix} \\ \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0{,}5 \end{pmatrix} \end{matrix} \begin{matrix} 14{,}5 \\ 15{,}5 \\ 16{,}5 \end{matrix} = \{0{,}5\,|\,17\};$$

$$Z \leftarrow Y_1 + Y_2 \leftarrow \{0{,}5|16{,}5\} + \{0{,}5|17\} \leftarrow \{0{,}5|33{,}5\}.$$

Выполнение продолжается аналогично для *SVVV, SVVVH, SVVVV* и т. д., пока будет удовлетворено выбранное условие завершения.

5.4. Нечеткие алгоритмы обучения

Известно, что обучающиеся системы улучшают функционирование в процессе работы, модифицируя свою структуру или значение параметров. Предложено большое число способов описания и построения обучающихся систем. Все они предполагают решение следующих задач: выбор измерений (свойств, рецепторов); поиск отображения пространства рецепторов в пространство признаков,

которые осуществляют вырожденное отображение объектов; поиск критерия отбора признаков. Причем в различных задачах для получения хороших признаков могут понадобиться разные критерии отбора. При обучении необходимо отвлечься от различий внутри класса, сосредоточить внимание на отличии одного класса от другого и на сходстве внутри классов. Необходим достаточный уровень начальной организации обучающейся системы. Для сложной структурной информации необходима многоуровневая обучающаяся система.

Следует выделить следующие группы нечетких алгоритмов обучения: обучающийся нечеткий автомат, обучение на основе условной нечеткой меры, адаптивный нечеткий логический регулятор, обучение при лингвистическом описании предпочтения.

Рекуррентные соотношения в алгоритмах первых двух групп позволяют получать функцию принадлежности исследуемого понятия на множестве заранее известных элементов. В третьей группе нечеткий алгоритм обучения осуществляет модификацию нечетких логических правил для удержания управляемого процесса в допустимых границах. В четвертой группе нечеткий алгоритм обучения осуществляет поиск вырожденного отображения пространства свойств в пространство полезных признаков и модификацию на их основе описания предпочтения.

5.4.1. Обучающийся нечеткий автомат

Рассмотрим автомат с четким входом $i(t)$ и зависимым от времени нечетким отношением перехода $\delta(t)$. Пусть $\tilde{s}(t)$ — нечеткое состояние автомата в момент времени t на конечном множестве состояний $S = \{s_1, \ldots, s_n\}$ и i_l — оценка значения $i(t)$.Состояние автомата в момент времени $(t+1)$ определяется $\min$ - $\max$композицией:

$$\mu_{\tilde{s}(t+1)}(s_k) = \sup_j \min\left(\mu_{\tilde{s}(t)}(s_j),\ \mu_{\delta(t)}(s_x, i_l, s_j)\right),$$

или аналогично ей. Обучение направлено на изменение нечеткой матрицы переходов:

$$\mu_{\delta(t)}(s_k, i_l, s_j) = \mu_{\delta(t-1)}(s_k, i_l, s_k), \qquad j \neq k,$$

$$\mu_{\delta(t)}(s_k, i_l, s_k) = \alpha_k \mu_{\delta(t-1)}(s_k, i_l, s_k) + (1 - \alpha_k)\lambda_k(t),$$

где $0 < \alpha_k < 1,\ 0 < \lambda_k(t) \leqslant 1,\ k = 1, \ldots, n$.

Константа λ_k определяет скорость обучения. Начало работы автомата возможно без априорной информации $\mu_{\tilde{s}(0)}(s_k) = 0$ или 1, а также с априорной информацией $\mu_{\tilde{s}(0)}(s_k) = \lambda_k(0)$.

Величина $\lambda_k(t)$ зависит от оценки функционирования автомата. Доказано, что имеет место сходимость матрицы переходов, независимо от того, есть ли априорная информация, т.е. $\mu_{\tilde{s}(0)}(s_j)$ может быть любым значением из интервала $[0, 1]$.

Пример. На рис.5.3 изображена модель классификации образов.

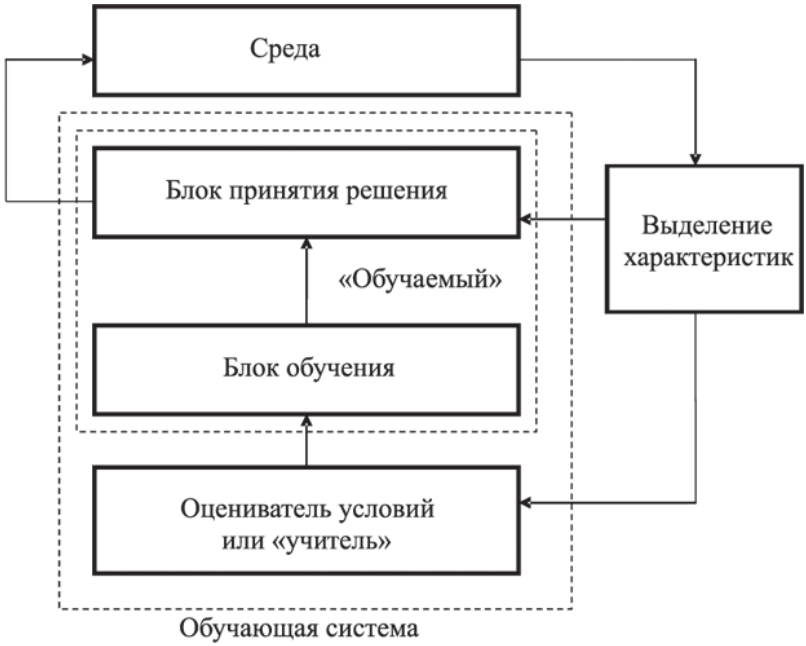


Рис.5.3. Модель классификации образов

Роль входа и выхода можно кратко объяснить следующим образом. Во время каждого интервала времени классификатор образов получает новый образец x' из неизвестной внешней среды. Далее x' обрабатывается в рецепторе, из которого поступает как в блок "обучаемый", так и в блок "учитель" для оценки. Критерий оценки должен быть выбран так, чтобы его минимизация или максимизация отражала свойства классификации (классов образов). Поэтому, благодаря естественному распределению образов, критерий может быть включен в систему, чтобы служить в качестве учителя для классификатора. Модель обучения формируется следующим образом. Предполагается, что классификатор имеет в распоряжении множество дискриминантных функций нескольких переменных. Система адаптируется к лучшему решению. Лучшее решение выделяет множество дискриминантных функций, которые дают минимум нераспознавания среди множества дискриминантных функций для данного множества образцов.

Моделируется поиск глобального экстремума функции следующим образом:

- область определения целевой функции делится на некоторое число подобластей (форма подобластей постоянно меняется) и описывается некоторым множеством точек;
- каждой точке приписывается состояние автомата, причем функция принадлежности в каждом состоянии указывает степень близости к оптимуму;
- выбирается состояние с максимальным значением функции принадлежности (эта точка называется кандидатом);
- формируется новая подобласть из точек, окружающих кандидата (размер подобласти растет, когда значение целевой функции в точке кандидата меньше, чем в других точках подобласти, и уменьшается в противоположном случае);
- когда подобласть пересекается с некоторой другой, или две точки-кандидаты находятся в одной подобласти, то подобласти разделяются, если степень разделения большая, или объединяются, если степень разделения малая;
- точки-кандидаты выбираются на этапе локального поиска в подобласти, затем во всей области среди точек-кандидатов ищется глобальная оптимальная точка;
- глобальный и локальный поиск осуществляется поочередно.

Алгоритм поиска глобального экстремума приведен на рис.5.4 .

Рис.5.4.

Пусть S — множество состояний, V — выходной универсум, σ — функция выхода (функция принадлежности, указывающая степень оптимума в состоянии s), $I(t)$ — текущее значение целевой функции, I_0— среднее значение $I(t)$.

Используется следующий алгоритм изменения функций перехода и выхода в случае глобального поиска:

если $I(t) > I_0$, то попытка успешна и

$$\mu_{\delta(t)}(s_k, s_j) = \alpha_k \mu_{\delta(t)}(s_k, s_j) + (1 - \alpha),$$

если $I(t) \leqslant I_0$, то попытка неудачна и

$$\mu_{\delta(t+1)}(u_i, s_j) = \alpha \mu_{\delta(t)}(u_i, s_j),$$

где $\alpha = 1 - |(I(t) - I_0)/I_0|$; $\alpha < 1$— гарантируемая сходимость.

В случае локального поиска:

если $I(t) > I_0$, то

$$\mu_{\delta(t+1)}(u_i, s_j) = \alpha\mu_{\delta(t)}(u_i, s_j) + (1 - \alpha),$$

если $I(t) \leqslant I_0$, то

$$\mu_{\delta(t+1)}(u_i, s_j) = \alpha\mu_{\delta(t)}(u_i, s_j).$$

5.4.2. Обучение на основе условной нечеткой меры

Пусть $X = \{x_1, \ldots, x_n\}$ — множество причин (входов) и $Y = \{y_1, \ldots, y_m\}$ — множество результатов. Если h — функция из X в интервал $[0, 1]$, $h(x_1) \leqslant \ \ldots \ \leqslant h(x_n)$ и g_x — нечеткая мера на X , то

$$\int_X h(x)g_x(\cdot) = \max_{i=1,\ldots,n} \min(h(x_i), g_X(H_i)),$$

где $H_i = \{x_i, \ldots, x_n\}$.

Задача состоит в оценке (уточнении) причин по нечеткой информации.

Пусть g_Y — нечеткая мера на Y, g_Y связана с g_X условной нечеткой мерой $\sigma_Y(\cdot|x)$:

$$g_Y = \int\limits_X \sigma_Y(\cdot|x) g_X.$$

Предполагается следующая интерпретация вводимых мер: g_X оценивает степень нечеткости утверждения "один из элементов X был причиной", $\sigma_Y(A|x)$, $A \subset Y$ оценивает степень нечеткости утверждения "один из элементов A является результатом благодаря причине x "; $g_Y(\{y\})$ характеризует степень нечеткости утверждения: " y — действительный результат".

Пусть $\mu_A(y)$ описывает точность информации A , тогда по определению $g_Y(A) = \int\limits_X \mu_A(y) g_X$.

Метод обучения должен соответствовать обязательному условию: при получении информации A нечеткая мера g_X меняется таким образом, чтобы $g_Y(A)$ возрастала. Предположим, что $g_X(\cdot)$ и $\sigma_Y(\cdot|x)$ удовлетворяют λ -правилу. Пусть $\sigma_Y(A|x_i)$ является убывающей, тогда

$$g_Y(A) = \bigvee_{i=1}^{n} \left[\sigma_Y(A|x_i) \wedge g_X(F_i)\right],$$

где $F_i = \{x_1, \ldots, x_i\}$. При этих условиях существует l :

$$g_Y(A) = \sigma_Y(A|x_l) \wedge g_X(F_l),$$

$$\sigma_Y(A|x_l) \wedge g_X(F_l) \geqslant \sigma_Y(A|x_{l-1}) \wedge g_X(F_{l-1}),$$

$$\sigma_Y(A|x_l) \wedge g_X(F_l) > \sigma_Y(A|x_{l+1}) \wedge g_X(F_{l+1}).$$

Обучение может быть осуществлено увеличением тех значений g_i ($i = 1, \ldots, n$) нечеткой меры g_X , которые увеличивают $g_Y(A)$, и уменьшением тех значений g_i ($i = 1, \ldots, n$) меры g_X , которые не увеличивают $g_Y(A)$. Можно показать, что на величину $g_Y(A)$ влияют только такие g_i , что $1 \leqslant i \leqslant l$. Следовательно, нечеткий алгоритм обучения следующий:

$$g^i = \alpha g^i + (1 - \alpha)\sigma_Y(A|x_i); \qquad i = 1, \ldots, l;$$
$$g^i = \alpha g^i; \qquad i = l + 1, \ldots, n.$$

Параметр $\alpha \in [0, 1]$ регулирует скорость обучения, т.е. скорость сходимости g^i . Чем меньше α , тем сильнее изменяется g^i . В приведенном алгоритме нет необходимости увеличивать g^i больше, чем на $\sigma_Y(A|x_i)$, так как большое увеличение g^i не влияет на $g_Y(A)$. Приведем некоторые свойства модели обучения.

Свойство 1. Если повторно поступает одна и та же информация, то происходит следующее:

а. новое g^i больше старого g^i ($i = 1, \ldots, l$) и новое g^i меньше старого g^i ($i = l + 1, \ldots, n$), следовательно, новая мера $g_Y(A)$ не меньше старой меры $g_Y(A)$, и новая мера

$$g_Y(A) = \sigma_Y(A|x_k) \wedge g_X(F_k), \qquad k \leqslant l;$$

b. при предположении $\sigma_Y(A|x_1) > \sigma_Y(A|x_2)$, $k < l$, g^1 сходится к $\sigma_Y(A|x_1)$ и g^i сходится к 0 для $i = 2, \ldots, n$.

Свойство 2. Если поступает одна и та же информация повторно: $h_A(y) = c$ для всех y, то

$$\sigma_Y(A|x) = \int_X c\sigma_Y(\cdot|x) = c, \quad \sigma_Y(A) = c \wedge g_X(X)$$

.

Следовательно, $l = n$ и g^i сходится к c для всех i.

Свойство 3. Предельное значение g^i не зависит от начального значения тогда, когда на вход повторно поступает одна и та же информация.

Пример. Пусть $\sigma_Y(\{y_j\}|x_i)$ подчиняется λ-правилу и имеет вид:

$$\sigma_Y(\{y_j\}|x_i) = \begin{array}{c} \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{array} \begin{array}{c} \begin{array}{ccccc} y_1 & y_2 & y_3 & y_4 & y_5 \end{array} \\ \begin{pmatrix} 0{,}7 & 0{,}23 & 0{,}16 & 0{,}8 & 0{,}39 \\ 0{,}4 & 0{,}64 & 0{,}32 & 0{,}16 & 0{,}08 \\ 0{,}16 & 0{,}49 & 0{,}57 & 0{,}24 & 0{,}16 \\ 0{,}08 & 0{,}16 & 0{,}04 & 0{,}64 & 0{,}32 \\ 0{,}17 & 0{,}34 & 0{,}25 & 0{,}45 & 0{,}5 \end{pmatrix} \end{array}.$$

Ясно, что y_j является результатом в основном причины x_j. На рис. 5.5 показано изменение априорной нечеткой плотности g_x при повторном появлении на входе одной и той же информации.

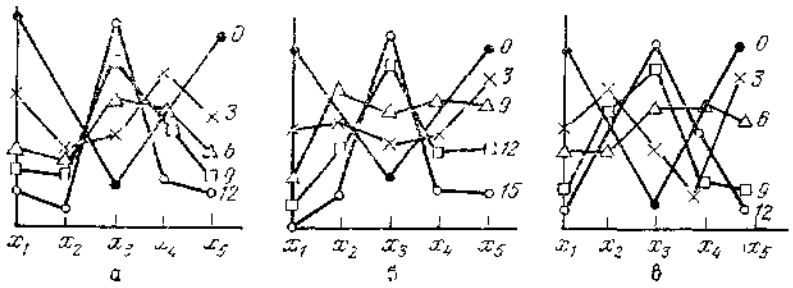


Рис. 5.5. Изменение оценок априорной нечеткой информации: а — в случае четкого подтверждения; б — в случае нечеткой информации подтверждения; в —в случае нечеткого попеременного подтверждения

На рис. 5.5, *в* на вход поступает переменно два различных множества: $h_{A_1} = (0{,}3, 0{,}8, 0{,}6, 0{,}2, 0{,}1)$ и $h_{A_2} = (0{,}1, 0{,}2, 0{,}6, 0{,}8, 0{,}3)$. В случае рис. 5.5, *а* на вход поступает $h_A = (0, 0, 1, 0, 0)$, а в случае рис. 5.5, *б* на входе $h_A = (0{,}3, 0{,}5, 0{,}8, 0{,}5, 0{,}3)$. На рисунке числа указывают номер итерации.

Обучающаяся модель достаточпо хорошо работает с нечеткой информацией, но скорость сходимости в этом случае меньше.

Пример. Рассмотрим модель глобального поиска экстремума неизвестной функции с несколькими локальными экстремумами. Для поиска глобального экстремума формируются критерии в виде некоторых функций:

x_1 — оценивает число точек, проанализированных на предыдущих шагах;

x_2 — оценивает среднее значение функции по результатам предыдущих шагов;

x_3 — оценивает число точек, значение функции в которых принадлежит десятке лучших в своей области;

x_4 — оценивает максимум по прошлым попыткам;

x_5— оценивает градиент функции.

В описанном случае g_X показывает степень важности подмножеств критериев и $\sigma_Y(\{y_j\}|x_i)$ оценивает предположение о нахождении экстремума в блоке y_j в соответствии с критерием x_i. Например, $\sigma_Y(\{y_j\}|x_i)$ может зависеть от числа ранее проанализированных точек в блоке y_j. Пусть входная информация A определяется формулой

$$\mu_A(y_j) = \frac{p_j - \min\limits_k p_k}{\max\limits_k p_k - \min\limits_k p_k},$$

где p_k — максимум анализируемой функции, найденный к рассматриваемому моменту в блоке y_j. Очевидно, что A сходится к максимизирующему множеству функции. На каждой итерации осуществляется следующее: проверяется заданное число новых точек; число этих точек выбирается пропорционально $g_Y(\{y_j\})$; в~каждой точке y_j вычисляется и нормализуется мера $\sigma_Y(\cdot|x_i$); нормализуется g_X ; по σ_Y и σ_X вычисляется $g_Y(\{y_j\})$, а затем $g_Y(A)$; посредством правил подкрепления корректируется $g_Y(\{x_i\})$. Затем выполняется новая итерация, и так до тех пор, пока не сойдется g_Y.

5.4.3. Адаптивный нечеткий логический регулятор

Нечеткий логический регулятор как простейший нечеткий алгоритм подробно описан в 5.5. Здесь опишем только способ уточнения правил управления, используемых в адаптивном нечетком логическом регуляторе (АНЛР). Ниже описан АНЛР для управляемых технологических процессов с одним входом и одним выходом. Соответствующая схема регулятора приведена на рис.5.6.

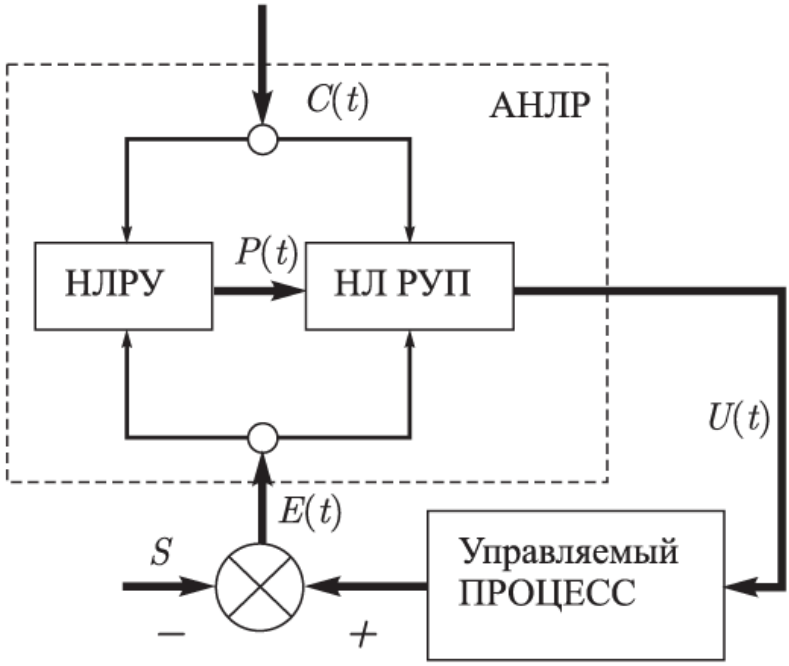


Рис. 5.6. Адаптивный нечеткий логический регулятор

АНЛР состоит из двух частей: нечеткого логического регулятора управляемого процесса (НЛРУП) и нечеткого логического регулятора управления (НЛРУ). На рис. 5.6 используются следующие обозначения:

u(t)—управление, генерируемое НЛРУП,

e(t)—ошибка (отклонение от устанавливаемого выходного значения процесса s),

s — желаемое значепие выхода управляемого процесса,

$$c(t) = e(t) - e(t-1);$$

p(t) —модификация управления.

Правила НЛРУП имеют форму: **if** $e = E_i$ **then if** $c = C_i$ **then** $u = U_i$.

Правила НЛРУ имеют форму: **if** $e = E_i$ **then if** $c = = C_j$ **then** $p = P_j$.

Здесь E_i, E_j, C_i, C_j, U_i, P_j— предварительно описанные нечеткие множества. Символ *p(t)* используется для модификации стратегии управления следующим образом: в нечетком правиле *i*, которое ухудшает течение процесса, заменяется значение управления U_i на $U_i' = U_i \otimes p_i(t)$. Правило *i* в НЛРУП заменяется на правило **if** $e = E_i$ **then if** $c = C_i$ **then** $u = U'_i$

Рассмотрим далее два алгоритма обучения при лингвистическом описании предпочтений: алгоритм формирования нечеткого отношения предпочтения на множестве альтернатив, описываемых наборами лингвистических значений признаков и алгоритм уточнения лингвистических критериев.

5.4.4. Алгоритм формирования нечеткого отношения предпочтения.

Пусть R — множество таких альтерпатив, что каждое $S \in R$ характеризуется набором оценок по n признакам: $S = (t_1, \ldots, t_n)$, и пусть B — семейство всех непустых конечных подмножеств множества R. Для некоторого $R' \in B$ известно подмножество выбранных альтернатив $R'' \subset R'$, т. е. для любых $S^n \in R^n$ и $S' \in R' \backslash R''$ имеет место доминирование $S'' \succ S'$. Предварительно, при анализе исходного множества альтерпатив, сформирован эталонпый набор нечетких оценок $A^0 = (t_1^0, \ldots, t_n^0)$. Значения функции принадлежности нечеткой оценки t^0_i указывают на степень близости значений i-го признака к значениям, определяющим идеальную альтернативу. Используя множество предпочтений

$$E = \{(S'', S') : S'' \in R'', S' \in R' \backslash R''\},$$

требуется найти обобщенные правила предпочтения на множестве R. Поясним на примере работу алгоритма, в основе которого лежит модель распознавания классов М. М. Бонгарда.

Пример. Рассмотрим задачу выбора для добывающего судна рациональпого района промысла с учетом следующих показателей: u_1 — время перехода, u_2 — прогпоз вылова, u_3 — стоимостная характеристика прогпозируемого объекта лова, u_k— гидрометеоусловия. Показатели, в сущности, играют роль лингвистических переменных, лингвистические значения которых, в том числе эталонные значения, приведены соответственно на рис. 5.7, *а, б, в, г* (численные значения базовых переменных даются условно).

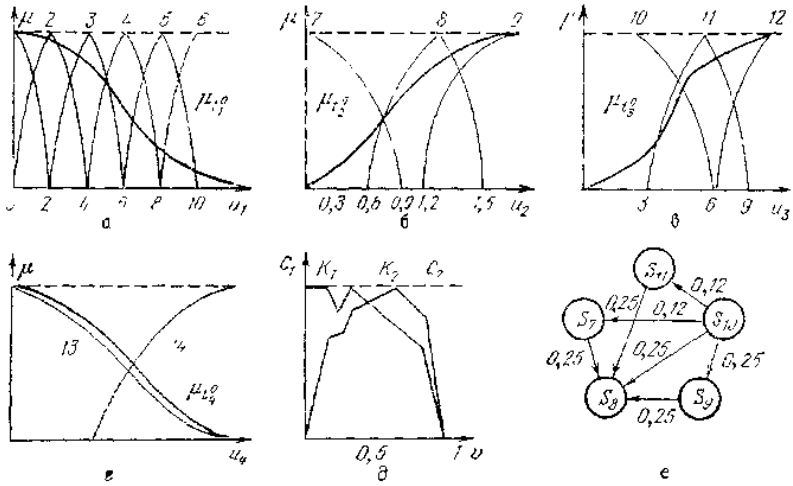


Рис. 5.7. Формирование нечеткого отношения предпочтения для задачи выбора рациональных районов промысла: а — терм-множество переменной «время перехода в сутках»; б — терм-множество неременной «прогноз вылова в тоннах»; в — терм-множество переменной «стоимость объекта лова в тыс. руб.»; г — терм-множество переменной «гидрометеоусловия в баллах»; д — значения нечетких логических признаков на интервале значений истинности; е — нечеткий граф предпочтения альтернатив

Цифрами па рис. 5.7 обозначаются следующие названия термов:
1 — очень хорошее, 2 — хорошее, 3 — нормальное, 4 — удовлетворительное, 5 — плохое, 6 — очень плохое, 7 — плохой, 8 — нормальпый, 9 — хороший, 10 — плохая, 11 — пормальная, 12 — хорошая, 13 — удовлетворительная, 14 — неудовлетворительная. Лицу, принимающему решепия, предложены альтернативы S_1 — S_6, (см. табл. 5.1). Пусть выбрана альтернатива S_1. Для обучения формируются две таблицы:

$$\mathcal{H}_1 = \{(S_1, S_2),\ (S_1, S_3),\ (S_1, S_4),\ (S_1, S_5),\ (S_1, S_6)\},$$
$$\mathcal{H}_2 = \{(S_2, S_1),\ (S_3, S_1),\ (S_4, S_1),\ (S_5, S_1),\ (S_6, S_1)\}.$$

Для каждой пары паборов (S_i, S_j) вычисляются оценки сравнения i-го элемента первого набора с i-м элементом второго набора:

$$\left.\begin{matrix}(t_1', \ldots, t_n')\\ (t_1'', \ldots, t_n'')\end{matrix}\right| \to (L^{\alpha}(t_1', t_1''), \ldots, L^{\alpha}(t_n', t_n'')),$$

где α определяет конкретный оператор, например, нечеткую меру сходства

$$L^{\alpha} = \mathcal{V}^{\alpha}(t^1 \to t^2) = \bigcup_i (\mu_{t^2}(u_i) \mid \mu_{t^1}(u_i)).$$

В результате получаются две таблицы наборов нечетких оценок поэлементного сравнения. На основе полученных таблиц, используя логические операторы и логические функции двух переменных, выделяются полезные логические признаки и минимальный базис, объединение значений истинности которого на строках первой и второй таблиц приведены на рис. 5.7, *д*. Содержательное значение утверждения, соответствующего минимальному базису, следующее:

$$\Phi_5(S_i, S_j) \succ \Phi_5(S_j, S_i) = (x_1^i \succ x_1^j) \,\&\, (x_2^i \succ x_2^j) \,\&\, (x_4^i \succ x_4^j) \succ$$
$$\succ (x_1^i \prec x_1^j) \,\&\, (x_2^i \prec x_2^j) \,\&\, (x_4^i \prec x_4^j), \quad x_k^m = \mathcal{V}^{\alpha}(t_k^m \to t_k^0),$$

t^m_k — лингвистическое значение *k*-го показателя, Φ_5 — логический признак. Физический смысл приведенного утверждения:
район S_i предпочтительнее района S_j, если утверждение [(время перехода до S_i , «меньше», чем до S_j) и (прогноз вылова в S_i «больше», чем в S_j) и (погодные условия в S_i «лучше», чем в S_j)] более истинно, чем обратное утверждение [(время перехода до S_i «больше», чем до S_j) и (прогноз вылова в S_i «меньше», чем в S_j) и (погодные условия в S_i «хуже», чем в S_j)].
Далее предположим, что среди неизвестных ситуаций S_7—S_{11} (табл. 5.3) необходимо выбрать лучшую альтернативу, используя минимальный базис. В табл. 5.4 изображена матрица предпочтений $\mathbf{M} = (\mu^{ij}(K_1) \mid \mu^{ij}(K_2))$, элементы которой вычислялись посредством гарантированной оценки

$$\mu^{ij}(K_1) = \max_{v \in [0,1]} \mu_{H_1(S_i, S_j)}(v),$$

где

$$H_i(S_1, S_2) = \bigcap (C_j^i \bigcap C_j(S_1, S_2)), \quad C_j(S_1, S_2)$$

— значение *j*-го логического признака на паре альтернатив (S_1,S_2), C^i_j— значение *j*-го признака на парах альтернатив *i*-го класса (i = 1, 2).

Таблица 5.3

	U_1	U_2	U_3	U_4		U_1	U_2	U_3	U_4
S_1	хор	хор.	хор.	уд.	S_7	плох.	хор.	плох.	уд.
S_2	оч. хор.	плох.	хор.	уд.	S_8	уд.	хор.	хор.	неуд.
S_3	оч. хор.	хор.	хор.	неуд.	S_9	плох.	хор.	хор.	уд.
S_4	уд.	хор.	хор.	уд.	S_{10}	уд.	хор.	норм.	уд.
S_5	оч. плох.	хор.	хор.	уд.	S_{11}	уд.	норм.	норм.	уд.
S_6	хор.	норм.	плох	уд.					

Таблица 5.4

S_i	S_j				
	S_7	S_8	S_9	S_{10}	S_{11}
S_7		0,88 0,38	1 0,38	0,88 0,38	0,88 0,38
S_8	0,75 1		0,75 1	0,75 1	0,75 1
S_9	1 0,38	0,88 0,38		0,88 0,38	0,88 0,38
S_{10}	1 0,38	1 0,38	1 0,38		1 0,38
S_{11}	0,88 0,38	0,88 0,38	0,88 0,38	0,88 0,38	

Каждый элемент матрицы содержит два значения. Верхнее значение указывает степень, с которой S_i доминирует над S_j. Нижнее значение указывает степень, с которой S_j доминирует над S_i. Для построения печеткого графа предпочтений альтернатив (рис. 5.7, е), используется следующее правило определения отношения доминирования *D:*

$$D(S_i, S_j) = \begin{cases} S_i \overset{\Delta}{\succ} S_j, & \text{если} \quad \mu_1 \geqslant \mu_2; \\ S_j \overset{\Delta}{\succ} S_i, & \text{если} \quad \mu_1 \leqslant \mu_2; \end{cases}$$

где $\mu_1 = \mu^{ij}(k_1) \vee \mu^{ji}(k_2), \quad \mu_2 = \mu^{ij}(k_2) \vee \mu^{ji}(k_1), \quad \Delta = |\mu_1 - \mu_2|.$

Согласно рис. 5.7, *е* S_{10} есть недоминируемая альтернатива, т. е. не существует альтернативы, которая с ненулевой степенью доминирует над S_{10}.

5.4.5. Алгоритм уточнения лингвистических критериев.

Глобальные представления ЛИР о выборе альтернатив формулируются в виде глобального критерия и решение многокритериальной задачи сводится к построению композиции $M_1 \circ M_2 = M$, где

$$M_1\colon \mathcal{F}(U^n)\to\mathcal{F}(Q^m),\quad U^n=U_1\times\ldots\times U_n,\quad Q^m=Q_1\times\ldots\times Q_m,$$
$$M_2\colon \mathcal{F}(Q^m)\to\mathcal{F}(Q),\quad U_i\ (i=1,\ n),\quad Q_j\ (j=1,\ m),$$

Q — множестиа значений признаков, локальных и глобального критериев. M_1 и M_2 формируются на основе высказываний типа: «если значения признаков u_1 , ..., u_n, характеризующие альтернативу u^i, оцениваются термами t_{1i}, ..., t_{ni}, то альтернатива удовлетворяет j-му критерию с оценкой $t_{n+j,i}$». M_1 и M_2 описываются наборами:

$$M_1=\{(t_{1i},\ \ldots,\ t_{ni},\ t_{n+j,\,i})\,|\,n+1\leqslant j\leqslant n+k,\ i=1,\ m_1\},$$
$$M_2=\{(t_{n+1,\,i},\ \ldots,\ t_{ni},\ t_{n+k+1,\,i}),\ i=1,\ \ldots,\ m_2\}.$$

Степень удовлетворения глобальному критерию для альтернативы $u^i\in U^n$ вычисляется следующим образом:

$$w(u^i)=\bigcup_{t^p\in M_2}\left(\bigcup_{t^e\in M_1}u^i\circ t^e\right)\circ t^p.$$

В процессе обучения уточняются оценки локальных и глобального критериев на основе сравнения выбранных ЛПР альтернатив R'' из множества предъявленных $R'\supset R''$. M заменяется некоторым $\widetilde{M}$, подтверждающим соответствующий выбор:

$$\widetilde{w}(u^i)\prec\widetilde{w}(u^j)\ \text{ для }\ u^i\in R',\quad u^j\in R'\backslash R''.$$

Обучение осуществляется в два этапа: формирование обобщенных описаний предпочтений ЛПР (нечеткая модель М. М. Бонгарда); модификация M при несовпадении предпочтений ЛПР с порядком оценок $w(u)$. На втором этапе выполняется следующее: генерация допустимых наборов оценок показателей; определение отношения предпочтения на парах сгенерированных альтерпатив, выделение из $M=M_1\cup M_2$ наборов, не подтверждающих выявленные предпочтения и, следовательно, подлежащих корректировке; корректировка оценок по критериям.

Пример. Пусть показатели u_1, u_2 и лингвистические критерии Q_1, Q_2, Q оцениваются терм-множеством {1 — «хорошо», 2—«норма», 3— «плохо»}, функции принадлежности которого изображены на рис. 5.8.

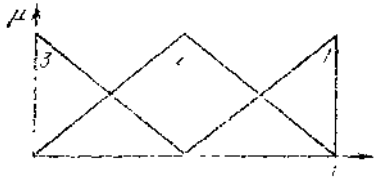
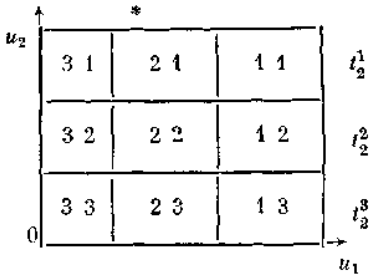


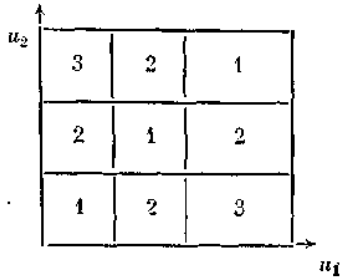
Рис. 5.8. Функции принадлежности терм-множества лингвистических критериев

Априорные сведения о критериях сформулированы в виде наборов оценок (термов), характеризующих степень удовлетворения критериям. В табл. 5.5 приведены наборы номеров термов.

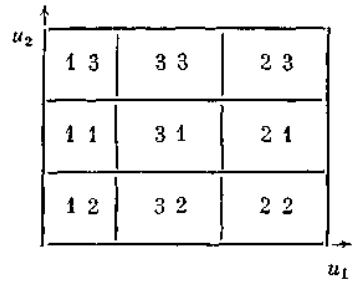
Таблица 5.5



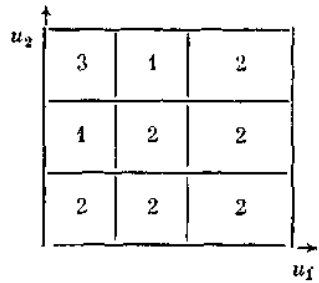
Номера оценок локальных критериев



Номера оценок глобального критерия



Номера оценок локальных критериев после корректировки



Номера оценок глобального критерия после корректировки

Анализируемые альтернативы описываются значениями

$$(u_1, u_2)\colon R' = \{(1, 0),\ (0{,}5,\ 1),\ (1,\ 1),\ (0,\ 1),\ (1,\ 0{,}5),\ (0{,}5,\ 0{,}5)\},$$
$$R'' = \{(0{,}5,\ 1)\}.$$

Альтернатива $u \in R''$ отмечена в табл. 5.5 звездочкой. В результате обучения формируется обобщенное описание предпочтений ЛПР: $D(u^i, u^j)$: $(t_1^i \succ t_1^j)\ \&\ \neg\,(t_2^i \succ t_2^j)$. На основании $D(u^i u^j)$ сформировано уточненное $\bar{M}$, которое приведено на рис. 5.9, *в, г.*

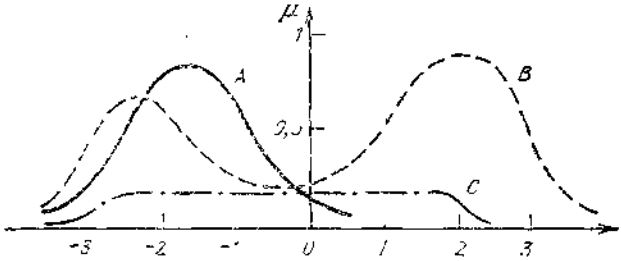


Рис. 5.9. Возможные оценки управляющего воздействия

5.5. Описание простейших нечетких алгоритмов

Простейшие нечеткие алгоритмы являются частным видом алгоритмов, определенных в 5.1. Для простейших нечетких алгоритмов рассматринаются функции входа и выхода и не используются функции переходов и операций. Этот тип алгоритмов получил широкое распространение при формализации опыта человека-оператора, управляющего технологическим процессом. Такие алгоритмы называются **нечеткими логическими регуляторами.** Разработан подход, направленный на формализацию опыта оператора, управляющего некоторым объектом. Определенные промышленные установки могут лучше управляться опытными операторами, чем обычными автоматическими регуляторами. Стратегия управления, используемая оператором, часто может быть сформулирована как набор правил, которые просто выполнить вручную, но трудно формализовать, используя обычные алгоритмы. Эта трудность возникает из-за того, что человек чаще использует качественные, а но количественные оценки при описании условий принятия конкретных решений. Следовательно, для моделирования управления такими процессами необходимо использовать нечеткую логику. В литературе описывается язык программирования, удобный для описания такого класса процессов. Приведем основные идеи этого подхода. Пусть качественная оценка *A* значения параметра, описывающего состояние процесса, формализована нечетким подмножеством *A* множества значений параметра *U,* а качественные оценки *B, C* значений управляющего воздействия формализованы нечеткими подмножествами *B, C* множества воздействий *V.* Тогда алгоритм управлепия будет строиться из правил следующего вида:

$$\text{if } A_i \text{ then } B_i \text{ else } C_i,$$

где каждое такое выражение задает отношение $R_i = (A_i \times B_i) \cup \neg(B_i \times C_i)$ в пространстве $U \times V$, а $\cup$, $\neg$, $\times$ — операции объединения, отрицания и декартова произведения соответственно.

Для заданного значения *u* каждое правило позволяет определить нечеткое множество воздействий $C_i' = u \circ R_i$. Нечеткое управляющее воздействие определяется как объединение $C' = \bigcup_{i=1}^{k} C_i'$. Для определения единственного управляющего воздействия выбирается значение с максимальной оценкой или, если имеется плато у функции принадлежности, то выбирается значение в центре плато и т. д.

Результаты применения полученного нечеткого алгоритма управления показаны на рис. 5.9 , где используются следующие обозначения:

— одно доминирующее правило;

-- — -- два противоречивых правила;

— · — отсутствие удовлетворительных правил.

Вид кривых может быть использован для оценки качества правил управления. Функция припадлежности *A* указывает, что для исследуемой области имеется одно доминирующее правило управления. Функция припадлежности *C* указывает, что не имеется подходящих правил для оценки управления. Функция принадлежности *B* указывает, что имеется, по крайней мере, два противоречивых правила. В двух последних случаях необходима модификация правил для того чтобы получить хороший алгоритм управления.

Наиболее широкое применение при решении практических задач получили нечеткие логические регуляторы, которые позволяют на основании лингвистической информации, полученной от опытного оператора, управлять сложными, плохо формализованными процессами.

Структура нечеткого логического регулятора, в котором используются эвристические правила принятия решений, показана на рис. 5.10

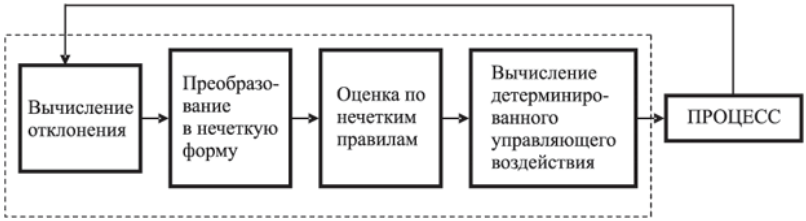


Рис.5.10.

Такие регуляторы применяются аналогично традиционным регуляторам с обратной связью. Определение управляющих воздействий состоит из четырех основных этапов:

1. Получение отклика;
2. Преобразование значения отклонения к нечеткому виду, такому, как "большой", "средний";
3. Оценка входного значения по заранее сформулированным правилам принятия решения с помощью композиционного правила вывода;
4. Вычисление детерминированного выхода, необходимого для регулирования процесса.

Опишем способ уточнения правил управления, используемых в адаптивном нечетком логическом регуляторе (АНЛР). Соответствующая схема регулятора приведена на рис. 5.6 АНЛР состоит из двух частей: нечеткого логического регулятора управляемого процесса (НЛРУП) и нечеткого логического регулятора управления (НЛРУ). На рис. 5. 6 используются следующие обозначения:

$U(t)$ — управление, генерируемое НЛРУП;

$E(t)$ — ошибка (отклонение от устанавливаемого выходного значения процесса S);

S — желаемое значение выхода управляемого процесса,
$C(t) = E(t) - E(t-1)$;

$P(t)$ — модификация управления.

Правила НЛРУП имеют форму: if $E = E_i$ then if $C = C_i$ then $U = U_i$.

Правила НЛРУ имеют форму: if $E = E_j$ then if $C = C_j$ then $P = P_j$.

Здесь E_i, E_j, C_i, C_j, U_i, P_j — предварительно описанные нечеткие множества. Символ $P(t)$ используется для модификации стратегии управления следующим образом: в нечетком правиле i, которое ухудшает течение процесса, заменяется значение управления U на $U_i' = U_i \otimes p_i(t)$. Правило i в НЛРУП заменяется на правило if $E = E_i$ then if $C = C_i$ then $U = U_i'$.

Описываемый здесь подход значительно расширяет сферу взаимодействия человек — машина посредством формализации нечетких алгоритмов. Далее будет рассмотрено несколько примеров практического применения нечеткого логического регулятора

5.5.1. Нечеткий логический регулятор процесса теплообмена.

При создании нечеткого логического регулятора нет необходимости в создании точной математической модели. Достаточно приблизительного представления о соотношении входных и выходных переменных, описывающих процесс.
Описание процесса теплообмена. Горячая вода, циркулирующая по замкнутому трубопроводу под воздействием электронасоса, используется для подогрева холодной воды. Скорость потока горячей

воды F_H является одним из двух входов нечеткого логического регулятора. Вторым входом является мощность, затрачиваемая на подогрев горячей: воды. Величина потребляемой мощности, поставляемой генератором, регулируется посредством усилителя и шагового мотора.

Скорость потока холодной воды *(F_c),* текущей из обычного водопроводного крана и подогреваемой в теплообменнике в результате теплообмена с горячей водой, регулируется вручную клапаном. Скорость потока холодной воды не регулируется автоматически, поэтому она рассматривается как внешнее возмущение в управляемом процессе.

Задача регулирования состояла в следующем: на основе значений скорости потока горячей воды F_H и потребляемой мощности W необходимо регулировать выходную температуру холодной воды T_{co} и входную температуру горячей воды T_{Hl} таким образом, чтобы они имели значения T_{cos} и T_{his} соответственно.

Задачу усложняет существенная зависимость обеих температур от мощности, а также существенная нелинейность процесса. Схема процесса теплообмена приведена па рис.5**.**11**.**

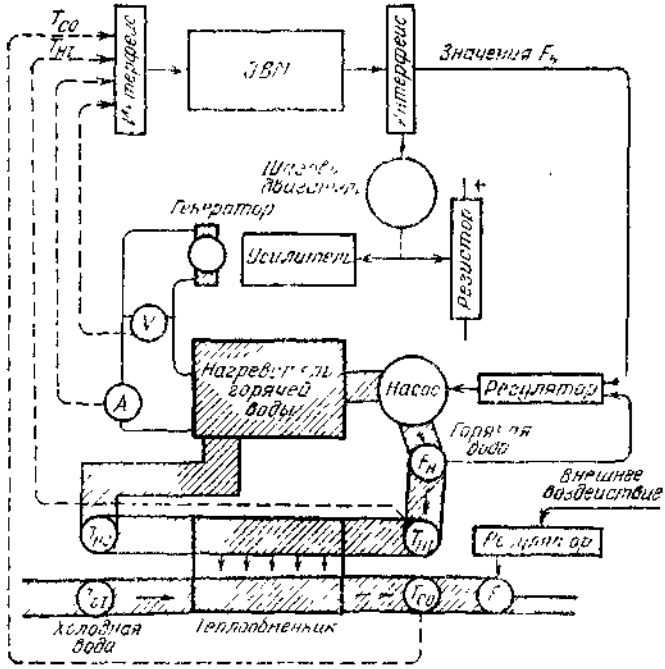


Рис. 5.11. Схема процесса теплообмена (пунктирные стрелки от *A* и *V* означают подачу сигналов тока и напряжения, а от T_{HI} и T_{CO} — сигналов значений температуры)

Выведем основные соотношения перечисленных характеристик процесса.

Связь между переменными $F_н$, W, $T_{со}$ и $T_{нI}$ имеет следующий вид:

$$W = c_v F_н (T_{нI} - T_{но}),$$

$$W = c_k F_к (T_{со} - T_{сI}),$$

где c_v, c_k — теплоемкости горячей и холодной воды. В этих выражениях предполагается, что вся энергия, получаемая от генератора, передается холодной воде. Следовательно,

$$T_{со} = T_{сI} + \frac{W}{c_k F_с},$$

т. е. температура $T_{со}$ зависит только от одной входной переменной *W*. Для того чтобы найти T_{HI} как функцию от входных переменных *W* и $F_н$, необходимо использовать зависимость от температуры

передачи тепла от одной жидкости к другой через некоторую поверхность:

$$W = AU/T_m,$$

где A — коэффициент, зависящий от конфигурации поверхности, U — коэффициент теплопроводности, T_m — средняя разность температур жидкостей. Приближенное выражение для потребляемой мощности с учетом передачи тепла через поверхность

$$W = AU\,\frac{T_{HI} + T_{HO} - T_{CI} - T_{CO}}{2}$$

и, следовательно, для выходной температуры воды

$$T_{HO} = \frac{2W}{AU} + T_{CI} + T_{CO} - T_{HI}$$

получаем приближенную зависимость:

$$T_{HO} = \frac{2W}{AU} + 2T_{CI} - T_{HI} + \frac{W}{c_h F_C},$$

которая дает следующее выражение для T_{HI}

$$T_{HI} = T_{CI} + \left(\frac{1}{2c_V F_H} + \frac{1}{2c_h F_C} + \frac{1}{AU}\right) W.$$

Выходная переменная T_{CO} зависит только от одной входной переменной W. Температура, же горячей воды T_{HI} существенно зависит как от потребляемой мощности W, так и от скорости потока горячей воды F_H . Эти факты указывают на то, чго наиболее простой способ решении поставленной выше задачи следующий: осуществить регулирование T_{CO} посредством W, затем попытаться регулировать T_{HI} посредством F_H. Следовательно, стратегия управления фактически состоит из двух достаточно независимых этапов.

Описание нечеткого логического регулятора процесса теплообмена. Для рассмотренною выше процесса теплообмена был разработан нечеткий алгоритм принятия решения, основанный на использовании отклонений T_{CO} и T_{HI} от установленных значений температур T_{COE} и T_{HIE} , а также на использовании изменения указанных отклонений T_{COE} , T_{HIE}, которые обозначаются далее T_{COEC} и T_{HIEC}.

Алгоритм предназначен для регулирования посредством W и F_H температуры потока холодной воды после теплообменника и температуры потока горячей воды перед теплообменником, таким образом, чтобы они находились вблизи установленных значений. Другими словами, необходимо регулировать мощность W и скорость потока горячей воды на основе переменных T_{COE} , T_{HIE}, T_{COEC} , T_{HIEC}. Значения переменных были описаны посредством нечетких

подмножеств, функции принадлежности которых приведены в табл. 5.6.

Таблица 5.6

Нечеткое подмножество	Формула
Большое положительное x	$1-\exp\left[-\left(\frac{0,5}{\mathrm{abs}\,(1-x)}\right)^{2,5}\right]$
Среднее положительное x	$1-\exp\left[-\left(\frac{0,25}{\mathrm{abs}\,(0,7-x)}\right)^{2,5}\right]$
Малое положительное x	$1-\exp\left[-\left(\frac{0,25}{\mathrm{abs}\,(0,4-x)}\right)^{2,5}\right]$
Близкое к нулю положительное x	$\exp\left[-5\,\mathrm{abs}\,(x-0,05)\right]$
Близкое к нулю отрицательное x	$\exp\left[-5\,\mathrm{abs}\,(x+0,05)\right]$
Малое отрицательное x	$1-\exp\left[\left(\frac{0,25}{\mathrm{abs}\,(-0,4-x)}\right)^{2,5}\right]$
Среднее отрицательное x	$1-\exp\left[-\left(\frac{0,25}{\mathrm{abs}\,(-0,7-x)}\right)^{2,5}\right]$
Большое отрицательное x	$1-\exp\left[-\left(\frac{0,5}{\mathrm{abs}\,(-1-x)}\right)^{2,5}\right]$

Нечеткие множества приведены на универсуме [—1, 1], т. е. значения переменных нормированы. Нечеткие значения выбирались следующим образом. Переменные T_{COE}, T_{HIE} считаются «большими положительными», если они больше чем 5°С, «средними положительными», если имеют место значения около 0,25°С. Для отрицательных значений характерные точки выбирались симметричными. Например, измеряемые значения T_{COE} лежат в интервале [—5, 5] и, следовательно, разделив граничные значения на 5, получии функцию принадлежности, приведенную в табл. 5.6. После введения нечетких значений формулируются словесные (лингвистические) правила, описывающие стратегию управления. Эти правила формулируются в виде условных предложений, не содержащих количественных значений.

Рассмотрим первоначально схему регулирования температуры холодной воды *Tco*. Регулировать температуру *Tco* разумно посредством изменения главным образом потребляемой мощности *W*. Увеличение *W* приводит к увеличению температуры *Tco,* следовательно, если имеет место положительное изменение мощности

W_{CH}, то имеет место положительное отклонение температуры T_{COE}. Аналогично, если имеет место отрицательное изменение W_{CH}, то и T_{COE} — отрицательное. Величина положительного или отрицательного изменения W_{CH} зависит от величины положительного или отрицательного отклонения T_{COE}, например, W_{CH} — большое положительное, если T_{COE} — большое положительное отклонение.
Для улучшения качества регулирования рассматривается изменение T_{COEC} отклонения T_{COE} между последовательными замерами. В этом случае регулирование осуществляется следующим образом. Если T_{CO} близка к установленной величине T_{COS} и в то же время достаточно быстро изменяется, так что T_{COEC} является большим или средним положительным, то разумно остановить этот быстрый процесс средним отрицательным изменением W. Эта ситуация возникает в случае, когда T_{CO} приближается достаточно быстро к значению T_{COS} от значения T_{CO}, которое меньше T_{COS}. В противоположном случае, если T_{COES} — большое и среднее отрицательное и T_{CO} снова близко к T_{COS}, то потребляемую мощность W необходимо подвергнуть среднему положительному изменению.
Используя описанное множество правил, можно увеличивать или уменьшать мощность W так, что температура T_{CO} будет достаточно быстро приближаться к требуемым значениям.
При регулировании температуры T_{CO} используется не только величина потребляемой мощности, но также и скорость потока горячей воды F_H. В устойчивом состоянии значения T_{CO} не зависят от F_H, но при изменении состояния выходная температура холодной воды T_{CO} увеличивается или уменьшается в зависимости от увеличении или уменьшения скорости потока горячей воды F_H. Поэтому, чтобы сократить время стабилизации процесса, скорость F_H подвергается среднему положительному изменению F_{HC}, если T_{COE} является большим или средним положительным. Эта регулировка скорости F_H оказывается разумной только в случае, если мы потребуем, чтобы отклонение T_{HIE} не было большим или средним положительным. В противоположном случае, когда значения T_{COE} являются большими или средними отрицательными и значения T_{HIE} не являются большими или средними отрицательными, скорость потока горячей воды F_H подвергается средним отрицательным изменениям F_{HC}.
Аналогично формулируется стратегия регулирования температуры потока горячей воды T_{HI}.
Алгоритм управления процессом теплообмена был написан на алгоритмическом языке APL, утверждения алгоритма на котором имеют следующий вид:

T_1 = (большое положительное отклонение T_{COE}) или (среднее положительное T_{COE}) ;
T_2 = (большое положительное T_{HIE}) или (среднее положительное T_{HIE}) ;
T_{HC} = если (T_1 и не T_2), то среднее положительное иначе F_{HC} .
Описанный алгоритм был использован для управления установкой теплообмена и показал результаты, не уступающие по основным характеристикам (быстродействие по возвращению в устойчивое состояние после возмущения, стабилизация для различных заданных значений входных и выходных температур) известным регуляторам для такого класса устройств.

5.5.2. Управление паровой машиной .

Нечеткий алгоритм используется для управления лабораторной паровой машиной. Множество правил, выраженных в виде нечетких условных утверждений, интерпретируется на ЭВМ для того чтобы осуществлять автоматизированное управление установкой.
В алгоритме нечеткого логического регулятора используются следующие четыре переменных, описывающих управляемый процесс:
P_E — отклонение давления в паровом котле, определенное как разность между текущим значением и выбранным заранее значением, соответствующим норме;
S_E — скорость изменения P_E;
C_{PE} — изменение отклонения давления, определяемое как разность между текущим давлением P_E и значением давления, полученным в предыдущем измерении;
C_{SE} — изменение скорости отклонения C_{PE}.
Регулирование осуществляется по двум алгоритмам: по одному корректируется степень подогрева пара, т. е. регулируется давление (H_c— изменение подогрева), по другому изменяется положение дросселя (T_c — изменение положения дросселя). В каждом алгоритме учитываются все приведенные выше переменные. Лингвистические правила, описывающие алгоритм управления, определяются заранее опытным человеком-оператором.
Переменная H_c представлена дискретным набором значений из 32 точек, а T_c — десятью точками. Переменные C_{PE}, P_E, C_{SE} и S_E представлялись 13 точками, равномерно распределенными между максимальными положительными и отрицательными значениями.
Для описания значении переменных человеком-оператором использовались следующие лингвистические значения (сокращенные названия соответствуют первым буквам слов на английской языке): PB — большое положительное, PM — среднее положительное, PS —

малое положительное, NO — нулевое, NS — малое отрицательное, NM — среднее отрицательное, NB — большое отрицательное.
Для переменных P_E и S_E дополнительно выделены отрицательные близкие к нулю значения (ниже нормы — NO) и положительные близкие к нулю значения (выше нормы — PO). Наряду с указанными подмножествами для оценки значений переменных использовалось нечеткое значение ANY, описываемое такой функцией принадлежности, которая равна единице для любого элемента (значения). Сложные значения на основе указанных получались посредством операции И, ИЛИ, НЕ, которые интерпретируются как min, max, вычитание из единицы. Правила управления формулировались в виде условных предложений, например, «если $P_E = NB$, то $H_c = P_E$ ». Приведенное условное предложение задает отношения между двумя нечеткими переменными P_E, H_c, которое описывается декартовым произведением двух нечетких модмножеств *NB* и *PB:* NB × PB. Декартово произведение удобно представлять матрицей из *n* столбцов и *m* строк, где *n* и *m* — число элементов универсумов для подмножеств *NB* и *PB,*
Предположим, что известно отношение *R* между перемеппыми P_E и H_c, тогда для некоторого значения можно определить выходное значение посредством правила композиции $y = x \circ R$. Для условных выражений «если *A,* то (если *B,* то С)» определяется декартово произведение $A \times B \times C$, которое используется для определения выхода *C* при входах *A′* и *B′:*

$$C' = (A' \times B') \circ (A \times B \times C).$$

В описываемом алгоритме два или более правил комбинировались при помощи связки ИНАЧЕ, которая интерпретировалась как операция max.
Например, «если $P_E = \text{NB}$ и $C_{PE} = \text{НЕ (NB или NM)}$ и $S_E = ANY$, то H_c = PM, иначе, если $P_E = \text{NB}$ и $C_{PE} = \text{NC}$ и $S_E = ANY$ и $C_{SE} = ANY$, то $H_C = PM$, иначе, если...».
Оба алгоритма, алгоритм управления давлением и алгоритм управления скоростью (дросселем), приведены в виде сложных условных выражений. Приведем их полное описание.
Алгоритм управления давлением:

«если P_E = NB, то (если C_{PE} = НЕ (NB или NM), то H_C = PB);
если P_E = (NB или *NM*), то (если C_{PE} = NC, то H_C = PM);
если P_E = *NS*, то (если C_{PE} = PS или NO, то H_C = PM);
если P_E = NO, то (если C_{PE} = (PB или PM), то H_C = PM);
если P_E = NO, то (если C_{PE} = (NB или *NM*), то H_C = NM);
если P_E = (PO или NO), то (если C_{PE} = NO, то H_C = NO);
если P_E = PO, то (если C_{PE} = (NB или NM), то H_C = PM);
если P_E = PO, то (если C_{PE} = (PB или PM), то H_C = NM);
если P_E = PS, то (если C_{PE} = (PS или NO), то H_C = NM);
если P_E = (PB или PM), то (если C_{PE} = NS, то H_C = NM);
если P_E = PB, то (если C_{PE} = НЕ (NB или NM), то H_C = NB);
если P_E = NO, то (если C_{PE} = PS, то H_C = PS);
если P_E = NO, то (если C_{IE} = NS, то H_C = NS);
если P_E = PO, то (если C_{PE} = NS, то H_C = PS);
если P_E = PO, то (если C_{PE} = PS, то H_C = NS)».

Алгоритм управления скоростью:
«если S_E = NB, то (если C_{SE} = НЕ (NB или NM), то T_C = PB);
если S_E = NM, то (если C_{SE} = (PB или PM или PS), то T_C = = PS);
если S_E = NS, то (если C_{SE} = (PB или PM), то T_C = PS);
если S_E = NO, то (если C_{SE} = PB, то T_C = PS);
если S_E = (PO или NO), то (если C_{SE} = (PS или NS или NO), то T_C = NO);
если S_E = PO, то (если C_{SE} = PB, то T_C = NS);
если S_E = PS, то (если C_{SE} = (PB или PM), то T_C = NS);
если S_E = PM, то (если C_{SE} = (PB или PS или PM), то T_C = NS);
если S_E = PB, то (если C_{SE} = НЕ (NB или NM), то T_C = NB)».

5.5.3. Управление процессом подогрева воды.

Опишем нечеткий алгоритм, построенный на основе опыта человека-оператора, позволяющий управлять установкой подогрева воды, управление которой обычными способами затруднено из-за нелинейности и изменчивости процесса.

Управляемый процесс подогрева воды. На рис. 5.12 схематически показана установка подогрева воды.

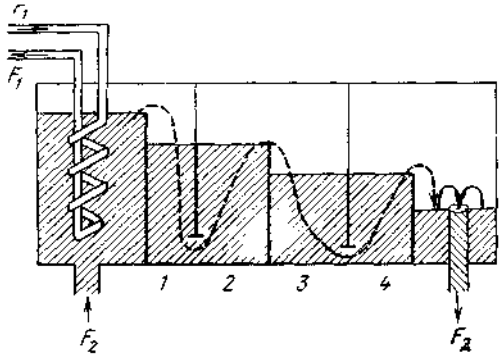


Рис.5.12. Схема процесса подогрева воды

Бак с теплой водой разделяется на несколько отсеков. Переменный поток холодной воды F_2 проходит последовательно отсеки и покидает бак в последнем отсеке. Холодная вода нагревается в теплообменнике, в котором течет по трубам переменный поток горячей воды F_1 с температурой около 90°С. Задача состоит в поддержании постоянной температуры воды в одном из отсеков и, по возможности, в сохранении постоянства потока F_2, посредством регулирования динамических значений F_1 и F_2. Температура воды, покидающей нагревающий отсек, должна регулироваться так, чтобы минимизировать время задержки потока воды. Для такого процесса обычно требуется постоянное количество воды, так что поток F_2 во время устойчивого периода должен поддерживаться постоянным. Поток F_2 может быть изменен только во время изменения желаемой температуры. Следовательно, основной переменной, используемой при управлении процессом, будет поток горячей воды F_1.

Предыдущие исследования этого процесса показали, что при создании управляющих (регулирующих) устройств возникают трудности, связанные с преодолением нелинейности, асимметричности процессов нагревания и охлаждения, различных помех, с сокращением времени реагирования устройств. С другой стороны, на протекание процесса влияет окружающая среда.

Нечеткий логический регулятор процесса подогрева воды. Для описания функций принадлежности нечетких множеств, характеризующих нечеткие оценки значений, была выбрана функция $\mu(x) = (1 + (a(x - c))^b)^{-1}$. Этот вид функции удобен тем, что, изменяя параметры *a, b, c,* можно получить хорошее приближение желаемой функции: *c* позволяет менять точку, в которой минимум

нечеткости ($\mu = 1$); *a* изменяет протяженность функции принадлежности (ее ширину); *b* позволяет менять контрастность. Нечеткие множества, используемые в алгоритме, описывались функциями принадлежности, приведенными в табл. 5.7. В табл. 5.7 *x* обозначает отклонения температуры от фиксированного значения, *dx* — изменение отклонения, F_1— поток теплой воды, dF_1 — изменение F_1, F_2 — поток холодной воды.

Таблица 5.7

Названия нечетких оценок значений	Универсум	Функция принадлежности
Не малое	x	$1 - (1 + 0{,}5x)^{-1}$
Малое	x	$(1 + 0{,}5x)^{-1}$
Очень малое	x	$(1 + x^4)^{-1}$
Незначительно малое	x	$(1 + 0{,}5x)^{-1}$ для $x \geqslant 1$, иначе 0,5
Немалое	x	$(1 + (3(x - 1))^2)^{-1}$
Среднее малое	x	$(1 + (3(x - 0{,}5))^2)^{-1}$
Чрезвычайно малое	x	$(1 + (3x)^2)^{-1}$
Малое	dx	$(1 + (3dx)^2)^{-1}$
Среднее	dx	$(1 + (3(dx - 0{,}5))^2)^{-1}$
Большое	dx	$(1 + (dx - 2)^2)^{-1}$
Очень большое	F_1	$(1 + 2(F_1 - 12)^2)^{-1}$
Очень маленькое	F_1	$(1 + 2(F_1)^2)^{-1}$
Близкое к значению в устойчивом состоянии	F_1	$(1 + (3(F_1 - 1))^2)^{-1}$
Очень близкое к значению в устойчивом состоянии	F_1	$(1 + (3(F_1 - 0{,}5))^2)^{-1}$
Малое	dF_1	$(1 + (2(dF_1 - 0{,}2))^2)^{-1}$
Среднее	dF_1	$(1 + (2(dF_1 - 1))^2)^{-1}$
Большое	dF_1	$(1 + (dF_1 - 3)^2)^{-1}$
Очень большое	F_2	$(1 + 2(F_2 - 18))^{-1}$
Очень маленькое	F_2	$(1 + 2(F_2 - 1)^2)^{-1}$

Первая стратегия описывается следующим множеством правил:

если *x* «не малое», то F_1 «очень большое»
то F_2 «очень маленькое»
если *x* «малое», то F_1 «очень маленькое»
то F_2 «в устойчивом состоянии»
если *x* «очень малое», то F_2 «в устойчивом состоянии»
то, если увеличение *x* «малое»
то уменьшить F_1 «мало»
то, если увеличение *x* «среднее»
то уменьшить F_1 «среднее»
то, если увеличение *x* «большое»
то уменьшение F_1 «большое».

Вторая стратегия реализуется следующими правилами:

если *x* «не малое», то F_1 «очень большое»

то F_2 «очень маленькое»
если x «незначительно малое», то F_1 «очень маленькое»
то F_2 «в устойчивом состоянии»
если x «немалое», то увеличение F_1 «большое»
то F_2 «в устойчивом состоянии»
если x «средне малое», то увеличение F_1 «среднее»
то F_2 «в устойчивом состояпии»
если x «чрезвычайно малое», то увеличение F_1 «малое»
то F_2 «в устойчивом состоянии».

Третья стратегия описывается следующим множеством правил:
если x «не малое», то F_1 «очень большое»
то F_2 «очень маленькое»
если x «малое», то F_1 «близкое к значению в устойчивом состоянии»
то F_2 «в устойчивом состоянии»
если x «очень малое», то F_1 «очень близко» к значению в устойчивом состоянии
то F_2 «в устойчивом состоянии».

Человек-оператор может использовать несколько стратегий поддержания температуры около желаемого значения. Обычно для вычисления величины изменения входа оператор использует отклонение и скорость изменения отклонения температуры. Для сравнения были проанализированы три стратегии:

1. При оценке необходимого изменения величины входного потока оператор использует отклонение и скорость изменения отклонения температуры.
2. При изменении входного потока оператор использует информацию только об отклонении температуры.
3. При регулировании потока около нейтрального положения оператор использует отклонение температуры.

В третьей стратегии предполагается известным абсолютное значение потока F_1, при котором имеет место устойчивое состояние установки. Следовательно, предполагается известной температура постоянного потока.

Эти правила описывают стратегию в случае, когда температура ниже нормы. Аналогичное множество правил было использовано для случая, когда температура выше нормы.

Сравнительный анализ нечетких логических регуляторов, использующих описанные множества правил, соответствующих стратегиям, показал, что первые два регулятора работают с точностью подобной точности человека-оператора (колебания температуры были около 1,5°С). Третий регулятор показал лучшие результаты (колебания температуры около 0,5°С). Это объясняется тем, что в третьем

регуляторе использовалась информация о величине потока в устойчивом состоянии.

5.6. Алгоритмы нечеткой оптимизации

5.6.1.Задачи нечеткого математического программирования

Главная цель нечеткого математического программирования — помочь лицу, принимающему решение, разобраться в выдвинутых им допущениях. Нечеткий подход не подменяет собой простейшего анализа в поисках разумной точности. Он облегчает задачу лица, принимающего решения, позволяя не формулировать явно точные ограничения. Вот почему плодотворный обмен идеями между теорией нечетких множеств и классическим программированием может явиться значительным шагом к созданию новых методов.

Стандартная задача нечеткого математического программирования формулируется обычно как задача максимизации (или минимизации) заданной функции на заданном множестве допустимых альтернатив, которое описывается системой равенств или неравенств. Например:

$$f(x) \to \max, \text{ при } \varphi_i(x) \leqslant 0, \qquad i = 1, ..., m, \ x \in X,$$

где X — заданное множество альтернатив, $f: X \to R$ — заданная функция, которую нужно максимизировать, и $\varphi_i: X \to R$ — заданные функции ограничений.

При моделировании в нечеткой форме реальных задач принятия решений в распоряжении исследователя-математика могут оказаться лишь нечеткие описания функции f и φ_i, параметров, от которых зависят эти функции, и самого множества X. Таким образом, задача стандартного математического программирования превратится в задачу нечеткого математического программирования.

Формы нечеткого описания исходной информации в задачах принятия решений могут быть различными; отсюда и различия в математических формулировках соответствующих задач нечеткого математического программирования.

Перечислим некоторые из таких формулировок.

Задача 1. Максимизация заданной обычной функции $f: X \to R$ на заданном нечетком множестве допустимых альтернатив $\mu: X \to R$.

Задача 2. Нечеткий вариант стандартной задачи математического программирования. Пусть определена следующая задача:

$$f(x) \to \max, \text{ при } \varphi_i(x) \leqslant 0, \quad i = 1, ..., m, \ x \in X.$$

Нечеткий вариант этой задачи получается, если "смягчить" ограничения, т.е. допустить возможность их нарушения с той или иной степенью. Кроме того, вместо максимизации функции $f(x)$ можно стремиться к достижению некоторого заданного значения этой функции, причем различным отклонениям значения функции от этой величины приписывать разные степени допустимости.

Задача 3. Нечетко описана "максимизируемая" функция, т.е. задано отображение $\mu_\varphi; X \times R \to [0,1]$, где X — универсальное множество альтернатив, R — числовая ось.

В этом случае функция $\mu_\varphi(x_0, r)$ при каждом фиксированном $x_0 \in X$ представляет собой нечеткое описание оценки результата выбора альтернативы x_0 (нечеткую оценку альтернативы x_0) или нечетко известную реакцию управляемой системы на управление x_0. Задано также нечеткое множество допустимых альтернатив $\mu_C; X \to [0,1]$.

Задача 4. Заданы обычная максимизируемая функция $f: X \to R$ и система ограничений вида

$\varphi_i(x) \leqslant b_i, \quad i = 1, \ldots, m$, причем параметры в описаниях функций $\varphi_i(x)$ заданы в форме нечетких множеств.

Задача 5. Нечетко описаны как параметры функций, определяющих ограничения задачи, так и самой максимизируемой функции.

Рассмотрим, например, подробнее задачу линейного программирования с нечёткими коэффициентами. Нечеткость в постановке задачи нечеткого математического программирования может содержаться как в описании множества альтернатив, так и в описании целевой функции.

$$f(x) \to \max, \qquad g(x) \leqslant 0, \qquad x \in X. \qquad (1)$$

На практике часто сталкиваются с применением точной теории оптимизации к неточным моделям, где нет оснований приводить точно определенные числа и где слишком часто появляются трудности вычислительного характера при описании больших систем.

Нечеткую обстановку можно рассматривать как множество X альтернатив вместе с его нечеткими подмножествами, представляющими собой нечетко сформулированные критерии (цели и ограничения), т.е. как систему $(X, f_0, f_1, \ldots, f_n)$. Принять во внимание по возможности все критерии в такой задаче означает построить функцию

$$D = f_0 \cap f_1 \cap \ldots \cap f_n, \qquad (2)$$

в которую цели и ограничения входят одинаковым образом.

Решение можно определить как нечеткое подмножество универсального множества альтернатив. Оптимум соответствует той области X, элементы которой максимизируют D. Это и есть случай нечеткого математического программирования.

Очевидно, что в реальных ситуациях неразумно проводить резкую границу для множества допустимых альтернатив. Может случится так, что распределения, попадающие за эту границу, дадут эффект, более желательный для лица, принимающего решения.

Например, ясно, что при несовместных распределениях эта область пустая. В таком случае налицо необходимость модификации ограничений. Желательно выяснить, как изменить ограничения задачи, чтобы появились допустимые решения и задача стала разрешимой.

В таких случаях представляется целесообразным вводить нечеткое множество допустимых элементов и, следовательно, рассматривать проблему как задачу нечеткого математического программирования с применением подхода, дающего человеку больше свободы в использовании его субъективных представлений о ситуации.

Формы нечеткого описания исходной информации в задачах принятия решений могут быть различными; отсюда и различия в математических формулировках соответствующих задач нечеткого математического программирования.

Нечеткий вариант стандартной задачи математического программирования получается, если "смягчить" ограничения, т.е. допустить возможность их нарушения с той или иной степенью. Кроме того, вместо максимизации целевой функции $f(x)$ можно стремиться к достижению некоторого заданного ее значения, причем различным отклонениям значения $f(x)$ от этой величины приписывать различные степени допустимости (например, чем больше отклонение, тем меньше степень его допустимости).

Пусть a — заданная величина функции цели $f(x)$, достижение которой считается достаточным для выполнения цели принятия решений, и пусть имеется пороговый уровень b, такой, что неравенство $f(x) < a - b$ означает сильное нарушение неравенства $f(x) \geqslant a$. Тогда функцию принадлежности для нечеткой функции цели можно определить следующим образом:

$$\mu_G(x) = \begin{cases} 0, & \text{если} \quad f(x) \leqslant a - b, \\ \mu_a(x), & \text{если} \quad a - b < f(x) < a, \\ 1, & \text{если} \quad f(x) \geqslant a, \end{cases} \quad (3)$$

где μ_a — функция принадлежности, описывающая степени выполнения соответствующего неравенства с точки зрения лица, принимающего решения.

Аналогично определяется функция принадлежности $\mu_C(x)$ для нечетких ограничений. В результате исходная задача оказывается сформулированной в форме задачи выполнения нечетко определенной цели, к которой применим подход Беллмана-Заде .

При моделировании ситуации в форме задачи линейного программирования

$$\min\{cx \mid Ax \leqslant b,\ x \geqslant 0\} \quad (4)$$

о коэффициентах a_{ij}, b_i и c_i известно лишь то, что они находятся в некотором множестве, отражающем все реальные возможности.

В отдельных случаях точное описанное множество ограничений (допустимых альтернатив) может оказаться лишь приближением реальности в том смысле, что в реальной задаче альтернативы вне множества ограничений могут быть не допустимыми, а лишь в той или иной степени менее желательными для лица, принимающего решения, чем альтернативы внутри этого множества.

Рассмотрим задачу нахождения минимума на заданной области. Пусть задана область вида

$$P = \{x \in {R_+}^n \mid a_{i1}x_1 + \ldots + a_{in}x_n \subseteq b_i,\ i = 1, \ldots, m\}, \quad (5)$$

где a_{ij}, b_i — нечеткие подмножества множества R, а бинарная операция $+$ обозначает сложение нечетких множеств. Требуется найти $\min\limits_{x \in P} \{c, x\}$ на заданной области.

Коэффициент при каждой переменной в ограничениях можно считать функцией полезности, определенной на числовой оси. Можно полагать, что эти коэффициенты дают субъективную оценку различных возможностей, включая, таким образом, другие не определенные ограничения.

Сведем решение исходной задачи к решению ряда задач линейного программирования. Для этого введем дискретные α -уровни. В результате нечеткие ограничения принимают следующий интервальный вид:

$$P = \begin{cases} \sigma_\alpha(a_{i1})x_1 + \ldots + \sigma_\alpha(a_{in})x_n, & i = 1, \ldots, m, \ \alpha = 1, \ldots, p, \\ x_j \geqslant 0, & j = 1, \ldots, n. \end{cases} \quad (6)$$

Таким образом, мы перешли от нечетких множеств к четко определенным и теперь, зная, что α — обычный интервал, можем записать нашу задачу в следующем виде:

$$\begin{aligned} &(a_{11}, a_{12})x_1 + (c_{11}, c_{12})x_2 \subseteq (b_{11}, b_{12}), \\ &(a_{21}, a_{22})x_1 + (c_{21}, c_{22})x_2 \subseteq (b_{21}, b_{22}). \end{aligned} \quad (7)$$

Теперь, чтобы привести задачу к виду обычной задачи линейного программирования, нам достаточно записать неравенства отдельно по левому и правому краям интервалов, с учетом знаков неравенства. Т.е., мы приведем систему к следующему виду:

$$\begin{aligned} a_{11}x_1 + c_{11}x_2 &\geqslant b_{11}, \\ a_{12}x_1 + c_{12}x_2 &\leqslant b_{12}, \\ a_{21}x_1 + c_{21}x_2 &\geqslant b_{21}, \\ a_2x_1 + c_{22}x_2 &\leqslant b_{22}. \end{aligned} \quad (8)$$

С помощью несложных преобразований мы перешли от задачи с нечеткими коэффициентами к задаче линейного программирования с четкими коэффициентами; при этом количество ограничений

увеличилось в два раза и полученную задачу мы можем решить симплексным методом.

Таким образом, из рассмотренного примера явно просматривается алгоритм решения задачи с нечеткими коэффициентами. Следуя ходу рассуждений в данном примере, составим такой алгоритм. Он имеет следующий вид:

1. Исходная задача.
2. Вводим дискретные α -уровни.
3. Ограничения принимают интервальный вид.
4. Записываем неравенства отдельно по левому и правому краям с учетом знаков неравенства (при этом размерность увеличивается).
5. Получаем задачу ЛП с четкими коэффициентами.
6. Решаем полученную задачу симплекс-методом.

Как видим, исходная задача нечеткого математического программирования представляется в виде совокупности обычных задач линейного программирования на всевозможных множествах уровня множества допустимых альтернатив. Если альтернатива x_0 есть решение задачи $\min\limits_{x \in P} \{c, x\}$ на множестве уровня α, то можно считать, что число α есть степень принадлежности альтернативы x_0 нечеткому множеству решений исходной задачи.

Перебрав, таким образом, всевозможные значения α, получаем функцию принадлежности нечеткого решения.

Если же и компоненты целевой функции c_i являются нечеткими, то необходимо выбирать для каждого уровня α соответствующие границы множеств $\sigma_\alpha(c_J)$, $J = 1, \ldots, n$ в соответствии с правилами интервальной арифметики, минимизируя предварительно таким образом: $\{c, x\}$.

Из данного примера видно, что за гибкость приходится платить ценой увеличения размерности задачи. Фактически, исходная задача с ограничениями по включению преобразуется в задачу с ограничениями

в виде неравенств, с которыми легко обращаться; при этом такая цена не слишком высока, поскольку сохраняется возможность использования хорошо разработанных классических методов.

5.6.2. Модели нечеткой ожидаемой полезности

При описании индивидуального принятия решения в рамках классического подхода, наряду с моделями математического программирования, широко применяются теория статистических решений и теория ожидаемой полезности. Последняя предназначена для анализа решений, когда неопределенность обусловлена отсутствием объективной физической шкалы для оценки предпочтительности альтернатив. В этих случаях используется субъективная шкала полезности лица, принимающего решение (ЛПР). В реальных ситуациях исходы, соответствующие принятым решениям (состояниям системы), являются подчас неточными, что влечет за собой размытость соответствующих им оценок функции полезности. Размытый вариант ожидаемой полезности формулируется, например, в модели, где выделяются и одновременно учитываются как случайные, так и нечеткие составляющие неопределенности. Выбор происходит на основе максимизации нечеткой ожидаемой полезности

$$ER_j = \sum_{i=1}^{n} \tilde{p}_i F(s_i, a_j, b_k),$$

где $\tilde{p}_i$ — размытая вероятность состояния s_i из множества состояний мира S, $F: \quad S \times A \times B \to \wp(R)$, $A = \{a\}$ — множество альтернатив, $B = \{b\}$ — множество критериев, R — множество оценок, а $\wp(R) = \{\mu_R \mid \mu_R : \ R \to [0,1]\}$ — класс всех нечетких подмножеств на множестве оценок R.

Существуют модели, в которых описываются нечеткие лотереи, нечеткие деревья предпочтения, нечеткие байесовские оценки и т.п., где неполнота информации о законе распределения вероятности

моделируется с использованием нечетких чисел и лингвистических вероятностей.

Например, задача анализа решений формулируется следующим образом. Пусть имеются две обычные вероятности лотереи: $A = [pu_{A_1},\ (1-p)u_{A_2}]$, где p — вероятность исхода с ожидаемой полезностью u_{A_1} и $(1-p)$ — вероятность исхода с ожидаемой полезностью u_{A_2}, а $B = [qu_{B_1},\ (1-q)u_{B_2}]$, где q — вероятность исхода с ожидаемой полезностью u_{B_1}, $(1-q)$ — вероятность исхода с ожидаемой полезностью u_{B_2}. Из теории ожидаемой полезности следует, что $A \succ B$, если

$$pu_{A_1}, +(1-p)u_{A_2} > qu_{B_1} + \ (1-q)u_{B_2}.$$

Будем считать, что вероятности p и q и ожидаемые полезности $u_{A_1},\ u_{A_2},\ u_{B_1},\ u_{B_2}$ точно не известны, т.е. введем

$$\mu_P :\ P \to [0,1], \quad \mu_Q :\ Q \to [0,1], \quad \mu_U :\ U \to [0,1].$$

Тогда, в соответствии с принципом обобщения, степени принадлежности альтернатив a и b множествам нечетких ожидаемых полезностей в нечетких лотереях A и B соответственно вычисляются

$$\mu_A(a) = \max_{pu_{A1}+(1-p)u_{A2}=a} \left[\min\{\mu_P(p),\ \mu_{A_1}(u_{A_1}),\ \mu_{A_2}(u_{A_2})\}\right],$$
$$\mu_B(b) = \max_{qu_{B1}+(1-q)u_{B2}=b} \left[\min\{\mu_P(p),\ \mu_{B_1}(u_{B_1}),\ \mu_{B_2}(u_{B_2})\}\right].$$

В случае лотереи с n исходами также для каждого ребра дерева решений подсчитывается значение нечеткой ожидаемой полезности.

5.7. Алгоритмы нечеткого контроля и управления

5.7.1. Игры в нечетко определенной обстановке

Во многих прикладных областях часто встречаются ситуации, в которых выполнение цели или результаты принятия решений одним лицом зависят не только от его действий, но и от действий другого лица или группы лиц, преследующих свои собственные цели. Рассмотренный подход к задачам принятия решений можно применять и для анализа подобных игровых ситуаций в нечетко определенной обстановке. Формулируется такая игра следующим образом.

Пусть X и Y — множества элементов, которые могут выбирать игроки 1 и 2, соответственно. Допустимые выборы (стратегии) игроков 1 и 2, описываются нечеткими множествами C_1 и C_2 в X и Y соответственно с функциями принадлежности μ_{C_1} и μ_{C_2}. Заданы также функции $f_1,\ f_2\ :\ X \times Y \to R$, причем значение $f_i(x,y)$ есть оценка игроком i ситуации (x,y) без учета допустимости выборов x и y. Цель игрока i описывается нечетким множеством G_i в R с функцией принадлежности $\mu_{G_i}\ :\ R \to [0,1]$. Следует заметить, что цель, поставленная игроком, может оказаться плохо совместимой или вообще несовместимой с его возможностями, т.е. с множеством его стратегий.

Целью игрока i можно считать нечеткое множество в $X \times Y$ с функцией принадлежности

$$\mu_{\bar{G}_i}(x,y) = \mu_{G_i}(f_i(x,y)), \qquad \forall (x,y) \in X \times Y.$$

Образом этого нечеткого множества при отображении f_i является заданное нечеткое множество цели игрока i.

Введем нечеткие множества D_1 и D_2 в $X \times Y$, определив их функции принадлежности следующим образом:

$$\mu_{D_1}(x,y) = \mu_{C_1}(x) \wedge \mu_{\bar{G}_1}(x,y),$$
$$\mu_{D_2}(x,y) = \mu_{C_2}(x) \wedge \mu_{\bar{G}_2}(x,y).$$

Смысл нечетких множеств D_1 и D_2 можно пояснить так. Если, например, игроку 1, известен конкретный выбор y^* игроком 2, то перед ним стоит задача достижения нечеткой цели $\mu_{\bar{G}_2}(x,y^*)$ при множестве допустимых альтернатив $\mu_{C_1}(x)$. В соответствии с подходом Беллмана-Заде , решение D_1 такой задачи определяется как пересечение нечетких множеств цели и ограничения:

$$\mu_{D_1}(x,y^*) = \mu_{C_1}(x) \wedge \mu_{\bar{G}_1}(x,y^*).$$

Таким образом, нечеткое множество D_1 можно рассматривать как семейство (по параметру y) решений задач достижения нечетких целей $\mu_{\bar{G}_1}(x,y^*)$. Аналогичный смысл придается и множеству D_2.

Далее будем считать, что при каждом фиксированном выборе одного игрока второй выбирает стратегию, которая максимизирует соответствующую ему функцию μ_{D_i}.

Если игрок полагается целиком лишь на свои возможности, то естественна его ориентация на получение **наибольшего гарантированного выигрыша**, т.е. рациональным считается такой способ оценки игроком 1 своих выборов, при котором он рассчитывает на наихудшую для него реакцию игрока 2 из множества возможных реакций последнего.

При этом важную роль играет имеющаяся в его распоряжении информация об интересах и **ограничениях** игрока 2. Если, например, игрок 1 имеет возможность первым выбрать свою стратегию, а игроку 2 становится известным этот выбор, то наибольший гарантированный выигрыш игрока 1 равен

$$H_1 = \max_{x \in X} \min_{y \in Y(x)} \mu_{D_1}(x, y).$$

Присутствующее в этом выражении множество $Y(x)$, зависящее от x, есть множество возможных реакций (ответов) игрока 2 на выбор x игрока 1. В этом смысле зависимость $Y(x)$ отражает степень информированности игрока 1 об интересах и **ограничениях** игрока 2.

Если величина H_1 слишком мала, это означает, что цель, к выполнению которой стремится игрок 1, слишком завышена (с учетом его возможностей). Поэтому естественным образом возникает следующая задача. Каково должно быть нечеткое множество стратегий игрока 1, которое гарантировало бы ему (при заданной информированности об игроке 2) достижение цели со степенью, не меньшей некоторого заданного числа α ?

Для решения этой задачи введем множество

$$X_\alpha = \left\{ x \mid \min_{y \in Y(x)} \mu_{\bar{G}_1}(x, y) \geqslant \alpha \right\} \subset X.$$

Если $X_\alpha = \varnothing$, то $H_1 < \alpha$, и, следовательно, игрок 1 не может гарантировать достижение своей цели со степенью большей или равной α, независимо от того, какое множество стратегий находится в его распоряжении.

Пусть $X_\alpha \neq \varnothing$, тогда можно заключить, что достижение цели со степенью не менее α можно гарантировать только тогда, когда $\mu_{C_1}(x) \geqslant \alpha$ при некотором $x \in X_\alpha$.

5.7.2. Многошаговые процессы принятия решений

Для простоты будем полагать, что управляемая система A является инвариантной по времени детерминированной системой с конечным числом состояний. Именно каждое состояние x_t, в котором система A находится в момент времени t, $t = 0, 1, 2, \ldots$, принадлежит заданному конечному множеству возможных состояний $X = \{\sigma_1, \ldots, \sigma_n\}$; при этом входной сигнал в момент времени t является элементом множества $U = \{\alpha_1, \ldots, \alpha_m\}$. Динамика системы во времени описывается уравнением состояния

$$x_{t+1} = f(x_t, u_t), \quad t = 0, 1, 2, \ldots$$

в котором f — заданная функция, отображающая $X \times U$ в X. Таким образом, $f(x_t, u_t)$ представляет собой последующее состояние для x_t при входном сигнале u_t. Считается также, что заданы начальное состояние x_0 и фиксированное время окончания процесса N.

Предполагается, что в каждый момент времени t на входную переменную наложено нечеткое ограничение C_t, являющееся нечетким множеством в U с функцией принадлежности $\mu_t(u_t)$. Кроме того, считается, что цель — нечеткое множество G_N в X, определяемое функцией принадлежности $\mu_{G_N}(u_N)$. Задача заключается в нахождении максимизирующего решения.

Можно записать решение как нечеткое множество в $U \times \ldots \times U$ в виде

$$D = C_0 \cap C_1 \cap \ldots \cap C_{N-1} \cap \bar{G}_N,$$

где G_N — нечеткое множество в $U \times \ldots U$, индуцируемое G_N в X. Для функции принадлежности имеем

$$\mu_D(u_0, \ldots, u_{N-1}) = \mu_0(u_0) \wedge \ldots \wedge \mu_{N-1}(u_{N-1}) \wedge \mu_{G_N}(x_N),$$

где x_N может быть выражено как функция от $u_1, \ldots, u_{N-1}$ и x_0 путем последовательного применения уравнения $x_{t+1} = f(x_t, u_t)$.

Для многошаговых процессов целесообразно представить решение в виде:

$$u_t = \pi_t(x_t), \quad t = 0, 1, \ldots, N-1,$$

где π_t — принятая "стратегия", или правило выбора входного воздействия u_t в зависимости от состояния системы x_t.

Таким образом, задача сводится к нахождению оптимальных стратегий π_t и соответствующей последовательности входных воздействий $u_1, \ldots, u_{N-1}$, максимизирующих μ_D. Для решения применяется метод динамического программирования:

$$\mu_D(u_0^M, ..., u_{N-1}^M) =$$
$$= \max_{u_0,\ldots,u_{N-2}} \max_{u_{N-1}} (\mu_0(u_0) \wedge \ldots \wedge \mu_{N-1}(u_{N-1}) \wedge \mu_{G_N}(f(x_{N-1}, u_{N-1}))) =$$
$$= \max_{u_0,\ldots,u_{N-2}} (\mu_0(u_0) \wedge \ldots \wedge \mu_{N-2}(u_{N-2}) \wedge \mu_{G_{N-1}}(x_{N-1})),$$

где

$$\mu_{G_{N-1}}(x_{N-1}) = \max_{u_{N-1}}(\mu_{N-1}(u_{N-1}) \wedge \mu_{G_n}(f(x_{N-1}, u_{N-1})))$$

может рассматриваться как функция принадлежности нечеткой цели в момент $t = N-1$, индуцированной заданной целью G_N в момент $t = N$.

Повторяя процесс обратных итераций, получаем систему рекуррентных уравнений

$$\mu_{G_{N-v}}(x_{N-v}) = \max_{u_{N-v}} (\mu_{N-v}(u_{N-v}) \wedge \mu_{G_{N-v+1}}(x_{N-v+1})),$$

где

$$x_{N-v+1} = f(x_{N-v}, u_{N-v}), \quad v = 1, \ldots, N,$$

которая дает решение задачи. Таким образом, максимизирующее решение достигается последовательной максимизацией величин u_{N-v}, причем u_{N-v}^{M} определяется как функция от $x_{N-v}, \quad v = 1, \ldots, N$.

В качестве простого примера рассмотрим систему с тремя состояниями σ_1, σ_2 и σ_3 и двумя входными сигналами α_1 и α_2. Пусть $N = 2$ и нечеткая цель в момент времени $t = 2$ определяется функцией принадлежности, принимающей значения

$$\mu_{G_1}(\sigma_1) = 0,3; \quad \mu_{G_2}(\sigma_2) = 1; \quad \mu_{G_3}(\sigma_3) = 0,8.$$

Пусть далее, нечеткие ограничения в моменты $t = 0$ и $t = 1$ задаются функциями

$$\mu_0(\alpha_1) = 0,7; \quad \mu_0(\alpha_2) = 1;$$
$$\mu_1(\alpha_1) = 1; \quad \mu_1(\alpha_2) = 0,6.$$

Допустим, что таблица изменения состояний, задающая функцию f, имеет следующий вид:

	σ_1	σ_2	σ_3
α_1	1	3	1
α_2	2	1	3

Находим функцию принадлежности нечеткой цели в момент $t = 1$:

$$\mu_{G_1}(\sigma_1) = 0,6; \quad \mu_{G_2}(\sigma_2) = 0,8; \quad \mu_{G_3}(\sigma_3) = 0,6.$$

Соответствующее максимизирующее решение имеет вид:

$$\pi_1(\sigma_1) = \alpha_2; \quad \pi_1(\sigma_2) = \alpha_1; \quad \pi_1(\sigma_3) = \alpha_2.$$

Аналогично, для $t = 0$имеем

$$\mu_{G_1}(\sigma_1) = 0,8; \quad \mu_{G_2}(\sigma_2) = 0,6; \quad \mu_{G_3}(\sigma_3) = 0,6,$$
$$\pi_0(\sigma_1) = \alpha_2; \quad \pi_0(\sigma_2) = \alpha_1 \vee \alpha_2; \quad \pi_0(\sigma_3) = \alpha_1 \vee \alpha_2.$$

Итак, если начальное состояние в момент времени $t = 0$есть σ_1, то максимизирующим решением будет α_2, причем соответствующее значение функции принадлежности μ_{G_i} равно 0,8.

5.7.3. Особенности контроля и управления в условиях стохастической неопределенности

При составлении проекта его авторы редко располагают полной априорной информацией об объекте и окружающей его среде, необходимой для синтеза корректной системы управления. Даже если известны системы уравнения, описывающие поведение системы, то часто оказывается, что нет данных о величине отдельных параметров, и к тому же нередко имеющиеся модели слишком сложны. В дальнейшем выясняется, что принятая при проектировании модель существенно отличается от реального объекта, а это значительно уменьшает эффективность разработанной системы управления. В связи с этим, актуальной становится возможность уточнения модели на основе наблюдений, полученных в условиях нормального функционирования объекта.

Таким образом, задача идентификации формулируется следующим образом: по результатам наблюдений над входными и выходными переменными системы должна быть построена оптимальная в

некотором смысле модель, т.е. формализованное представление этой системы.

В зависимости от априорной информации об объекте управления различают задачи идентификации в узком и широком смысле. Для вторых приходится предварительно решать большое число дополнительных проблем. К ним относятся: выбор структуры системы и задание класса моделей, оценка степени стационарности и линейности объекта, а также степеней и форм влияния входных воздействий на состояние, выбор информативных переменных и др. Задача идентификации в узком смысле состоит в оценке параметров и состояния системы по результатам наблюдений над входными и выходными переменными, полученными в условиях функционирования объекта. Для решения отмеченных проблем в современной теории управления обычно используют модели в пространстве состояний.

Проблеме построения алгоритмов управления объектами с неполной информацией в настоящее время уделяется большое внимание. Это объясняется прежде всего тем, что при создании систем управления сложными технологическими процессами обычно не располагают достоверными моделями объектов. Ни одна из существующих теорий не может претендовать на то, что единственно она дает правильное описание работы систем. Скорее, имеется целый спектр теорий, трактующих эти проблемы. При имеющемся сейчас узком рассмотрении лишь отдельных процессов и только на определенных уровнях описания получается одностороннее представление о системе, не позволяющее иметь достоверные оценки обо всех процессах.

Поведение реальной системы характеризуется некоторой неопределенностью, и при достаточно большом объеме информации об объекте некоторое внешнее возмущение, действующее на управляемый объект, можно представить как случайный процесс.

Стохастическое оптимальное управление в значительной степени базируется на основных положениях динамического программирования.

Для линейных систем с квадратичным критерием решение исходит из так называемой теоремы разделения, которая позволяет составлять наилучшую стратегию из двух частей: оптимального фильтра, который

вычисляет оценки состояния в виде условного среднего при заданных наблюдениях выходных сигналов, и линейной обратной связи. Оказывается, что линейная обратная связь может быть найдена путем решения задачи детерминированного управления. Оценка состояния характеризует выходную переменную фильтра Калмана, который, по существу, представляет собой математическую модель системы, когда управление осуществляется по наблюдениям. Таким образом, теорема разделения обеспечивает связь между теориями фильтрации и стохастического оптимального управления.

5.7.4. Контроль и управление динамическими системами в нечетких условиях

Применение стохастических методов для контроля и управления процессом в некоторых ситуациях оказывается затруднительным из-за отсутствия вероятностных распределений параметров. Сложность получения численных результатов при работе со случайными величинами также снижает практическую ценность стохастических алгоритмов. В случае неполной информации о сложном процессе удобнее представлять неточно заданные параметры в виде нечетких величин.

Коэффициенты целого ряда моделей фактически зависят от многих неучтенных факторов реального процесса. При описании процессов двухмерными моделями мы заменяем трехмерную модель однородным по третьему измерению слоем и значения коэффициентов для него определяем как среднее, средневзвешенное и т.д. Попытка внесения в модель ряда не учтенных ранее факторов и введение третьего измерения приводят к значительному усложнению модели и резкому повышению размерности задачи. К тому же, в такой усложненной модели появляются параметры, которые невозможно или крайне трудно измерить. При их задании опять вводятся некоторые допущения, которые только затрудняют и ухудшают точность решения задачи.

Как показывает практика, использование детерминированных моделей с четкими значениями параметров (даже при наличии адаптационного процесса их уточнения путем решения обратных задач) приводит к тому, что модель оказывается излишне грубой. Методы интервального анализа дают возможность построить модель для случая, когда для каждого из этих коэффициентов задан интервал допустимых значений.

Однако на практике, когда имеется информация, что некие значения коэффициентов более допустимы, чем другие, описание этих коэффициентов в виде нечетких множеств является более удачным. В этом случае на интервале дополнительно задается функция принадлежности, причем, если информация о различии допустимости имеет статистический характер, то эта функция может быть определена объективно, если нет — то субъективно, на основе приближенного отражения экспертом в агрегированном виде имеющегося у него неформализованного представления о величине этого коэффициента.

Естественно, что введение нечетких коэффициентов усложняет процесс моделирования, однако в этом случае решение адекватно принятым упрощениям, например, при исключении третьей координаты z понятие в точке (x, y) становится размытым, нечетким, так как относится не к точке, а к интервалу.

В общем случае динамику дискретных систем можно представить уравнением состояния:

$$x_{k+1} = F(x_k, u_k) \quad k = 0, \ldots, N, \quad x_k \in X, \quad u_k \in U,$$

где X — пространство состояний, U — множество допустимых управлений, F — переходная функция состояния, в общем случае нелинейная $F: X \times U \to X$.

Эта система является детерминированной, если в любой момент времени k можно однозначно определить ее новое состояние для момента времени $(k+1)$ по текущему состоянию l и управлению u_k.

Для стохастических систем переходная функция записывается в виде

$$F: X \times U \to XP,$$

где XP — множество распределений вероятности на X. Для учета неопределенностей в модель могут вводиться случайные величины или коэффициенты. Однако для подобных моделей необходимо иметь информацию для построения вероятностных распределений.

Не полностью определенные процессы можно моделировать с помощью аппарата нечетких множеств. Коэффициенты и некоторые величины могут быть заданы в виде функций принадлежности. Тогда динамика системы описывается нечетким отношением

$$F\colon X \times U \times X \to [0,1],$$

представляющим собой нечеткое подмножество декартова произведения $X \times U \times X$.

Величина $F(x_k, u_k, x_{k+1})$ рассматривается как интенсивность перехода или, точнее, как степень принадлежности элемента x_{k+1} образу пары (x_k, u_k) при отображении F, т.е. основной характеристикой системы является функция принадлежности $\mu(x_{k+1}|x_k, u_k)$.

Используя понятие нечеткого отношения, можно ввести следующие пути определения функции F:

1. Когда отсутствует модель процесса и имеется лишь лингвистическое описание желаемого поведения системы вида "если давление газа очень большое, то значительно увеличить расход". Подобные выражения дают информацию о том, что должно произойти в системе при поступлении на ее вход управляющих воздействий в форме нечетких множеств, определенных на универсальных множествах "давление газа" и "расход". Тогда нечеткое условное высказывание есть нечеткое отношение, которое определяется как

$$F(x,u) = \min(\mu(x), \lambda(x)); \quad \mu\colon X \to [0,1]; \quad \lambda\colon X \to [0,1].$$

Если F будет являться нечеткой функцией, то состояние нечеткой системы в момент времени $(k+1)$ есть условное по x_k и u_k нечеткое множество, характеризуемое функцией принадлежности $(x_{k+1}|x_k, u_k)$.

2. Возможно использование имеющейся модели системы для задания функции F. Рассмотрим вначале случай свободной динамики системы и построим рекуррентную процедуру оценки состояния динамической системы в нечетких условиях.

На практике ситуация усложняется частичным или полным отсутствием информации о статистических характеристиках шумов. Поэтому предлагается для решения задачи оценивания применять теорию нечетких множеств.

Рассмотрим нелинейную динамическую систему с дискретным временем:

$$x_{k+1} = F_k(x_k, w-k), \quad k = 1, 2, \ldots,$$

для которой измерение и состояние системы связаны соотношением

$$z_k = H_k(x_k, v_k).$$

В этих уравнениях:

- индекс k соответствует k -му моменту времени;
- F_k, H_k — нелинейные функции соответствующих аргументов;
- x_k — состояние динамической системы,
- w_k — нечеткая помеха, заданная для каждого момента времени k -функцией принадлежности $\mu(w_k)$;
- v_k — ошибка измерения с известной функцией принадлежности $\mu(v_k)$.

Предполагается известной и функция принадлежности для начального состояния $\mu(x_0)$.

В процессе функционирования системы в общем случае носитель начального нечеткого состояния расширяется. Чтобы уменьшить неопределенность ситуаций при принятии решений, необходимо

использовать дополнительную информацию о замерах и исследованиях в системе.

Будем предполагать независимость ошибок измерения, помех и состояния в смысле определения независимости нечетких величин.

При заданной условной функции принадлежности $\mu(x_k \mid \bar{z}_k)$ состояния x_k и при наличии последовательности измерений $\bar{z}_k = \{z_0, z_1, \ldots, z_k\}$, наилучшая четкая оценка состояния в момент времени k может быть найдена из соотношения

$$\mu(x_k^0) = \max_{x_k} \mu(x_k \mid \bar{z}_k).$$

При наличии известной условной функции принадлежности $\mu(x_{k+1} \mid \bar{z}_k)$ оптимальная точечная оценка состояния системы в момент $(k+1)$ может быть определена аналогично:

$$\mu(x_{k+1}^0) = \max_{x_{k+1}} \mu(x_{k+1} \mid \bar{z}_k).$$

Поскольку для реальных процессов функции $\mu(x_k \mid \bar{z}_k)$ и $\mu(x_{k+1} \mid \bar{z}_k)$ являются унимодальными, то процедура нахождения максимума довольно проста. Чтобы оценить состояния, выведем рекуррентную процедуру для функции принадлежности $\mu(x_{k+1} \mid \bar{z}_{k+1})$. На основании определения условной функции принадлежности можно записать, что

$$\mu(x_{k+1}|\bar{z}_{k+1}) = \mu(x_{k+1}, \bar{z}_{k+1}) = \mu(x_{k+1}, \bar{z}_k, z_{k+1}),$$

где вектор $\bar{z}_{k+1}$ представлен в виде $\bar{z}_{k+1} = \{\bar{z}_k, z_{k+1}\}$.

Используя определение $\mu(x_{k+1} \mid \bar{z}_{k+1})$ и уравнение для ошибки измерения, получаем:

$$\mu(x_{k+1},\ \bar{z}_k,\ z_{k+1}) = \sup_{v_{k-H}=H_{k-H}^{-1}(x_{k-H},z_{k-H})} \mu(v_{k+1}, x_{k+1}, \bar{z}_k).$$

Окончательно рекуррентные соотношения для нахождения апостериорной функции принадлежности для нечеткого состояния системы на любом шаге $(k+1$) можно записать следующим образом:

$$\begin{cases} \mu(x_{k+1}|\ \bar{z}_{k+1}) = \mu(x_{k+1}|\ \bar{z}_k) \wedge \sup\limits_{v_{k-H}=H_{k-H}^{-1}(x_{k-H},z_{k-H})} \mu(v_{k+1}); \\ \mu(x_{k+1}|\ \bar{z}_{k+1}) = \max\limits_{x_k} \left\{ \mu(x_{k+1}|\ \bar{z}_k) \wedge \sup\limits_{w_k=F_{k-H}^{-1}(x_{k-H},x_k)} \mu(w_k, x_k, \bar{z}_k) \right\}. \end{cases}$$

Рассмотрим теперь принципы управления нечеткой динамической системой для функции F. Допустим, что на управляющее воздействие u_k в каждый момент времени k наложены нечеткие ограничения $C_k \subset U$, характеризующиеся функцией принадлежности $\mu_{C_k}(u_k)$, и также задано начальное состояние x_0. Пусть $G_N \subset X$ — нечеткая цель, которую необходимо достигнуть в момент времени N. Эта цель характеризуется функцией принадлежности $\mu_{G_N}(x)$.

Оптимальные четкие управляющие воздействия $u_1^0,\ u_2^0, \ldots, u_{N-1}$ могут быть определены следующим образом:

$$\mu_D(u_0^0, \ldots, u_{N-1}^0) = \max_{u_0,\ldots,u_{N-2}} \left\{ \mu_{C_0}(u_0) \wedge \ldots \wedge \mu_{C_{N-2}}(u_{N-2}) \wedge \mu_{G_{N-1}}(x_{N-1}) \right\}.$$

Функция $\mu_{G_{N-1}}(x_{N-1})$ может рассматриваться как функция принадлежности для нечеткой цели в момент времени $N-1$, индуцированной конечной целью G_N для момента N. Зная текущее нечеткое состояние $\mu(x_k)$, нечеткое ограничение $\mu_{C_k}(u_k)$ и индуцированную нечеткую цель $\mu_{C_k}(u_k)$, на момент времени k можно найти эффективное четкое управление u_k^0

6. Прикладная теория алгоритмов. Характеризационный анализ

6.1. Принципы характеризационного анализа

Характерной чертой современной научно-технической революции является возрастающая роль вычислений комбинаторного (переборного) характера в прикладных задачах. Актуальной проблемой дискретной математики является построение эффективных как по объему необходимой памяти, так и по быстродействию комбинаторных алгоритмов.

Прикладные задачи можно подразделить на задачи анализа и задачи синтеза дискретных систем. Под решением задачи анализа подразумевается определение того, обладает ли модель Ψ_a, представляющая дискретную систему, требуемыми свойствами. Решение задачи синтеза предполагает проведение преобразования модели Ψ_a в модель Ψ_b, при котором достигается экстремум заданного функционала качества $\varphi(\Psi_b)$. В обоих случаях можно говорить об эквивалентировании. В задачах анализа по модели Ψ_a строится ее эквивалент, раскрывающий свойства модели. В задачах синтеза модель Ψ_a эквивалентируется синтезируемой моделью Ψ_b . Как анализ, так и синтез осуществляют с помощью комбинаторных алгоритмов.

Примитивным классом комбинаторных алгоритмов являются ГСН-алгоритмы (ГСН — Грубая Сила и Невежество). Это алгоритмы, лишенные какой-либо «искусности», они решают задачи «вслепую», проводя полный перебор возможных преобразований. В этом случае

отсутствуют теоретические предпосылки, на основании которых можно была бы предложить «утонченный» алгоритм решения. В алгоритмах этого класса реализуется синтаксическое эквивалентирование.

Синтаксическое эквивалентирование, называемое *эквивалентным преобразованием,* соответствует уровню знаний, при котором известна полная система аксиом, и заключается в построении очередного варианта для задач анализа и в замене модели Ψ_a моделью Ψ_b для задачи синтеза на основании той или иной аксиомы или закона, полученного из системы аксиом. Дерево решений при синтаксическом эквивалентировании изображено на рис. 6.1, *а.*

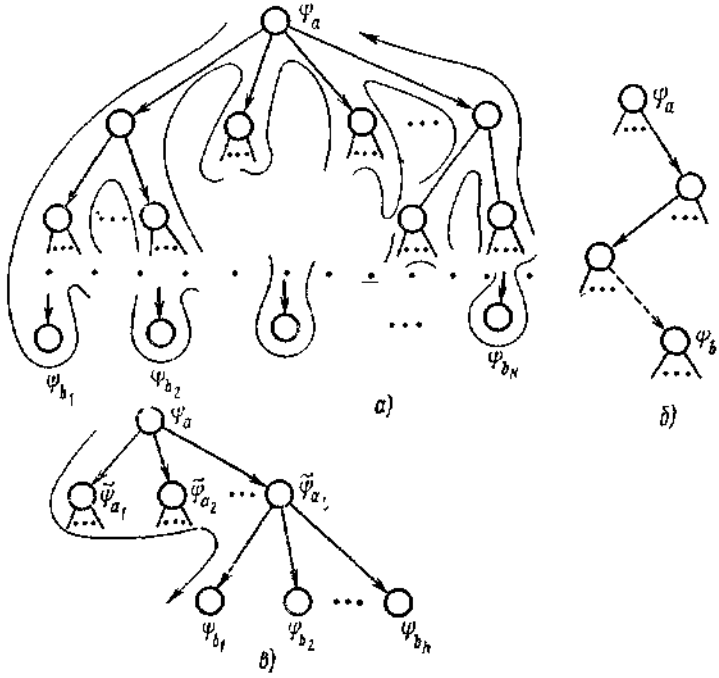


Рис. 6.1

Каждая висячая вершина дерева соответствует тупиковому решению. Характерными свойствами этого дерева являются:

1) комбинаторный рост числа висячих вершин при линейном росте размерности задачи;

2) необходимость «возврата» на предыдущий (i — 1)-й уровень при вычислении информации, соответствующей «соседней» вершине в i -м

уровне. Следствием этого свойства является принципиальная необходимость обхода всего дерева при поиске минимального решения. Эти свойства определяют явление, называемое «проклятием размерности».

При разработке математического и программного обеспечения ЭВМ, особенно при работе в реальном масштабе времени, при создании систем автоматизации проектирования (проблема САПР) и т. д., актуальным является проектирование быстродействующих пакетов прикладных программ. Для повышения быстродействия алгоритмов используют эвристики в виде соответствующих функционалов, найденных на основании опыта, аналогий, здравых соображений. В результате получают класс эвристических алгоритмов. Алгоритмы этого класса реализуют эвристическое эквивалентирование. При эвристическом эквивалентировании быстродействие алгоритма повышается, но оценить качество полученного решения невозможно; нельзя даже сказать, является ли оно тупиковым, т. е. далее не упрощаемым. Дерево решений, каждая вершина которого соответствует исходному заданию, а дуга - варианту перебора или преобразованию, изображено на рис. 6.1, *б.*

Добиться максимального быстродействия комбинаторных алгоритмов без порождения всех эквивалентных решений, т. е. использовать малый объем памяти, позволяют алгоритмы третьего класса, реализующие семантическое эквивалентирование.

При семантическом эквивалентировании не только известна полная система аксиом, но и конструктивный критерий, который позволяет: для задачи анализа проверить истинность предиката $P_0(\Psi_a)$:

$$P_0(\Psi_a) = \begin{cases} 1, \text{ если модель } \Psi_a \text{ обладает заданным свойством,} \\ 0 \text{ в противном случае;} \end{cases}$$

для задачи синтеза связать две различные абстракции - модели Ψ_a и Ψ_b - в единую систему с помощью предиката функциональной целостности $P_o(\Psi_a, \Psi_b)$:

$$P_0(\Psi_a, \Psi_b) = \begin{cases} 1, \text{ если преобразование } \Psi_a \to \Psi_b \text{ существует при взаимно однозначном соответствии между элементами моделей } \Psi_a \text{ и } \Psi_b, \\ 0 \text{ в противном случае.} \end{cases}$$

Общее, что характеризует модели Ψ_a и Ψ_b и что отличает их от других, - это смысл преобразования $\Psi_a \to \Psi_b$. Другими словами, смысл преобразования $\Psi_a \to \Psi_b$ — это свойство языковых выражений,

инвариантное относительно их модельных представлений. Изучением смысла занимается логическая семантика, которая является разделом металогики. Под семантикой, как правило, понимают дескриптивную (описательную) семантику, изучающую связь между знакосочетаниями формализованного языка и их интерпретациями (толкованиями) в терминах той системы понятий, формализацией которых является данный язык. Здесь под семантикой понимают изучение интерпретации одного формализованного языка в категориях другого, причем оба языка являются формализацией одной системы понятий. Этот вид семантики будем называть *проективным.* В частном случае, когда оба языка совпадают и исследуется выполнимость определенного свойства изучаемой модели, проективную семантику будем называть рефлексивной.

Рефлексивная семантика позволяет решать задачи анализа моделей. Проективная семантика преобразования $\Psi_a \rightarrow \Psi_b$ позволяет вычислять экстремальное значение функционала качества $\varphi(\Psi_b)$ решения и строить соответствующую оптимальную модель Ψ_b не строя все эквивалентные модели $\{\Psi_{b_i}\}$, что намного уменьшает трудоемкость алгоритмов.

Для определения проективной семантики преобразования $\Psi_a \rightarrow \Psi_b$ необходимо:

1. Найти числовые характеристики $\{v_i\}$ модели Ψ_b, однозначно определяющие значение $\varphi(\Psi_b)$.
2. Установить свойства S_b модели Ψ_b , при наличии которых вычислимы $\{v_i\}$.
3. Выявить свойства S_a модели Ψ_a, однозначно определяющие свойства S_b модели Ψ_b .
4. Найти числовые характеристики модели Ψ_a, обладающей свойствами S_a, однозначно определяющие $\varphi(\Psi_b)$.

Таким образом, наличие свойств S_a позволяет однозначно вычислить $\varphi(\Psi_b)$ без фактического построения Ψ_b .

Для нахождения рефлексивной семантики анализа модели Ψ_a необходимо :

1. Выявить свойства S_a модели Ψ_a, однозначно определяющие предикат $P_0(\Psi_a)$.
2. Найти числовые характеристики модели Ψ_a , определяющие наличие свойств S_a.

В обоих случаях основным моментом является установление свойств S_a модели Ψ_a , которые определяют истинность предиката P_o (Ψ_a) или $P_0(\Psi_a, \Psi_b)$. Эти свойства модели Ψ_a будем искать в виде свойств отсутствия запрещенных фигур, образующих основу критерия выпол-

нения свойств S_a. Знание запрещенных фигур позволяет эффективно решать задачи анализа и конструктивно вычислять функционал $\varphi(\Psi_b)$ без генерации всех эквивалентных моделей $\{\Psi_{b_i}\}$ (см. рис. 6.1, в) при решении задач синтеза. **Поиск проективных и рефлексивных семантик, основой которых являются запрещенные фигуры, и их изучение отнесем к разделу металогики и будем называть *конструктивной семантикой.*** Взаимоотношение всех трех семантик преобразования иллюстрирует рис. 6.2.

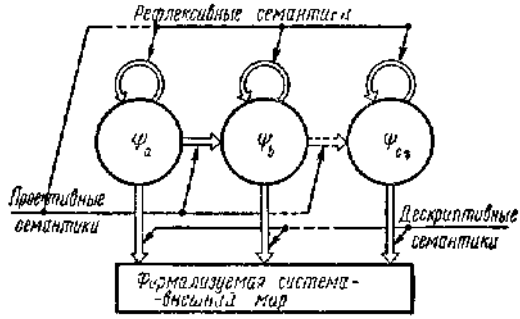


Рис. 6.2

Проблему поиска запрещенных фигур называют *характеризацион-ной проблемой.* **Эта проблема определяется классом рассматриваемых моделей $\mathbf{K}_a = \{\Psi_a\}$ и характеризуемым свойством S_a, определяющим предикат $P_o(\Psi_a)$ или $P_0(\Psi_a, \Psi_b)$.** Для решения характеризационной проблемы требуется определить множество запрещенных фигур $\mathbf{K}_З = \{\Psi_i\}$, т.е. таких моделей Ψ_i, отсутствие которых в данной модели $\Psi_i \in \mathbf{K}_a$ является необходимым и достаточным условием того, что Ψ_a обладает свойством S_a, и при этом никакая из моделей $\Psi_i \in \mathbf{K}_З$ не присутствует в другой запрещенной фигуре — модели $\Psi_j \in \mathbf{K}_З$.

Формализуем понятие *отсутствия* и *присутствия* одной модели в другой. На множестве моделей $\mathbf{K}_a$ можно задать отношение упорядочения $P_п$ такое, что $(\Psi_i, \Psi_j) \in P_п$, если Ψ_i присутствует в Ψ_j. Отношение $P_п$ называется *отношением подчинения,* а модель Ψ_i — *подчиненной* модели Ψ_j. Отношение подчинения моделей служит обобщением известного отношения *быть подмоделью.* Модель Ψ_i является подмоделью модели Ψ_j, если получена из Ψ_j удалением

некоторых элементов носителя и сигнатуры. Харакгеризационная проблема с заданным отношением подчинения $Р_п$ на классе моделей конкретизируется в характеризационную задачу. Поскольку на классе моделей $\mathbf{К}_а$ может быть задано много отношений упорядочения, характеризационную проблему можно рассматривать как целое множество характеризационных задач, каждая из которых имеет собственное решение в виде множества запрещенных фнгур.

От выбора отношения подчинения в решении характеризационной задачи многое зависит: и компактность множества запрещенных фигур, и разрешимость этой задачи вообще. Принципиальным является тот факт, что **характеризационная проблема всегда разрешима.** Нужно только правильно выбрать отношение подчинения и соответственно получить постановку разрешимой характеризационной задачи. Покажем, каким должно быть отношение подчинения.

Теорема 6.1 (принцип локальности).

Множество запрещенных фигур $К_З = \{\Psi_{з_i}\}$ *для класса моделей* $\mathbf{К}_a = \{ \Psi_a \}$ *с отношением подчинения* $P_п$ *и свойства* S_a *существует (характеризационная задача разрешима) тогда и только тогда, когда справедливо, что всякая модель* Ψ_i *подчиненная модели* Ψ_j *, обладающей свойством* S_a*, также обладает им.*

Допустим, что множество запрещенных фигур существует. Тогда, по определению запрещенной фигуры, никакая модель Ψ_a , обладающая свойством S_a, не имеет запрещенной фигуры в качестве подчиненной. С другой стороны, пусть отношение подчинения таково, что моделям Ψ_a , которым присуще свойство S_a, подчинены только модели, которые им также обладают (рис. 6.3).

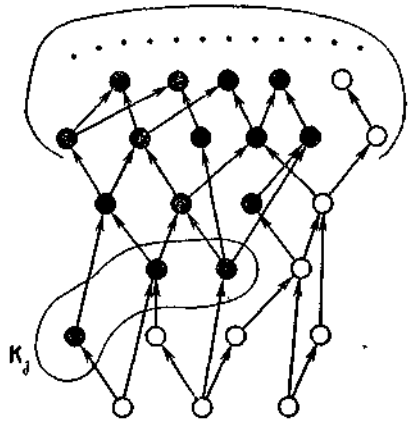


Рис. 6.3

В этом случае, по определению запрещенной фигуры, множество запрещенных фигур $\mathbf{K}_З$ образуется из минимальных элементов отношения упорядочения $P_П$ на множестве $\mathbf{K}_a \setminus \check{\mathbf{K}}_a$, где $\check{\mathbf{K}}_a \subset \mathbf{K}_a$ — подкласс моделей Ψ_a, обладающих свойством S_a.

В качестве примера рассмотрим проблему характеризации двудольных графов. Задачи определения двудольности графа и преобразования графа в двудольный имеют много приложений: от проектирования надежных автоматов до построения эффективных информационных систем с распараллеливанием обработки. Пусть, например, библиотечная информационная система реализуется на ЭВМ, имеющей два дисковых устройства (рис. 6.4, а).

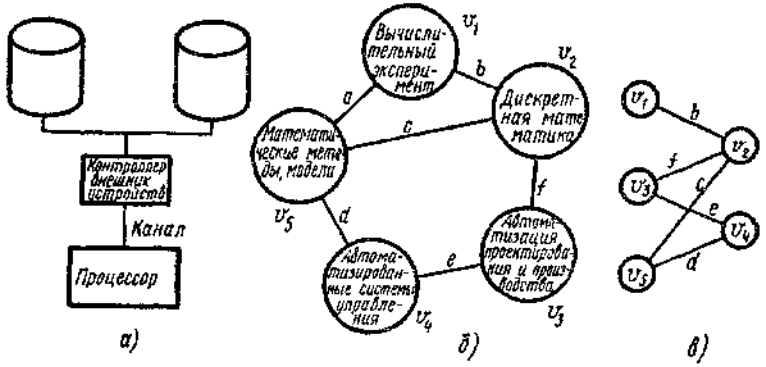


Рис. 6.4

Для достижения максимальной производительности информационно-поисковой системы необходимо так декомпозировать базу данных на две части, чтобы за счет одновременной установки головок чтения-записи на двух дисковых устройствах получалось минимальное время обработки запроса. Файлам базы данных поставим в соответствие вершины графа. Две вершины смежны, если оба соответствующих файла необходимы для ответа на запрос некоторого типа (рис. 6.4, б). Оптимальное распределение файлов по дискам определяется разбиением множества вершин графа на два множества, внутри которых как можно меньше ребер (что соответствует достижению максимального параллелизма). Эта задача сводится к выявлению проективной семантики преобразования графов в двудольные с минимальным удалением ребер. Семантику определит решение характеризационной проблемы двудольности графов (рис. 6.4, в). Класс $\mathbf{K}_a$ состоит из моделей, сигнатура которых образуется одним бинарным симметричным антирефлексивным отношением. Свойство S_a - *быть двудольным.* Исследуем два отношения подчинения: $P^1_{\text{п}}$ - *быть частичным подграфом* и $P^2_{\text{п}}$ - *быть сводимым графом. Сводимым* называется граф G_a, получающийся из G_b в результате последовательного склеивания несмежных вершин (объединения их в одну) с соответствующим объединением их окрестностей. Оба отношения удовлетворяют принципу локальности. Следовательно, в обоих случаях характеризационная задача разрешима. Диаграммы отношений подчинения приведены на рис. 6.5, *а, б.*

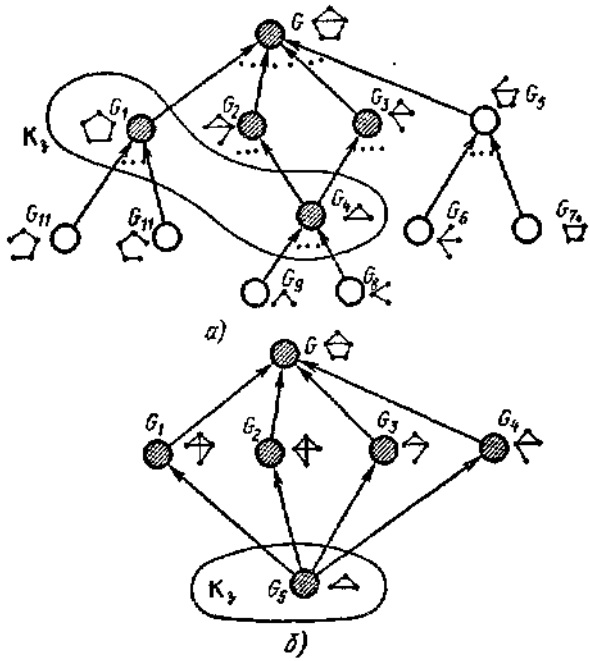


Рис. 6.5

В случае отношения подчинения $P^1_{\text{п}}$, множеством запрещенных фигур является множество нечетных циклов, в случае отношения подчинения $P^2_{\text{п}}$ — множество полных графов порядка больше двух.

Итак, всегда можно решить характеризационную проблему, а следовательно, получить множество запрещенных фигур. Это множество определяет конструктивный семантический критерий для решения задачи анализа. Более того, знание множеств запрещенных фигур используется в методе семантического эквивалентирования в задачах синтеза. Основополагающим принципом семантического эквивалентирования является то, что знание конкретных запрещенных моделей, присутствующих в модели Ψ_a и мешающих ей обладать свойством S_a, позволяет определить те локальные структуры, преобразование которых необходимо для того, чтобы получить модель $\tilde{\Psi}_a$, которой присуще свойство S_a. При этом осуществляется только минимальный (неизбежный) перебор вариантов преобразований, т. е. вычислительный процесс в смысле трудоемкости улучшить нельзя.

При семантическом эквивалентировании дерево решений (см. рис. 6.1, *в)* состоит из двух звезд. Первая звезда соответствует преобразованию $\Psi_a \to \tilde{\Psi}_a$, $P_0(\tilde{\Psi}_a, \Psi_b) = 1$, вторая —$\tilde{\Psi}_a \to \Psi_b$. При поиске минимального решения необходим обход всех ветвей первой звезды, во второй звезде достаточно взять любую ветвь, так как с точки зрения минимальности решения, определяемого значением $\varphi(\Psi_b)$, все ветви второй звезды равнозначны.

Рассмотрим подробно процесс семантического эквивалентирования. Основу этого процесса составляют способы преобразования запрещенных фигур в эквивалентные разрешенные. Эквивалентность определяется смыслом преобразования $\Psi_a \to \Psi_b$. Как правило, способом преобразования запрещенной фигуры в разрешенную является удаление, введение или расщепление элемента носителя или сигнатуры либо переход к некоторой подчиненной модели.

Для семантического эквивалентирования при преобразовании графа в двудольный возможно несколько способов преобразования запрещенных фигур — циклов нечетной длины. В случае использования преобразования для вложения графа в гиперкуб при проектировании надежных автоматов запрещенная фигура преобразуется в разрешенную введением на ребро нечетного числа вершин (строго говоря, здесь не одно преобразование, а множество). Если преобразование используется для оптимальной декомпозиции базы данных, то способ преобразования заключается в удалении ребра. Принципиальным является то, что способ преобразования запрещенной фигуры в разрешенную существует всегда, когда преобразование $\Psi_a \to \Psi_b$ вообще имеет смысл. Действительно, для каждой модели Ψ_a существует эквивалентная модель $\tilde{\Psi}_a$, обладающая свойством S_a. Следовательно, существует преобразование Ψ_a в $\tilde{\Psi}_a$, а любое преобразование модели Ψ_a в модель, обладающую свойством S_a, обязательно преобразует запрещенные фигуры в разрешенные. Локализация глобального преобразования Ψ_a в Ψ_b на запрещенную фигуру $\Psi_3 \subset \Psi_a$включает ее преобразование в разрешенную. Таким образом, способ преобразования запрещенной фигуры в разрешенную существует всегда.

В общем случае возможно множество преобразований запрещенной фигуры в разрешенную. Пусть $R_i = \{r_{i_j}\}$ - множество способов преобразований запрещенной фигуры $\Psi_i \in K_3$ (т. е. для всякого j $r_{i_j}(\Psi_i)$ — разрешенная фигура). Построим множество базисных способов преобразований $R_i^0 \subset R_i$, т. е. такое минимальное по

включению множество $\{r_j^0\}$, что для любого r_{i_j} найдется последовательность преобразований из R^0_i, переводящая Ψ_i в $r_{i_j}(\Psi_i)$.

Рассмотрим способы преобразования запрещенных фигур для преобразования графа в линейный, что важно в задачах организации данных в информационно-поисковых системах и базах данных. **Основными характеристиками размещения данных в памяти являются объем занимаемой памяти и время доступа.** При представлении информационно-поисковой системы графом **объем памяти определяется мощностью носителя, а время доступа — временем считывания слов модели.** Одним из способов оптимальной организации данных в памяти является линейное размещение, которое предполагает такое линейное (полное) упорядочение объектов данных, что ответ на каждый запрос есть цепочка непосредственно следующих в этом упорядочении объектов данных. Поставим в соответствие объектам данных элементы носителя графа, ответам на запросы — слова. Граф называется линейным, если допускает линейное размещение элементов носителя, при котором все слова представляются цепочками. Таким образом, для получения оптимальной организации данных необходимо преобразовать граф в линейный и, следовательно, решить характеризационную проблему линейности графа.

Не конкретизируя вид запрещенных фигур (этот вопрос подробно рассмотрен далее), отметим, что возможны два способа преобразования запрещенных фигур в разрешенные:

1) расщепление элемента носителя x на x и x' с соответствующей заменой x на x' в некоторых словах, в которые входил x;

2) расщепление слова М, т. е. замена его на два слова.

Можно показать, что эти способы образуют базисное множество способов преобразований запрещенных, фигур в разрешенные.

Проиллюстрируем способы преобразования запрещенных фигур в разрешенные на примере графа (рис. 6.6, *а),* представляющего гипотетическую библиотечную информационную систему, включающую информацию о следующих книгах:

X_1 —*Ржевский В. В.* Процессы открытых горных работ.— М.: Недра, 1978;

x_2 — *Самарский А. А.* Теория разностных схем.- М: Наука, 1977;

x_3 — *Марчук Г. И.* Методы вычислительной математики.— М.: Наука, 1977;

x_4 — Основы автоматизации управления производством /Под ред. *Макарова И. М.* — М.: Высшая школа, 1983;

она рассчитана на следующие запросы:

M_1: Книги по вычислительным методам. Ответ $\{x_2, x_3\}$;

M_2: Книги по автоматизации процессов. Ответ $\{x_1, x_4\}$;

M3: Книги авторов (редакторов), фамилии которых начинаются с А до М. Ответ {x3, x4};
M4: Книги авторов (редакторов), фамилии которых начинаются с Н до Я. Ответ {x1, x2}.

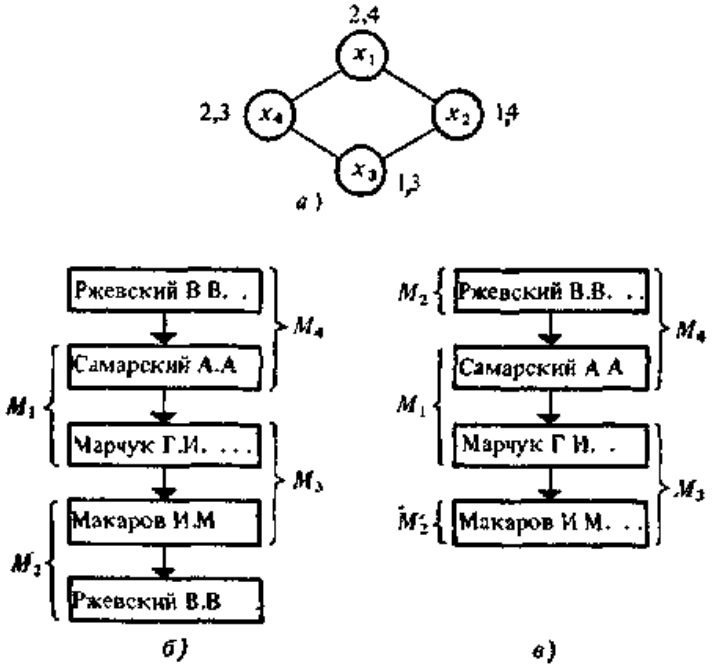


Рис. 6.6

Данный мограф не является линейным. Можно показать, что каждый элемент носителя и каждое слово входят в запрещенную фигуру. Расщепление элемента носителя или слова должно преобразовать мограф в линейный. Рассмотрим, например, расщепление элемента x_1. Преобразованный мограф является линейным и представляется линейным упорядочением, приведенным на рис. 6.6, *б*. При расщеплении любого слова, например M_2, мограф также становится линейным и представляется линейным упорядочением, приведенным на рис. 6.6, *в*. В первом случае увеличивается объем памяти, занимаемый объектами данных, во втором — увеличивается среднее время ответа на запросы.

Каждый способ преобразования r_{i_j} характеризуется стоимостью

c_{i_j}. Функционал качества преобразования $\varphi(\Psi_b)$ определяется стоимостью произведенных преобразований запрещенных фигур в разрешенные.

Для выполнения преобразования $\dot{\Psi}_a \to \tilde{\Psi}_a$ неизбежно преобразование каждой запрещенной фигуры в разрешенную, поэтому выберем для каждой запрещенной фигуры $\Psi_{з_i} \subset \Psi_a$ один из способов ее преобразования; их совокупность определит глобальное преобразование $\dot{\Psi}_a \to \tilde{\Psi}_a$. Такое преобразование в общем случае может не достигать экстремума функционала $\varphi(\Psi_b)$, даже если для каждой $\Psi_{з_i}$ выбран способ преобразования с наименьшей стоимостью. Это обусловлено тем, что возможен способ преобразования $\Psi_{з_i}$, пусть имеющий не наименьшую стоимость, но преобразующий заодно и другую запрещенную фигуру $\Psi_{з_i}$. Более того, взаимосвязь способов преобразования запрещенных фигур может оказаться очень сложной, поэтому простой выбор способов преобразования по одному на каждую $\Psi_{з_i} \subset \Psi_a$ не обеспечивает достижения экстремума функционала качества $\varphi(\Psi_b)$.

Существует процедура семантического эквивалентирования, основанная на построении семантической таблицы. Столбцам этой таблицы соответствуют запрещенные фигуры, присутствующие в модели; строкам — способы преобразования. Элемент (*i, j)* таблицы равен 1, если *i*-е преобразование превращает *j*-ю запрещенную фигуру в разрешенную, и 0 в противном случае. Покрытие столбцов строками семантической таблицы определяет минимальное по включению множество преобразований, которое необходимо выполнить для получения из Ψ_a модели $\tilde{\Psi}_a$, обладающей свойством S_a. Учитывая, что каждый способ преобразования может иметь собственную стоимость, делаем вывод, что для достижения экстремума $\varphi(\Psi_b)$ необходимо найти минимальное по стоимости покрытие семантической таблицы. Иногда в семантической таблице строкам соответствуют запрещенные фигуры, а столбцам — их преобразования в разрешенные. Поскольку это не принципиально, будем использовать оба варианта построения семантической таблицы.

Рассмотрим задачу семантического эквивалентирования графа, изображенного на рис. 6.4, *б,* в двудольный. Функционал качества — минимум удаленных ребер. Запрещенные фигуры (циклы нечетной длины) образованы следующими множествами ребер: $\Psi_{з_1} = \{a, b, c\}$,

$\Psi_{3_2} = \{a, b, d, e, f\}$. Способ преобразования — удаление ребра — имеет стоимость 1; конкретный способ преобразования обозначим указанием удаляемого ребра. Семантическая таблица имеет следующий вид:

Таблица 6.1

Ψ_{3_1}	Ψ_{3_2}	
1	1	*a*
1	1	*b*
1		*e*
	1	*d*
	1	*e*
	1	*f*

Минимальным является, например, покрытие $\pi = \{a\}$. Следовательно, удаляя ребро *a,* получаем двудольный граф (рис. 6.4, *в);* при этом достигнут минимум $\varphi(\Psi_b) = 1$.

В общем случае способ преобразования может быть связан не с элементами носителя или сигнатуры модели, а с некоторыми ее компонентами. Поэтому в общем случае семантическое эквивалентирование предполагает построение иерархической системы таблиц некоторой глубины *k,* определяемой количеством уровней в способах преобразований. Покрытие столбцов строками первой таблицы указывает, какие компоненты запрещенных фигур должны быть изменены при приведении модели Ψ_a к интерпретируемому в терминах модели Ψ_b виду. Определение подкомпонент, которые надо изменить для изменения найденных на предыдущем шаге компонент, сводится к покрытию второй таблицы, построенной аналогично, и т. д. до построения *k*-й таблицы, строки или столбцы которой соответствуют элементам носителя или сигнатуры, которые необходимо изменить при приведении модели Ψ_a к интерпретируемому в терминах модели Ψ_b виду (рис. 5.7).

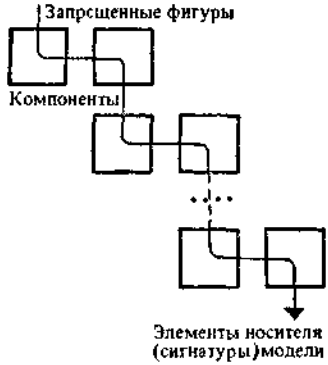


Рис. 6.7

Для определения минимального числа элементов носителя или сигнатуры, соответствующих элементам *k*-й таблицы, необходима генерация всех множеств, каждое из которых состоит из покрываемых строк (столбцов) в последней, *k*-й, таблице. Эта генерация соответствует перебору всех покрытий *(k* — 1)-й таблицы. Для получения всех покрытий *(k* — 1)-й таблицы необходима генерация всех покрытий *(k* — 2)-й таблицы и т. д. Таким образом, для нахождения минимального решения необходим перебор всех сочетаний покрытий первых *(k* — 1) таблиц. Эта процедура принципиально содержит перебор.

Таким образом, определение сложности решения является минимальным по трудоемкости стандартным процессом. Этот процесс на много порядков менее трудоемок, чем процесс фактической генерации всех эквивалентных структур при поиске минимального решения синтаксическим эквивалентированием.

Способы преобразования запрещенных фигур, будучи по отношению к ним однозначными, по отношению к модели в целом могут быть как однозначными, так и неоднозначными. Например, способ преобразования запрещенных фигур для двудолыюсти графа — нечетных циклов,— основанный на удалении ребра, однозначен и для модели в целом. Рассмотрим способ преобразования нечетных циклов, заключающийся в расщеплении одной из вершин *v* на две несмежные вершины *v* и *v'*, каждая кз которых инцидентна одному из двух ребер *хну* расщепляемой вершины (рис. 6.8, *а)* (обозначим способ преобразования как *v* (*x*, *y)).*

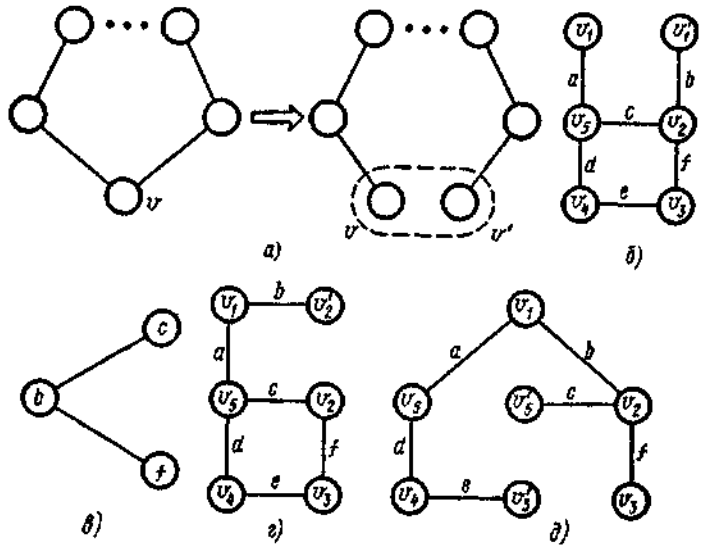
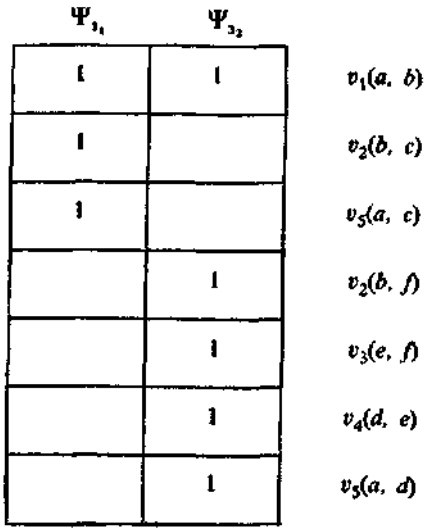


Рис. 6.8

Этот способ преобразования используется в семантическом эквивалентировании при декомпозиции информационной системы с требованием максимального быстродействия, т. е. максимального параллелизма обработки информации. Такое требование обусловливает размещение всех разделов данных, соответствующих смежным вершинам, на разных дисках, т. е. приводит к дублированию некоторых разделов. Данный способ преобразования запрещенной фигуры для модели в целом (графа) не однозначен. Дело в том, что преобразование запрещенной фигуры при расщеплении вершины *v,* инцидентной ребрам *x* и *y,* фиксирует только то, что новая вершина *v* инцидентна ребру *x,* а новая вершина *v′* — ребру *y.* Для графа в целом это означает расщепление вершины *v* на *v* и *v′* с соответствующим разбиением окрестности Г*v,* при котором *x* и *y* попадают в разные новые окрестности. Рассматриваемый способ преобразования не указывает распределения всех вершин по новым окрестностям. Семантическая таблица для графа (см. рис. 6.4,6) при таком способе преобразования имеет следующий вид:



В случае неоднозначных способов преобразований необходимо проверить все покрытия семантической таблицы. Рассмотрим три покрытия:

$$\pi = \{v_1(a, b)\}, \quad \pi_2 = \{v_2(b, c), v_2(b, f)\}, \quad \pi_3 = \{v_5(a, c), v_3(e, f)\}.$$

Первое покрытие, имеющее минимальную мощность, дает минимальное решение (рис. 6.8, *б).* Но и второе покрытие, имеющее неминимальную мощность, также определяет минимальное решение. Оба преобразования, входящие в π_2, расщепляют v_2. Ответ на вопрос, сколько новых вершин дают эти расщепления вместе, дают построение и раскраска специального графа, который построен на множестве ребер, инцидентных v_2 (рис. 6.8, в). Раскраска графа $\{b\}$, {c, *f*} определяет число вершин после расщепления и разбиение окрестности Γv_2. Полученное решение минимально (рис. 6.8, г). Преобразование, соответствующее π_3 (рис. 6.8, *д),* не минимально.

Таким образом, если применяются способы преобразований, неоднозначные для модели в целом, то необходимо для каждого покрытия строить специальные графы и определять их минимальную раскраску. Поэтому особенно актуальны оценки хроматического числа графа, которые позволяют быстро отсеять большую часть покрытий, соответствующих графам с большим хроматическим числом. Важны

также «быстрые» методы раскраски графов. В целом семантическое эквивалентирование при минимальном (неизбежном) переборе позволяет получить абсолютно оптимальное решение.
На основе семантического эквивалентирования получаем двухконтурную структуру быстродействующего пакета прикладных программ (рис. 6.9), в котором с помощью модуля «Характер» происходит автоматическая настройка на оптимальную стратегию при проведении преобразования $\Psi_a \rightarrow \Psi_b$.

Рис. 6.9

Это придает «интеллектуальность» пакету прикладных программ и позволяет отнести подобные пакеты к классу систем искусственного интеллекта.
Ранее нами были рассмотрены характеризационные проблемы вложения графа в плоскость, в булево пространство, характеризации параллельно-последовательной структуры диаграммы Хассе и раскраски графов. Остановимся теперь на характеризационных проблемах частичного упорядочения мографа, проектировании логических схем в функциональных несвязных базисах, проектировании многовыходных логических схем, разложении графа переходов в частичное декартово произведение, а также характеризационных проблемах, возникающих при проектировании оптимальных размещений данных в информационных системах. Характеризация моделей позволяет выявить объективные причины, определяющие сложность решения и трудоемкость его поиска.

6.2. Характеризация частичного упорядочения мографа

Прежде чем установить причины, приводящие к тому, что одному первичному терму сопоставляют две (и более) вершины в структурном

графе (расщепляют первичные термы), рассмотрим преобразование мографа в структурный граф при задании тупиковой ДНФ булевой функции

$$f(x_1, x_2, \ldots, x_5) = \underbrace{x_1\bar{x}_3\bar{x}_5}_{1} \vee \underbrace{x_1x_4}_{2} \vee \underbrace{x_2x_3}_{3} \vee$$

$$\vee \underbrace{\bar{x}_1x_3\bar{x}_5}_{4} \vee \underbrace{x_3\bar{x}_4x_5}_{5} \vee \underbrace{x_2x_4x_5}_{6},$$

которая определяется моделью $\Psi_a = \langle M, S_2, S_3 \rangle$;

$M = \{x_1, \bar{x}_1, x_2, x_3, \bar{x}_3, x_4, \bar{x}_4, x_5, \bar{x}_5\}$,

$S_2 = \{\{x_1, x_4\}, \{x_2, x_3\}\}$,

$S_3 = \{\{x_1, \bar{x}_3, \bar{x}_5\}, \{\bar{x}_1, x_3, \bar{x}_5\}, \{x_3, \bar{x}_4, x_5\}, \{x_2, x_4, x_5\}\}$.

Мограф $G^M(\Psi_a)$ изображен на рис. 6.10, *а*.

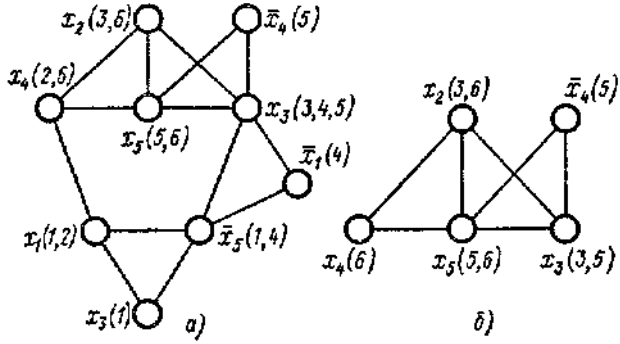


Рис. 6.10

Рассмотрим подмограф $(G^M)'$ (рис. 6.10,б), задающий третью, пятую и шестую простые импликанты. Сопоставим третьей импликанте x_2x_3 цепь $v(x_2) \leqslant v(x_3)$. Пятой простой импликанте $x_3\bar{x}_4x_5$ сопоставим цепь $v(x_3) \leqslant v(\bar{x}_4) \leqslant v(x_5)$, шестой импликанте $x_2x_4x_5$ - цепь $v(x_2) \leqslant v(x_4) \leqslant v(x_5)$. При таком задании отношения упорядоченности получаем, что $v(x_2) \leqslant v(x_3) \leqslant v(\bar{x}_4)$, т. е. $v(x_2)$ сравнима с $v(\bar{x}_4)$, $v(x_2) \gtrless v(\bar{x}_4)$,

что противоречит заданию.

Чтобы ответить на вопрос, является ли такое задание отношения $\leq$ просто неудачным, не зная семантики преобразования, необходимо построить полностью синтаксическое дерево этого преобразования, висячим вершинам которого соответствуют диаграммы H_i, рассмотрев

при этом не только эти три простые импликанты, но и весь заданный мограф $G^M(\Psi_a)$. Число висячих вершин этого дерева равно $2! \cdot 2! \cdot 3! \cdot 3! \cdot 3! \cdot 3! = 5184$. Фактическим построением такого количества диаграмм можно убедиться, что не существует способа задания отношения $\leq$ в рассматриваемом мографе $G^M(\Psi_a)$, при котором имеет место взаимно однозначное соответствие между первичными термами $x_i^{\sigma_i}$ и вершинами диаграмм Н такое, что каждая цепь $v(x_a) \leqslant v(x_b) \leqslant \ \ldots \ \leqslant v(x_c)$ взаимно однозначно соответствует простой импликанте $x_a \cdot x_b \ldots x_c$.

Трудоемкость выявления таких противоречий, а следовательно, и построения абсолютно минимальной схемы значительно меньше, если известна семантика преобразования, определяемая распределением запрещенных фигур.

Теорема 6.2. *Запрещенными фигурами* Q_A, $Q_Б$ *преобразования мографа* G^M *в диаграмму Н являются подмографы типов* А к Б (рис. 6.11).

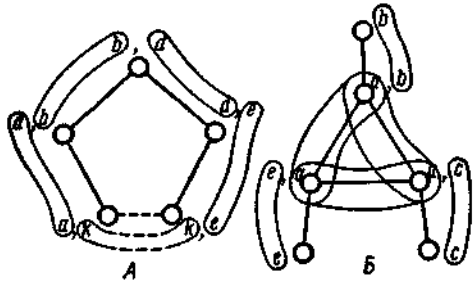


Рис. 6.11

Подмограф типа *А* представляет собой цикл нечетной длины с циклической перестановкой весов. Подмограф типа Б является треугольником с висячими вершинами, каждая из которых имеет общий вес со смежной висячей вершиной и все вершины треугольника имеют общий вес, при этом висячие вершины могут совпадать как попарно, так и все вместе с соответствующим объединением их идентификаторов. Наличие одной из таких фигур в мографе делает принципиально невозможным задание отношения $\leq$ при преобразовании $G^M \rightarrow H$ без расщепления первичных термов в мографе G^M, причем такое расщепление без знания семантики преобразования $G^M \rightarrow H$ происходит «вслепую». При этом кроме ликвидации запрещенных фигур расщепляются лишние первичные термы, что уменьшает оптимальность получаемого решения. Как будет показано

далее, в структурном графе Н, построенном по мографу G^M без выделения запрещенных фигур, произведены лишние расщепления. Выделение запрещенных фигур типов А и Б сводится к задаче нахождения циклов нечетной длины в мографе с последующей проверкой распределения весов на них. При выделении циклов нечетной длины в мографе вершины с одним весом не рассматривают как вершины, которые могут соответствовать только висячим вершинам фигур типа Б.

В приведенном выше примере для выделения циклов нечетной длины достаточно рассмотреть подграф (G^M)", изображенный на рис. 6.12, *а.*

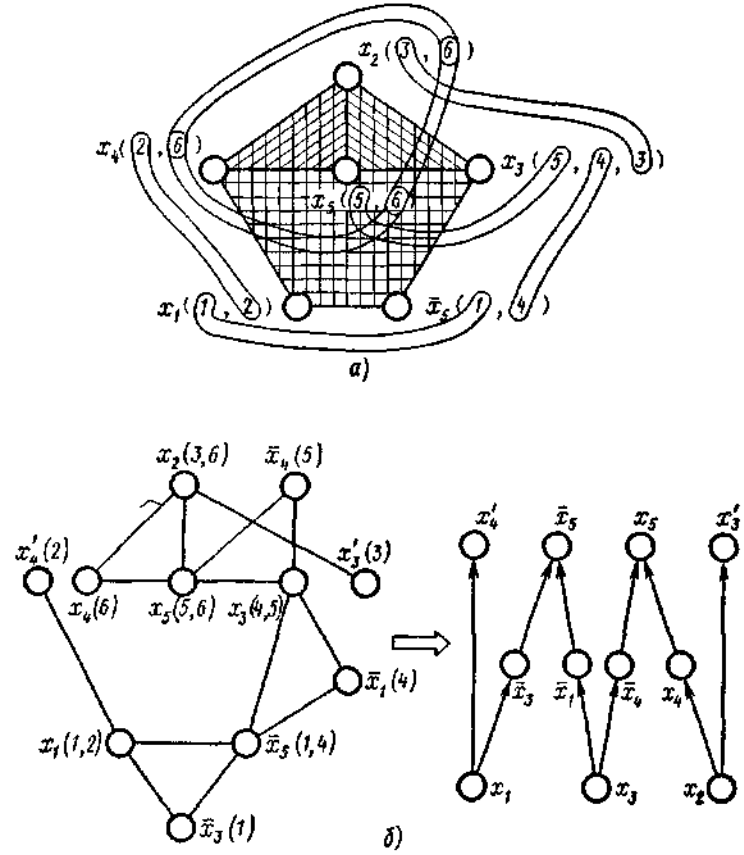


Рис. 6.12

Анализируя этот мограф, устанавливаем, что запрещенными фигурами типа А являются следующие подмографы:

$Q_1 = \{x_2(3, 6), x_5(6, 5), x_3(5, 3)\}$,

$Q_2 = \{x_2(3, 6), x_4(6, 2), x_1(2, 1), \bar{x}_5(1, 4), x_3(4, 3)\}$,

$Q_3 = \{x_5(6, 5), x_3(5, 4), x_5(4, 1), x_1(1, 2), x_4(2, 6)\}$;

запрещеннойфигурой типа Б является подмограф $\{\{x_2, x_4, x_5\}, \{x_1, x_4\}, \{x_2, x_3\}, \{x_5, x_4\}\}$.

Фигуру типа Б в дальнейшем будем задавать соответствующим треугольником. В данном случае четвертая запрещенная фигура имеет следующий вид: $Q_4 = \{x_4(2, 6), x_5(5, 6), x_2(3, 6)\}$.

Основное свойство запрещенных фигур типов А и Б заключается в том, что при расщеплении любой вершины фигуры типа *А* и любой вершины треугольника фигуры типа Б эти подмографы перестают быть запрещенными фигурами. Таким образом, подобные расщепления - это способы преобразования запрещенных фигур типа А и Б в разрешенные. Процедура семантического эквивалентирования мографа в частично упорядоченный стандартная. В семантической таблице строкам соответствуют запрещенные фигуры типа А или Б, столбцам — расщепляемые вершины мографа. Семантическая таблица для рассматриваемого примера приведена в табл. 6.3.

Таблица 6.3

Q_i	$x_1(1,2)$	$x_2(3,6)$	$x_3(4,5)$	$x_3(3,4)$	$x_3(3,5)$	$x_4(2,6)$	$x_5(5,6)$	$\bar{x}_5(1,4)$
Q_1	0	1	0	0	1	0	1	0
Q_2	1	1	0	1	0	1	0	1
Q_3	1	0	1	0	0	1	1	1
Q_4	0	1	0	0	0	1	1	0

Для уменьшения трудоемкости определения покрытия семантической таблицы будем удалять поглощающиеся строки и столбцы. В данном случае правила поглощения следующие.

Столбец α *поглощается столбцом* β, если не найдется третьего столбца, взвешенного той же буквой, что и столбец а, и векторное произведение столбцов α и β равно столбцу α.

Строка ос *поглощается строкой β*, если векторное произведение этих строк равно строке β .

В рассматриваемом случае первый и восьмой столбцы поглощаются шестым. Вычеркивая поглощающиеся столбцы, имеем шесть покрытий семантической таблицы:

$\{x_2(3, 6), x_4(2, 6)\}$, $\{x_2(3, 6), x_5(5, 6)\}$, $\{x_2(3, 6),$

$x_3(4, 5)\}, \{x_4(2, 6), x_5(5, 6)\}, \{x_3(3, 4), x_5(5, 6)\}, \{x_3(3, 5), x_4(2, 6)\}.$

Каждое из этих покрытий порождает два расщепления, следовательно, мощность расширения носителя модели Ψ_a равна 2 и абсолютно минимальная реализация рассматриваемой тупиковой ДНФ содержит 11 ключей $L = |M_a| + |\Delta M_a| = 11$.

Для определенности рассмотрим последнее покрытие и будем отличать букву x_3 в третьем слове от x_3 в пятом слове, добавляя штрих в верхнем индексе: x'_3. В дальнейшем это переименование будем называть *штриховкой* буквы в соответствующем слове. Аналогично произведем штриховку буквы x_4 во втором слове. В результате получаем модель $\tilde{\Psi}_a$:

$$\tilde{\Psi}_a = \langle M_a, S_2, S_3 \rangle,$$

$$M_a = \{x_1, \bar{x}_1, x_2, x_3, x'_3, \bar{x}_3, x_4, x'_4, \bar{x}_4, x_5, \bar{x}_5\},$$

$$S_2 = \{\{x_1, x'_4\}, \{x_2, x'_3\}\},$$

$$S_3 = \{\{x_1, \bar{x}_3, \bar{x}_5\}, \{\bar{x}_1, x_3, \bar{x}_5\}, \{x_3, \bar{x}_4, x_5\}, \{x_2, x_4, x_5\}\};$$

она эквивалентна заданной и интерпретируется в терминах частично упорядоченного множества (рис. 6.12, б):

$$x_1 \leftrightarrow v(x_1),\ x_2 \leftrightarrow v(x_2),\ x_3 \leftrightarrow v(x_3),$$

$$\bar{x}_1 \leftrightarrow v(\bar{x}_1),\ x'_3 \leftrightarrow v(x'_3),\ \bar{x}_3 \leftrightarrow v(\bar{x}_3),$$

$$x_4 \leftrightarrow v(x_4),\ x'_4 \leftrightarrow v(x'_4),\ \bar{x}_4 \leftrightarrow v(\bar{x}_4),$$

$$x_5 \leftrightarrow v(x_5),\ \bar{x}_5 \leftrightarrow v(\bar{x}_5),$$

при этом

$$x_1\bar{x}_3\bar{x}_5 \leftrightarrow v(x_1) \leqslant v(\bar{x}_3) \leqslant v(\bar{x}_5),$$

$$x_1x'_4 \leftrightarrow v(x_1) \leqslant v(x'_4),\ x_2x'_3 \leftrightarrow v(x_2) \leqslant v(x'_3),$$

$$\bar{x}_1x_3\bar{x}_5 \leftrightarrow v(x_3) \leqslant v(\bar{x}_1) \leqslant v(\bar{x}_5),$$

$$x_3\bar{x}_4x_5 \leftrightarrow v(x_3) \leqslant v(\bar{x}_4) \leqslant v(x_5),$$

$$x_2x_4x_5 \leftrightarrow v(x_2) \leqslant v(x_4) \leqslant v(x_5).$$

Если в покрытие первой таблицы вошли хотя бы две буквы лексикографически одинаковые, то слова, в которых происходит их штриховка, определяются раскраской графа G. Граф G построен на множестве слов, содержащих эти буквы, и две вершины в нем смежны, если буква должна расщепляться по этим словам.

Таким образом, знание семантики преобразования $G^M \to H$ позволило заменить перебор 5184 фактически строящихся диаграмм H_i перебором шести покрытий семантической таблицы. В общем случае трудоемкость при знании семантики уменьшается в комбинаторное число раз по сравнению с количеством всех эквивалентных решений. Рассмотрим примеры, иллюстрирующие теорему 6.2.

Пример 6.1.

Мограф $G^M = \langle V, S_2, S_3 \rangle$, $V = \{a, b, c, d, e\}$, $S_2 = \{\underbrace{\{a, e\}}_{1}, \underbrace{\{b, d\}}_{2}$,

$\underbrace{\{d, c\}}_{3}\}$, $S_3 = \{\underbrace{\{a, b, c\}}_{4}\}$, содержит запрещенные

фигуры $Q_1 = \{b(2, 4), c(4, 3), d(3, 2)\}$;

$Q_2 = \{a(4, 1), b(4, 2), c(4, 3)\}$ (рис. 6.13, *а),* которые порождают семантическую таблицу (табл. 6.4).

Таблица 6.4

Запрещенные фигуры	$a(1,4)$	$b(2,4)$	$c(3,4)$	$d(2,3)$
Q_1	0	1	1	1
Q_2	1	1	1	0

Табл. 6 4 имеет три покрытия:

$\pi_1 = \{b(2, 4)\}, \pi_2 \{c(3, 4)\}, \pi_3 = \{a(1, 4), d(2, 3)\}$; с помощью них исходный мограф можно привести к интерпретируемому виду тремя способами (рис. 6.13. б).

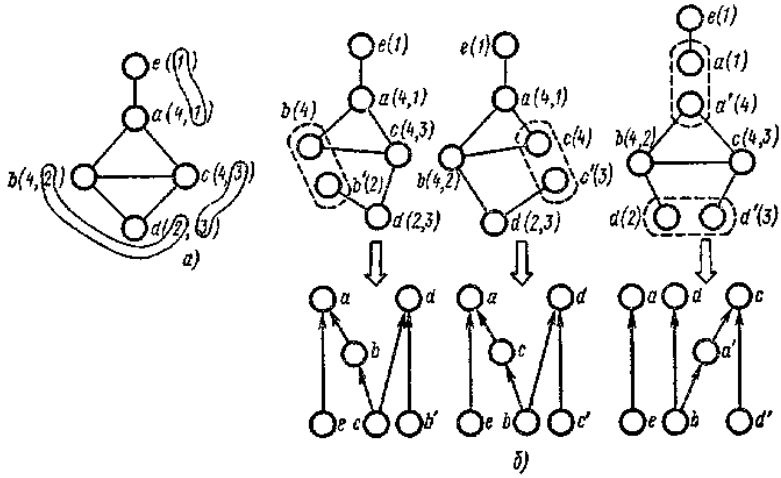


Рис. 6.13.

Пример 6.2. Мограф $G^M = \langle V, S_2, S_3 \rangle$,

$$V = \{a, b, c, d\},\ S_2 = \{\underbrace{\{a, d\}}_{1},\ \underbrace{\{b, d\}}_{2},\ \underbrace{\{c, d\}}_{3}\},\ S_3 = \{\underbrace{\{a, b, c\}}_{4}\},$$

солепжит той заггоепенные фигуры типа А и одну типа Б:

$Q_1 = \{a(1, 4), b(2, 4), b(2, 4), c(3, 4)\}$;

$d(1, 2)\}$, $Q_2 = \{b(2, 4), c(3, 4), d(2, 3)\}$, $Q_3 = \{a(1, 4), c(3, 4), d(1, 3)\}$, $Q_4 = \{a(1, 4),$

они порождают семантическую таблицу (табл. 6.5).

Таблица 6.5

Запрещенные фигуры	$a(1,4)$	$b(2,4)$	$c(3,4)$	$d(1,3)$	$d(1,2)$	$d(2,3)$
Q_1	1	1	0	0	1	0
Q_2	0	1	1	0	0	1
Q_3	1	0	1	1	0	0
Q_4	1	1	1	0	0	0

Табл. 6.5 имеет шесть покрытий:

$$\pi_1 = \{a(1, 4), b(2, 4)\},\ \pi_2 = \{a(1, 4), c(3, 4)\},$$
$$\pi_3 = \{b(2, 4), c(3, 4)\},\ \pi_4 = \{a(1, 4), d(2, 3)\},$$
$$\pi_5 = \{b(2, 4), d(1, 3)\},\ \pi_6 = \{c(3, 4),$$

d(l, 2)}; каждое из них порождает диаграммуХассе сложности 6. Для определенности выберем первое покрытие, произведя штриховку буквы *a* в первом слове, буквы *b* - во втором. В результате получаем мограф

$$\tilde{G}^M = \langle \tilde{V}, S_2, S_3 \rangle,$$

$$\tilde{V} = \{a, a', b, b', c, d\},\ S_2 = \{\underbrace{\{a', d\}}_{1},\ \underbrace{\{b', d\}}_{2},\ \underbrace{\{c, d\}}_{3}\},\ S_3 = \{\underbrace{\{a, b, c\}}_{4}\},$$

эквивалентирующий заданный и интерпретируемый в категориях частично упорядоченного множества $\langle V, \leqslant \rangle$:

$$\{a', d\} \leftrightarrow v(a') \leqslant v(d),\ \{b', d\} \leftrightarrow v(b') \leqslant v(d),$$
$$\{c, d\} \leftrightarrow v(c) \leqslant v(d),\ \{a, b, c\} \leftrightarrow v(c) \leqslant v(a) \leqslant v(b).$$

Рассмотримустойчивость запрещенных фигур в зависимости от граничных условий, т. е. от условий пересечения запрещенной фигуры с остальной частью мографа.

Условие неустойчивости запрещенной фигуры типа А. *Композиция запрещенной фигуры типа* А *и слова, носители которых совпадают, не должна нарушать условий интерпретируемости мографа в категориях частично упорядоченного множества.*

Рассмотрим мограф

$$G^M = \langle V,\ S_3\rangle,\ V = \{x_1,\ \bar{x}_2,\ x_2,\ x_3,\ x_4,\ x_5\},$$

$$S_3 = \{\underbrace{\{x_1,\ \bar{x}_2,\ x_4\}}_{1},\ \underbrace{\{x_2,\ x_3,\ x_4\}}_{2},\ \underbrace{\{x_1,\ x_3,\ x_5\}}_{3},\ \underbrace{\{x_2,\ x_3,\ x_5\}}_{4}\},$$

содержащий запрещенную фигуру типа А:

$$Q_A = \{x_1\,(1,\ 3),\ x_3\,(3,\ 2),\ x_4\,(2,\ 1)\}.$$

При преобразовании, этого мографа в структурный граф необходимо расщепить одну из букв x_1, x_3, x_4. Для определенности расщепляем x_4 во втором слове, в результате получаем мограф

$$\tilde{G}^M = \langle \tilde{V},\ S_3\rangle,\ \tilde{V} = \{x_1,\ \bar{x}_2,\ x_2,\ x_3,\ x_4,\ x_4',\ x_5\},$$

$$S_3 = \{\underbrace{\{x_1,\ \bar{x}_2,\ x_4\}}_{1},\ \underbrace{\{x_2,\ x_3,\ x_4'\}}_{2},\ \underbrace{\{x_1,\ x_3,\ x_5\}}_{3},\ \underbrace{\{x_2,\ x_3,\ x_5\}}_{4}\},$$

интерпретируемый в категориях частично упорядоченного множества (рис. 6.14, *а).*

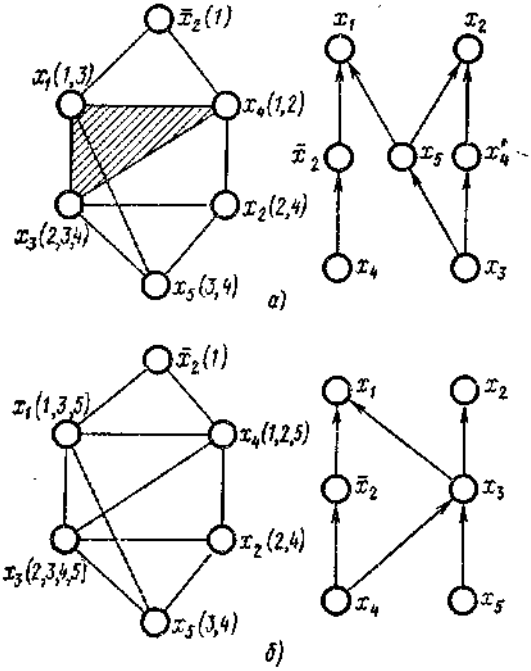


Рис. 6.14

Введем понятие пары внешне неустойчивых вершин. Парой внешне неустойчивых вершин относительно подмографа $(G^M)'$ мографа G^M называется пара вершин v_k, v_l, взвешенных первичными термами $\bar{x}_i$, x_i, $v_k(\bar{x}_i)$, $v_l(x_i)$ таких, что объединение вершин, смежных с $v_k(\bar{x}_i)$ согласно идентификатору α, и вершин, смежных с $v_l(x_i)$ согласно идентификатору β, включает носитель подмографа $(G^M)'$.

В рассматриваемом примере имеется пара внешне неустойчивых вершин $v(\bar{x}_2)$, $v(x_2)$ относительно носителя выделенной запрещенной фигуры Q. Роль α играет идентификатор 1, роль β — либо 2, либо 4. Согласно соотношению Порецкого

$$A\bar{x} \vee Bx = A\bar{x} \vee Bx \vee AB,$$

в мограф, не нарушая эквивалентности задания им булевой функции, можно добавить слово AB.

В случае наличия в мографе пары внешне неустойчивых вершин слово AB имеет вид

$$(\alpha \setminus \bar{x}_i) \bigcup (\beta \setminus x_i).$$

В данном случае это $x_1x_3x_4$, так как мограф является дистрибутивной решеткой.

Добавление слова $\{x_1, x_3, x_4\}$ вызывает неустойчивость запрещенной фигуры $\{x_1(1, 3), x_3(2, 3), x_4(1, 2)\}$, в результате мограф

$$G^M = \langle V, S_3 \rangle, \quad V = \{x_1, \bar{x}_2, x_2 \; x_3, x_4, x_5\},$$

$$S_3 = \{\underbrace{\{x_1, \bar{x}_2, x_4\}}_{1}, \underbrace{\{x_2, x_3, x_4\}}_{2}, \underbrace{\{x_1, x_3, x_5\}}_{3}, \underbrace{\{x_2, x_3, x_5\}}_{4} \underbrace{\{x_1, x_3, x_4\}}_{5}\}$$

является интерпретируемым в категориях частично упорядоченного множества (рис. 6.14, б).

При добавлении слов в случае пары внешне неустойчивых вершин относительно запрещенной фигуры следует рассмотреть и добавление связей, которые могут привести к появлению дополнительных запрещенных фигур. Добавление связей (ребер) в мографе имеет место в случае, когда добавляемое слово строго включает носитель запрещенной фигуры.

Условия неустойчивости запрещенной фигуры типа Б.

1. *Запрещенная фигура типа* Б *неустойчива, если идентификатор слова, взвешивающий висячую вершину, по которому она смежна с вершиной треугольника фигуры, взвешивает еще одну вершину треугольника.*

Рассмотрим мограф

$$G^M = \langle V,\ S_2,\ S_3\rangle,\ V = \{x_1,\ \bar{x}_6,\ x_2,\ x_3,\ x_4,\ x_5\},$$
$$S_2 = \{\underbrace{\{x_1,\ \bar{x}_6\}}_{1},\ \underbrace{\{x_3,\ x_4\}}_{2}\},\ S_3 = \{\underbrace{\{x_1,\ x_3,\ x_5\}}_{3}\ \underbrace{\{x_2,\ x_3,\ x_5\}}_{4}\},$$

удовлетворяющий этому условию. Он не содержит запрещенных фигур: взаимно однозначное соответствие между первичными термами (буквами) графа и вершинами структурного графа, при котором каждое слово взаимооднозначно соответствует пути, имеет следующий вид:

$$\{x_1,\ \bar{x}_6\} \leftrightarrow v(x_1) \leqslant v(\bar{x}_6),\ \{x_3,\ x_4\} \leftrightarrow v(x_4) \leqslant v(x_3),$$
$$\{x_1,\ x_3,\ x_5\} \leftrightarrow v(x_1) \leqslant v(x_5) \leqslant v(x_3),$$
$$\{x_2,\ x_3,\ x_5\} \leftrightarrow v(x_2) \leqslant v(x_5) \leqslant v(x_3).$$

2. *Запрещенная фигура типа* Б *неустойчива, если условие* 1 *выполняется для двух висячих вершин, но устойчива, если оно выполняется для всех трех вершин.*

Рассмотрим мограф

$$G^M_\alpha = \langle V,\ S_3\rangle,\ V = \{a,\ b,\ c,\ d,\ e,\ f\},$$
$$S_3 = \{\underbrace{\{a,\ c,\ d\}}_{1},\ \underbrace{\{a,\ b,\ e\}}_{2},\ \underbrace{\{b,\ c,\ f\}}_{3},\ \underbrace{\{a,\ b,\ c\}}_{4}\},$$

содержащий неустойчивую запрещенную фигуру типа А

$$Q_A = \{a(1,\ 2),\ b(2,\ 3),\ c(1,\ 3)\}$$

и устойчивую запрещенную фигуру типа Б, основанием которой является треугольник с вершинами *a, b, c* и висячими вершинами *d, e, f*. Эта фигура будет неустойчивой при добавлении одного из слов

$$\{b,\ e,\ f\},\ \{c,\ d,\ f\},\ \{a,\ d,\ e\}.$$

Частным случаем частичного упорядочения мограф является транзитивная ориентация графа. Здесь мограф рассматривают без моделизации, т. е. словами полученной модели являются максимально полные подграфы графа, над которым проводится транзитивная ориентация.

Для этого случая имеем три запрещенные фигуры. Первыми двумя фигурами являются фигуры типа $Q_А$ и $Q_Б$, рассматриваемые без моделизации, третьей фигурой - граф G^M_α без моделизации, являющийся композицией неустойчивой запрещенной фигуры типа А и устойчивой фигуры типа Б.

Рассмотрим применение теоремы 6.2 при транзитивной ориентации графов на примере представителя одного из транзитивно неориентируемых семейств графов — графа, дополняющего цикл, длина которого превышает пять, до полного. Рассмотрим цикл, длина которого равна шести. Этот цикл до полного дополняет граф G, изображенный на рис. 6.15, *а*.

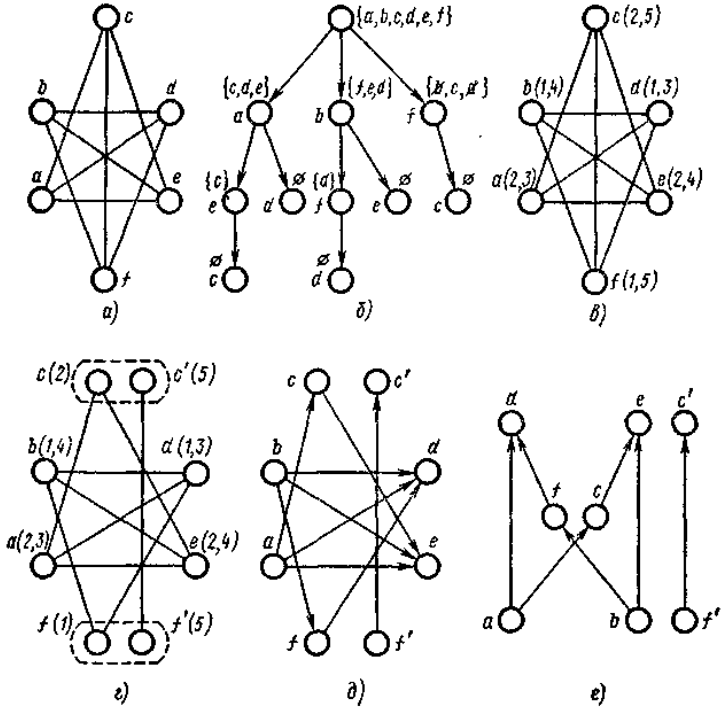


Рис. 6.15

Применяя алгоритм выделения полных подграфов к рассматриваемому графу (рис. 6.15, б), получаем множество полных подграфов $\{\{b, d, f\}, \{a, c, e\}, \{a, d\}, \{b, e\}, \{c, f\}\}$, которое образует сигнатуру мографа (рис. 6.15, в):

$$G^M = \langle V, S_2, S_3 \rangle, \; V = \{a, b, c, d, e, f\},$$

$$S_3 = \underbrace{\{b, d, f\}}_{1}, \underbrace{\{a, c, e\}}_{2}, \; S_2 = \{\underbrace{\{a, d\}}_{3}, \underbrace{\{b, e\}}_{4}, \underbrace{\{c, f\}}_{5}\}.$$

Мограф $G^M = \langle V, S_2, S_3 \rangle$ содержит две устойчивые запрещенные фигуры типа Б:

$Q_{Б1} = \{\{b, d, f\}, \{b, e\}, \{a, d\}, \{c, f\}\}$, $Q_{Б2} = \{\{a, c, e\}, \{a, d\}, \{c, f\}, \{b, e\}\}$.
Расщепляя одну из вершин треугольника каждой запрещенной фигуры, получаем транзитивно ориентируемый граф. Возможны девять способов расщепления. В каждом из них следует расщеплять две вершины, так как эти треугольники не пересекаются. Для определенности выберем вершины *c* и *f* (рис. 6.15, *г).* В результате получаем транзитивно ориентируемый граф (рис. 6.15, *д),* который после удаления транзитивно замыкающих дуг преобразуется в структурный граф (рис. 6. 15, *е).*

6.3. Характеризация выходной связности логических схем

Проектирование многовыходных логических схем в топологических базисах не отличается от проектирования одновыходных схем, если допускается съем информации с элементов, не являющихся минимальными или максимальными, при этом накладывается только одно ограничение: не расщеплять элементы графа, взвешенные выходными буквами f_i. При проектировании многовыходных схем в функциональных базисах, как правило, реализуемую систему булевых функций сводят к одной функции или ищут пересечения единичных областей булевых функций и синтезируют схемы по булевым функциям, описывающим эти пересечения. Окончательная схема представляет собой композицию схем, реализующих поведение заданных булевых функций на пересечениях рабочих областей, и схем-сборок, выходы которых совпадают с выходными каналами искомой схемы. В случае же, когда единичные области не пересекаются, при использовании известных методов имеет место несвязная реализация системы булевых функций. Но именно этот случай часто встречается на практике. В предлагаемом подходе к проектированию логических схем имеем связную реализацию системы булевых функций. Рассмотрим следующий пример. Пусть задана система булевых функций вида

$$f_1 = \underbrace{x_1x_2\bar{x}_3x_4}_{1} \vee \underbrace{\bar{x}_1x_2x_3x_4}_{2};\ f_2 = \underbrace{x_1x_2x_3x_4}_{3} \vee \underbrace{x_1x_2\bar{x}_3\bar{x}_4}_{4}.$$

Преобразуем мограф $G^M(\{f_i\})$, задающий эту систему (рис. 6.16, *а),* в структурный граф $H(\{f_i\})$ так, чтобы максимальные элементы были взвешены буквами, идентифицирующими выходные каналы, и ни одна из вершин, не являющаяся максимальным элементом, не была взвешена выходной буквой f_i .

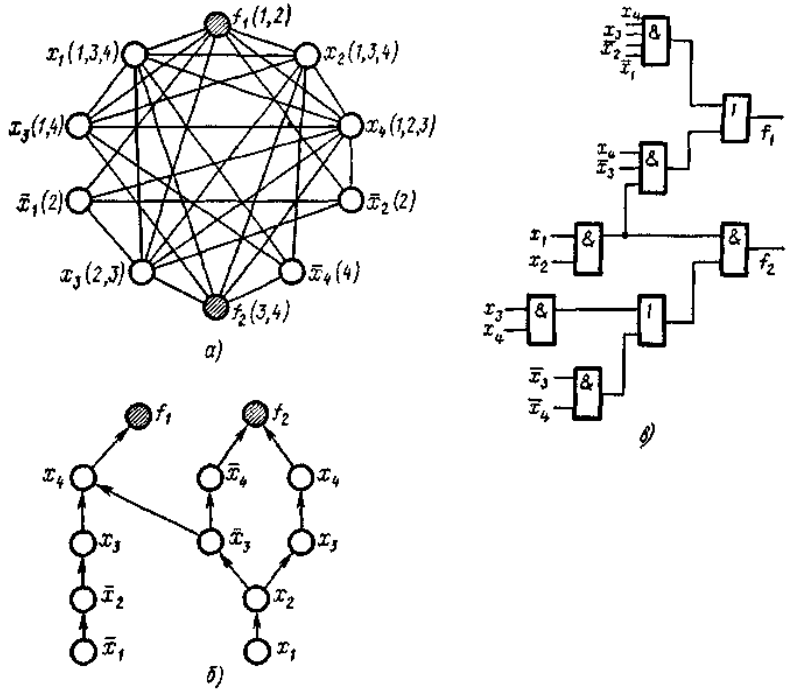


Рис. 6.16

Структурный граф $H(\{f_i\})$ изображен на рис. 6.16, б. С помощью коалгебры графов преобразуем его в логическую схему (рис. 6.16, в). Вершины мографа и структурного графа, определяющие многовыходную логическую схему, будем раскрашивать в два цвета: вершины, взвешенные входными буквами x_i, $\bar{x}_i$, - белым цветом, вершины, взвешенные выходными буквами f_i , - черным (на рисунках черному цвету соответствует штриховка).

При преобразовании мографа $G^M(\{f_i\})$, задающего систему булевых функций, в структурный граф $H(\{f_i\})$ на последний накладывается следующее ограничение: максимальные элементы структурного графа, и только они, должны быть взвешены выходными буквами f_i, которые при этом преобразовании не расщепляются.

Систему булевых функций $f(x) = \{f_i\}$ представим в виде модели

$$\Psi_a = \langle M_a,\ p,\ S_1,\ S_2, \ldots, S_n \rangle,$$

где $M_a = \{m_1, m_2, \ldots, m_n, m_{n+1}, \ldots, m_{n+k}\}$, $S_i \subset M_a^i$, $i = 1, 2, \ldots, n$;

p - одноместный предикат, разбивающий M_a на два подмножества:

$$p = \begin{cases} 0 \text{ на элементах } m_i \text{ при } i = 1, 2, \ldots, n, \\ 1 \text{ на элементах } m_j \text{ при } j = (n+1), \ldots, (n+k). \end{cases}$$

Структурный граф $H(F(x))$ представим в виде модели

$$\Psi_b = \langle M_b, \leqslant, q \rangle,$$

где

$$q = \begin{cases} 1, \text{ если из } p(m_i) \geqslant p(m_j) \text{ следует } m_i \geqslant m_j, \\ 0 \text{ в противном случае.} \end{cases}$$

При исследовании вопроса преобразования модели Ψ_a в модель Ψ_b и построения многовыходного структурного графа следует рассматривать модели со следующими ограничениями:

1) если в модели Ψ_a найдутся два слова μ_α и μ_β такие, что $\mu_\alpha \subset \mu_\beta$, то эти слова заменяют одним словом μ_a;

2) если в слове имеются хотя бы две одинаковые буквы m_a и m_b ($m_a = m_b$), то одну из них заменяют другой;

3) если в модели Ψ_a найдутся хотя бы два слова μ_α и μ_β такие, что

$$\mu_\alpha = \alpha m_r, \ \mu_\beta = \alpha m_t,$$

где $p(m_r) = p(m_t) = 1$, α состоит из букв m_i, $p(m_i) = 0$, то одно из этих слов заменяют другим.

Очевидно, для того чтобы было можно частично упорядочить буквы модели Ψ_a, необходимо, чтобы мограф G^M не содержал запрещенных фигур Q_A, Q_B (см. рис. 6.11). В этом случае возможно частичное упорядочение букв модели Q_A, Q_B без учета предиката q в модели Ψ_b.

Найдем запрещенные фигуры в мографе $G^M(F(x))$, характеризующие фиксирование максимальных (минимальных) элементов в соответствующем структурном графе $H(F(x))$ при преобразовании $G^M(F(x)) \to H(F(x))$. Для определенности будем фиксировать максимальные элементы, которые соответствуют выходным каналам проектируемой логической схемы.

Условие частичного упорядочения букв модели Ψ_a, в котором учитываются заданные максимальные элементы, устанавливает следующая теорема.

Теорема 6.3. *Между буквами модели*

$$\Psi_a = \langle M_a, p, S_1, S_2, \ldots, S_n \rangle,$$

мограф G^M которой не содержит мографов Q_A, Q_B, существует отношение частичной упорядоченности тогда и только тогда, когда мограф G^M не содержит модельных подграфов Q_E (рис. 6.17).

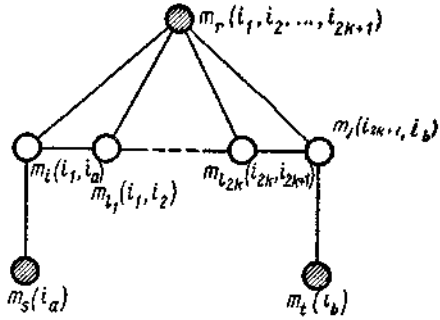


Рис. 6.17

Сложность логических схем в основном определяется сложностью соответствующего структурного графа Н. Следовательно, структурная минимизация булевой функции *f* определяется распределением запрещенных фигур Q_A, Q_B в графе $G^M(f)$, системы булевых функций $\{f_i\}$ - распределением запрещенных фигур Q_A, Q_E, Q_E в графе $G^M(\{f_i\})$. Таким образом, минимальной логической схеме будут соответствовать те ДНФ булевых функций, в которых необходимо произвести минимальное количество расщеплений первичных термов для того, чтобы образовавшийся граф был интерпретируем в категориях структурных графов.

Рассмотрим точную структурную минимизацию булевой функции *f*, которая сводится к покрытию семантической таблицы глубины два (рис. 6.18) и, если необходимо, раскраске графов, соответствующих покрытиям второй таблицы.

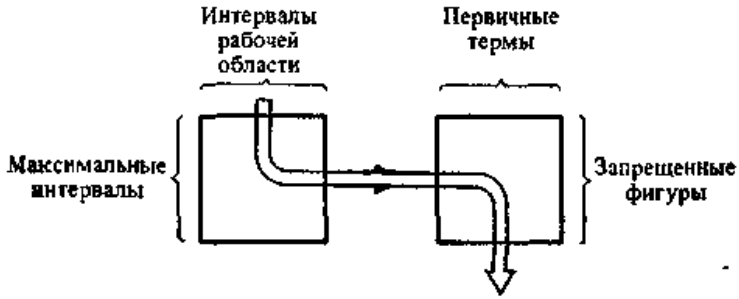


Рис. 6.18

Первая таблица формируется на основе покрытия таблицы различий для практических булевых функций. Покрытие первой таблицы

порождает тупиковую ДНФ, которой соответствует мограф G^M. Преобразование его в частично упорядоченный, т. е. интерпретируемый структурным графом Н, выполняется с помощью покрытия второй таблицы. Рассмотрим пример.

Пример 6.3. Определим сложность абсолютно минимального структурного графа (диаграммы), реализующего булеву функцию $f(x_1, x_2, x_3, x_4) = \vee (0, 1, 2, 4, 9, 11, 13)$, и равную нулю на остальных наборах.

Выделим максимальные интервалы и построим для рассматриваемой функции таблицу Квайна (табл. 6.6).

Таблица 6.6

Номера строк	Максимальные интервалы	Единичные точки						
		0000	0001	0010	0100	1001	1011	1101
1	000 –	v	v					
2	00 – 0	v		v				
3	0 – 00	v		–	v			
4	– 001		v		–	v		
5	10 – 1					v	v	
6	1 – 01					v	–	v

Подчеркнутая галочка в таблице Квайна соответствует обязательному максимальному интервалу. (Максимальный интервал является обязательным, если найдется единичная точка, принадлежащая этому и только этому интервалу.) Множество обязательных максимальных интервалов образуют ядро покрытия.

Находим тупиковые ДНФ заданной функции, покрывая столбцы таблицы Квайна строками таблицы. Имеем два покрытия: первая, вторая, третья, пятая, шестая строки и вторая, третья, четвертая, пятая, шестая строки. Этим двум покрытиям соответствуют ТДНФ функции f вида

$$f'(x_1, x_2, x_3, x_4) = \underbrace{x_1\bar{x}_3x_4}_{1} \vee \underbrace{x_1\bar{x}_2x_4}_{2} \vee \underbrace{\bar{x}_1\bar{x}_3\bar{x}_4}_{3} \vee$$

$$\vee \underbrace{\bar{x}_1\bar{x}_2x_4}_{4} \vee \underbrace{\bar{x}_1\bar{x}_2\bar{x}_3}_{5};$$

$$f''(x_1, x_2, x_3, x_4) = x_1\bar{x}_3x_4 \vee x_1\bar{x}_2x_4 \vee \bar{x}_1\bar{x}_3\bar{x}_4 \vee \bar{x}_1\bar{x}_2\bar{x}_4 \vee \bar{x}_2\bar{x}_3x_4.$$

Первой ТДНФ соответствует мограф, изображенный на рис. 6.19, а.

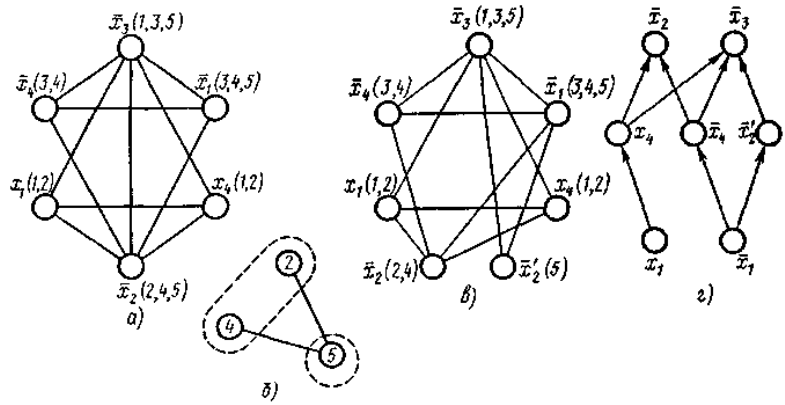


Рис. 6.19

В этом графе имеются циклы нечетной длины с циклической перестановкой весов следующего вида:

$Q_{A1} = \{\bar{x}_2(4, 5), \bar{x}_3(5, 3); \bar{x}_4(3, 4)\}$; $Q_{A2} = \{x_1(1, 2), \bar{x}_2(2, 5), \bar{x}_3(5, 1)\}$; $Q_{A3} = \{x_4(1, 2), \bar{x}_2(2, 5), \bar{x}_3(5, 1)\}$.

ПервойТДНФ соответствует семантическая таблица (табл. 6.7).

Таблица 6.7

Q_i	$x_1(1,2)$	$\bar{x}_2(4,5)$	$\bar{x}_2(2,5)$	$\bar{x}_3(3,5)$	$\bar{x}_3(1,5)$	$\bar{x}_4(3,4)$	$x_4(1,2)$
Q_1	0	1	0	1	0	1	0
Q_2	1	0	1	0	1	0	0
Q_3	0	0	1	0	1	0	1

Одним из минимальных покрытий является $\pi = \{\bar{x}_2(4, 5), \bar{x}_2(2, 5)\}$. Поскольку преобразования, входящие в него, расщепляют лексикографически одинаковые буквы, строим граф на словах {2, 4, 5} (рис. 6.19,б). Этот граф раскрашивается в два цвета: {2, 4} и {5}. Следовательно, достигнуто минимальное расщепление букв. После штриховки буквы $\bar{x}_2$ в пятом слове получаем ДНФ, интерпретируемую в категориях диаграммы Хассе со сложностью 7 (рис. 6.19,в):

$$f'(x_1, x_2, x_3, x_4, \bar{x}_2') = x_1\bar{x}_3x_4 \vee x_1\bar{x}_2x_4 \vee \bar{x}_1\bar{x}_3\bar{x}_4 \vee \bar{x}_1\bar{x}_2\bar{x}_4 \vee \bar{x}_1\bar{x}_2'\bar{x}_3.$$

Рассматривая аналогично остальные покрытия и распределение запрещенных фигур в мографе второго тупиковой ДНФ, получаем, что найденная упорядочиваемая ДНФ является абсолютно минимальной. Следовательно, сложность диаграммы, реализующей эту функцию, также равна 7 (рис. 6.19, г).

Приведем точное решение задачи структурной минимизации системы булевых функций, основанное на использовании запрещенных фигур

преобразования мографа в диаграмму с фиксированными максимальными элементами. Оно заключается в выполнении следующих этапов:
1. С помощью одного из известных методов формируют множество многовыходных простых импликант (МПИ) (многовыходных максимальных интервалов).
2. Строят импликантную таблицу Квайна, в которой каждой строке соответствуют МПИ, столбцу - конституенты единицы (или импликанты) исходных булевых функций $f_i(X) \in F(X)$. При этом конституента единицы (импликанта) столько раз входит в таблицу, сколько функций принимают на нем значение единица.
3. Находят покрытия столбцов строками импликантной таблицы. Таким образом определяются ТДНФ системы булевых функций.
4. Для каждой ТДНФ системы булевых функций, которой соответствует модель Ψ_a, строят решетчатую ДНФ (РДНФ) системы булевых функций минимальной сложности, т. е. осуществляют преобразование $\Psi_a \rightarrow \Psi_b$.
5. Из всех РДНФ системы булевых функций выбирают РДНФ минимальной сложности. Далее строят многовыходной структурный граф *H*.

Для того чтобы удалить все запрещенные фигуры, построим семантическую таблицу *R,* каждой строке которой взаимно однозначно соответствует буква (в скобках указываются идентификаторы двух слов, в которые эта буква входит при образовании запрещенной фигуры), а столбцу - запрещенная фигура Q_A, Q_B, Q_E;

$$r_{ij} = \begin{cases} 1, \text{ если буква, соответствующая } i\text{-й строке, входит в } j\text{-ю запрещенную фигуру;} \\ 0 \text{ в противном случае.} \end{cases}$$

Строкам соответствуют буквы модели $m_i \in M_a$, для которых $p(m_i) = 0$. Тогда покрытие столбцов строками в матрице *R* соответствует множеству букв, которые необходимо расщепить при преобразовании $\Psi_a \rightarrow \Psi_b$.

Проиллюстрируем точный метод минимизации систем булевых функций с учетом их теоретико-структурных свойств.

Пример 6.4. Пусть задана система булевых функций $F(X) = \{f_1(X), f_2(X), f_3(X), f_4(X)\}$, зависящая от пяти переменных (табл. 6.8). Выделяем все МПИ, затем строим таблицу Квайна и в результате получаем две ТДНФ данной системы:

$$F_1(X) = x_4x_5f_1 \vee xx_4f_2 \vee x_3x_5f_3 \vee \bar{x}_1x_2f_4 \vee x_1\bar{x}_2f_4 \vee x_2x_3f_4,$$

Таблица 6.8

x_5	x_4	x_3	x_2	x_1	f_1	f_2	f_3	f_4
0	0	0	0	1	0	0	0	1
0	0	0	1	0	0	0	0	1
0	0	1	0	1	0	0	0	1
0	0	1	1	0	0	0	0	1
0	0	1	1	1	0	0	0	1
0	1	1	0	0	0	1	0	0
0	1	1	0	1	0	1	0	1
0	1	1	1	0	0	1	0	1
0	1	1	1	1	0	1	0	1
0	1	0	0	1	0	0	0	1
0	1	0	1	0	0	0	0	1
1	0	0	0	1	0	0	0	1
1	0	0	1	0	1	0	0	1
1	0	0	1	1	1	0	0	0
1	0	1	0	0	0	0	1	0
1	0	1	0	1	0	0	1	1
1	0	1	1	0	1	0	1	1
1	0	1	1	1	1	0	1	1
1	1	0	0	0	1	0	0	0
1	1	0	0	1	1	0	0	1
1	1	0	1	0	1	0	0	1
1	1	0	1	1	1	0	0	0
1	1	1	0	0	1	1	1	0
1	1	1	0	1	1	1	1	1
1	1	1	1	0	1	1	1	1
1	1	1	1	1	1	1	1	1

Рассмотрим первую из ТДНФ, представляя ее в виде модели Ψ_a^1 :

$M_a = \{x_1, \bar{x}_1, x_2, \bar{x}_2, x_3, x_4, x_5, f_1, f_2, f_3, f_4\}$;

$p(x_1) = p(\bar{x}_1) = p(x_2) = p(\bar{x}_2) = p(x_3) = p(x_4) = p(x_5) = 0,$

$p(f_1) = p(f_2) = p(f_3) = p(f_4) = 1.$

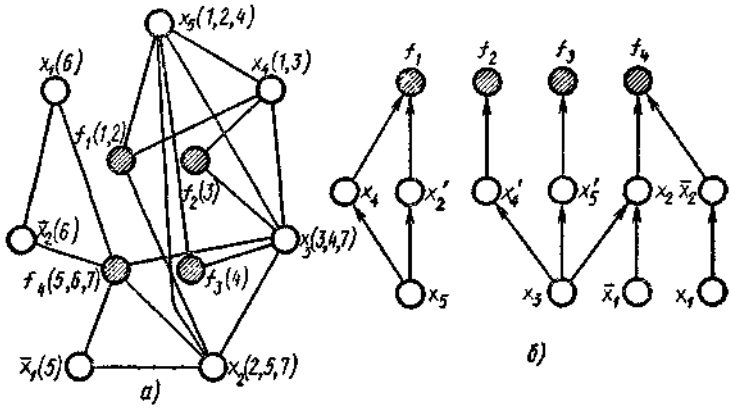


Рис. 6.20

Слова модели Ψ_a^1 соответствуют конъюнкциям функции $F_1(X)$. Построим модельный граф G^M (рис. 6.20, а). Перечислим запрещенные фигуры, содержащиеся в G^M:

$Q_{A1} = \{x_5(1, 4), x_4(1, 3), x_3(3, 4)\}$; $Q_{A2} = \{x_5(2, 4), x_3(4, 7), x_2(2, 7)\}$;

$Q_{E3} \supset \{x_5(1, 4), x_4(1, 3)\}$; $Q_{E4} \supset \{x_5(1, 4), x_3(3, 4)\}$;

$Q_{E5} \supset \{x_5(2, 4), x_3(3, 4)\}$; $Q_{E6} \supset \{x_5(1, 4), x_3(4, 7)\}$;

$Q_{E7} \supset \{x_5(2, 4), x_3(4, 7)\}$; $Q_{E8} \supset \{x_5(2, 4), x_2(2, 5)\}$;

$Q_{E9} \supset \{x_5(2, 4), x_2(2, 7)\}$; $Q_{E10} \supset \{x_4(1, 3), x_3(3, 4)\}$;

$Q_{E11} \supset \{x_4(1, 3), x_3(3, 7)\}$; $Q_{E12} \supset \{x_2(2, 7), x_3(3, 7)\}$;

$Q_{E13} \supset \{x_2(2, 7), x_3(4, 7)\}$.

Покрытиями семантической таблицы (табл. 6.9) являются множества строк: {1, 2, 3, 8}, {1, 2, 3, 5, 6}, {1, 2, 4, 5, 6}, {1, 2, 4, 5, 8}, {3, 4, 6, 7, 8}, {1, 4, 5, 6, 7, 8}.

Построим для лексикографически одинаковых букв графы на множестве слов, в которые входят эти буквы. Раскраска их определяет необходимое расширение (ΔM_a) носителя M_a в каждом конкретном случае.

Таблица 6.9

Буквы	Запрещенные фигуры												
	1	2	3	4	5	6	7	8	9	10	11	12	13
$x_5(1, 4)$	1	0	1	1	0	1	0	0	0	0	0	0	0
$x_5(2, 4)$	0	1	0	0	1	0	1	1	1	0	0	0	0
$x_4(1, 3)$	1	0	1	0	0	0	0	0	0	1	1	0	0
$x_3(3, 4)$	1	0	0	1	1	0	0	0	0	1	0	0	0
$x_3(3, 7)$	0	0	0	0	0	0	0	0	0	0	1	1	0
$x_3(4, 7)$	0	1	0	0	0	1	1	0	0	0	0	0	1
$x_2(2, 5)$	0	0	0	0	0	0	0	1	0	0	0	0	0
$x_2(2, 7)$	0	1	0	0	0	0	0	0	1	0	0	1	1

Мощность расширения носителя $|\Delta M_a|$ для каждого из покрытий соответственно равна 3, 3, 3, 3, 3, 4. Следовательно, минимальная сложность L структурного графа H равна 14 (рис. 6.20,б): $L = |M_a| + |\Delta M_a| = 14$. Рассмотрим модель $\Psi_a^{(2)}$, соответствующую второй ТДНФ:

$M_a = \{x_1, \bar{x}_1, x_2, \bar{x}_2, x_3, x_4, x_5, f_1, f_2, f_3, f_4\}$;

$p(x_1) = p(\bar{x}_1) = p(x_2) = p(\bar{x}_2) = p(x_3) = p(x_4) = p(x_5) = 0$;

$p(f_1) = p(f_2) = p(f_3) = p(f_4) = 1$.

Мограф G_M (рис. 6.21, *а),* определяющий эту модель, содержит запрещенные фигуры следующего вида:

$Q_{A1} = \{x_5(1, 4),\ x_4(1, 3),\ x_3(3, 4)\}$;

$Q_{E2} \supset \{x_5(1, 4),\ x_4(1, 3)\},\ Q_{E3} \supset \{x_5(1, 4),\ x_3(3, 4)\}$;

$Q_{E4} \supset \{x_5(2, 4),\ x_3(3, 4)\},\ Q_{E5} \supset \{x_5(1, 4),\ x_3(4, 7)\}$;

$Q_{E8} \supset \{x_4(1, 3),\ x_3(3, 7)\},\ Q_{E9} \supset \{x_5(2, 4),\ x_2(2, 5)\}$.

Покрывая семантическую таблицу (табл. 6.10), получаем, что минимальное расширение носителя $|\Delta M|$ равно двум. Оно и порождает минимальный структурный граф (рис 6.21,б), определяющий данную систему булевых функций *F(X)*.

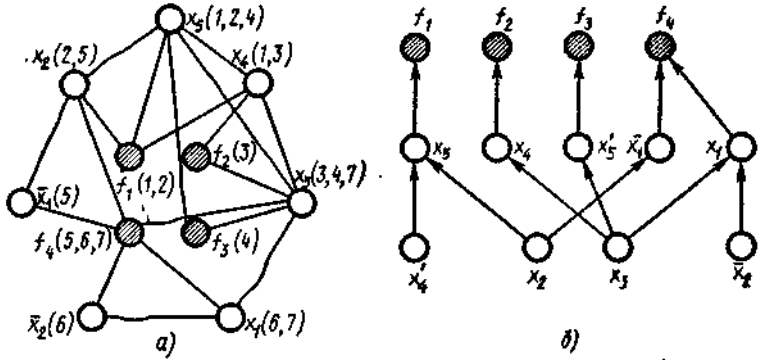


Рис. 6.21

Таблица 6.10

Буквы	Запрещенные фигуры								
	1	2	3	4	5	6	7	8	9
$x_5(1, 4)$	1	1	1	0	1	0	0	0	0
$x_2(2, 4)$	0	0	0	1	0	1	0	0	1
$x_4(1, 3)$	1	1	0	0	0	0	1	1	0
$x_3(3, 4)$	1	0	1	1	0	0	1	0	0
$x_3(3, 7)$	0	0	0	0	0	0	0	1	0
$x_3(4, 7)$	0	0	0	0	1	1	0	0	0
$x_2(2, 5)$	0	0	0	0	0	0	0	0	1

Рассмотрим оптимизирующие функционалы усечения вариантов, построенные на основании учета структуры запрещенных фигур. Исследуем более подробно процесс приведения произвольного мографа к упорядочиваемому виду.

Количество меток каждой вершины v_i мографа равно собственной частоте соответствующей буквы модели, а количество общих меток для пары вершин v_i и v_j (ребра *(i, j))* - значению взаимной частоты соответствующих букв. Следовательно, ребро (*i, j*) мографа можно характеризовать тремя величинами: f_i, f_j и f_{ij}, где f_i — собственная частота буквы *i*, f_j - собственная частота буквы *j*, f_{ij}— взаимная частота букв i и j $(i \neq j)$.

Прослеживая процесс образования запрещенных фигур Q_A, Q_B, Q_E, замечаем, что чем больше сумма $f_i + f_j$ собственных частот букв i и j при данной взаимной частоте f_{ij} или чем меньше взаимная частота f_{ij} букв i и j при данной сумме их собственных частот, тем больше вероятность вхождения букв i, j в подмодели вида Q_A, Q_B, Q_E. Охарактеризуем каждое ребро (i, j) мографа $\Psi(f)$ значением производной от модели, вычисленной на соответствующей паре букв:

$$\frac{\partial \Psi}{\partial S}(i, j) = \frac{f_i - 2f_{ij} + f_j}{f_{ij}}.$$

Тогда чем больше значение производной, тем выше степень неодинакового участия букв в словах; чем больше неоднородность мографа, тем больше вероятность образования подмоделей вида Q_A, $Q_Б$, Q_E в этой модели, тем более сложный (в смысле числа элементов) синтезируемый граф соответствует этой модели. Поэтому максимальный интервал I функции f, которому соответствует полный подграф, будем оценивать средним значением $p(I)$ производной, вычисленной для каждого ребра этого подграфа, т. е. выражением

$$p(I) = \frac{1}{r(r-1)} \left(\sum_{i=1}^{r-1} \sum_{j=i+1}^{r} \frac{f_i - 2f_{ij} + f_j}{f_{ij}} \right), \quad (6.1)$$

где r — ранг простой импликанты I (он равен числу первичных термов, образующих импликанту). Следовательно, оценка (6.1) позволяет синтезировать оптимальные ДНФ булевой функции, учитывая ее теоретико-структурные свойства.

Используя оптимизирующий функционал, представленный в виде оценки (6.1), приведем приближенный алгоритм структурной минимизации булевой функции.

1. Заданную ДНФ функции f задаем в виде матрицы инцидентности Q.

2. Выделяем простые импликанты функции f и записываем их в список I.

3. Строим ядро функции f. Если оно пустое, то переходим к п. 6, в противном случае из списка I вычеркиваем элементы ядра и переходим к п. 4.

4. В матрице Q заменяем единичные интервалы функции f, покрываемые вычеркнутыми из списка I простыми импликантами, на эти простые импликанты.

5. Если любая строка матрицы Q представляет собой простую импликанту, то переходим к п. 8, в противном случае — к п. 6.

6. По матрице Q строим частотную матрицу отношений $F = Q^{\mathrm{T}} \times Q$.
7. Согласно (6.1) оцениваем каждую простую импликанту из списка I. Выбираем простую импликанту с минимальной оценкой, вычеркиваем ее из списка I и переходим к п. 4.
8. Полученная матрица Q задает минимизированную булеву функцию с учетом ее теоретико-структурных свойств.

Пример 6.5. Минимизируем с учетом теоретико-структурных свойств булеву функцию $f(x_1, x_2, x_3, x_4)\|_1 = \vee(2, 3, 4, 5, 8, 9, 11, 12, 14, 15)$; на остальных наборах она равна нулю.

1. Матрица Q имеет вид

$$Q = \begin{array}{c} \begin{array}{cccccccc} x_1 & \bar{x}_1 & x_2 & \bar{x}_2 & x_3 & \bar{x}_3 & x_4 & \bar{x}_4 \end{array} \\ \left\| \begin{array}{cccccccc} 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \end{array} \right\| \end{array} \begin{array}{c} 2 \\ 3 \\ 4 \\ 5 \\ 8 \\ 9 \\ 11 \\ 12 \\ 14 \\ 15 \end{array}.$$

2. Список I простых импликант булевой функции следующий:

0 0 1 – 1 0 0 – 1 1 – 0

0 1 0 – – 0 1 1 1 – 1 1

– 1 0 0 1 0 – 1 1 1 1 –

3. Функция I имеет ядро, состоящее из обязательных простых импликант 010—,001 — и 100—. После вычеркивания элементов ядра список I принимает вид

– 1 0 0 1 1 – 0

– 0 1 1 1 – 1 1

1 0 – 1 1 1 1 –

4. В результате соответствующей замены строк матрица Q имеет следующий вид:

$$Q' = \begin{array}{|cccccccc|l} x_1 & \bar{x}_1 & x_2 & \bar{x}_2 & x_3 & \bar{x}_3 & x_4 & \bar{x}_4 & \\ \hline 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0\,0\,1\,- \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0\,1\,0\,- \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1\,0\,0\,- \\ 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 11 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 12 \\ 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 14 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 15 \end{array}.$$

5. Строки матрицы Q с четвертой по седьмую не соответствуют простым импликантам, поэтому переходим к п. 6.

6. Частотная матрица отношения F' ($F' = (Q')^{\mathrm{T}} \times Q'$), соответствующая матрице Q', такова:

$$F' = \begin{array}{|cccccccc|l} x_1 & \bar{x}_1 & x_2 & \bar{x}_2 & x_3 & \bar{x}_3 & x_4 & \bar{x}_4 & \\ \hline 5 & 0 & 3 & 2 & 3 & 2 & 2 & 2 & x_1 \\ 0 & 2 & 1 & 1 & 1 & 1 & 0 & 0 & \bar{x}_1 \\ 3 & 1 & 4 & 0 & 2 & 2 & 1 & 2 & x_2 \\ 2 & 1 & 0 & 3 & 2 & 1 & 1 & 0 & \bar{x}_2 \\ 3 & 1 & 2 & 2 & 4 & 0 & 2 & 1 & x_3 \\ 2 & 1 & 2 & 1 & 0 & 3 & 0 & 1 & \bar{x}_3 \\ 2 & 0 & 1 & 1 & 2 & 0 & 2 & 0 & x_4 \\ 2 & 0 & 2 & 0 & 1 & 1 & 0 & 2 & \bar{x}_4 \end{array}$$

7. Оцениваем каждую простую импликанту из списка, полученного в п. 3. Согласно (6.1), имеем

$$p(-1000) = p(x_2\bar{x}_3\bar{x}_4) = \frac{1}{3\cdot 2}\left(\frac{4-2\cdot 2+3}{2} + \frac{4-2\cdot 2+2}{2} + \frac{3-2\cdot 1+2}{2}\right) \approx 0{,}91;$$

$$p(-011) \approx 0{,}91;\ p(10--1) \approx 1{,}98;\ p(1-11) \approx 0{,}58;\ p(11-0) \approx 0{,}58;$$

$$p(111-) \approx 0{,}67.$$

Выбираем простую импликанту 1 1 - 0 и переходим к п. 4.

4. В матрице Q' заменяем конституенты единицы функции f, покрываемые импликантой 1 1 — 0, на эту простую импликанту. В результате получаем матрицу

$$Q' = \begin{array}{|cccccccc|c|} \hline x_1 & \bar{x}_1 & x_2 & \bar{x}_2 & x_3 & \bar{x}_3 & x_4 & \bar{x}_4 & \\ \hline 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0\,0\,1\,- \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0\,1\,0\,- \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1\,0\,0\,- \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1\,1\,-\,0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 11 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 15 \\ \hline \end{array}.$$

5. Матрица Q содержит конституенты единицы функции, не являющиеся простыми импликантами. Переходим к п. 6.

6. Частотная матрица отношений F'', соответствующая Q'', имеет вид

$$F'' = \begin{array}{|cccccccc|c|} \hline x_1 & \bar{x}_1 & x_2 & \bar{x}_2 & x_3 & \bar{x}_3 & x_4 & \bar{x}_4 & \\ \hline 4 & 0 & 2 & 2 & 2 & 1 & 2 & 1 & x_1 \\ 0 & 2 & 1 & 1 & 1 & 1 & 0 & 0 & \bar{x}_1 \\ 2 & 1 & 3 & 0 & 1 & 1 & 1 & 1 & x_2 \\ 2 & 1 & 0 & 3 & 2 & 1 & 1 & 0 & \bar{x}_2 \\ 2 & 1 & 1 & 2 & 3 & 0 & 2 & 0 & x_3 \\ 1 & 1 & 1 & 1 & 0 & 2 & 0 & 0 & \bar{x}_3 \\ 2 & 0 & 1 & 1 & 2 & 0 & 2 & 0 & x_4 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & \bar{x}_4 \\ \hline \end{array}.$$

7. По формуле (6.1) оцениваем простые импликанты из списка I, покрывающие оставшиеся коисгитуенты единицы функции f. Имеем:

$$p(-\,0\,1\,1\,1) \approx 0{,}75;\ p(1\,-\,1\,1) \approx 0{,}5;\ p(1\,0\,-\,1) \approx 0{,}91;\ p(1\,1\,1\,-) \approx 1{,}16.$$

Выбираем простую импликанту 1 — 1 1 и переходим к п. 4.

4. После соответствующей замены строк, получаем матрицу

$$Q''' = \begin{array}{|cccccccc|c|} \hline x_1 & \bar{x}_1 & x_2 & \bar{x}_2 & x_3 & \bar{x}_3 & x_4 & \bar{x}_4 & \\ \hline 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0\,0\,1\,- \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0\,1\,0\,- \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1\,0\,0\,- \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1\,1\,-\,0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1\,-\,1\,1 \\ \hline \end{array}$$

5. Любая строка матрицы Q''' соответствует простой импликанте. Переходим к п. 8. П. 5 реализуем, используя следующее положение. Если конституента единицы функции f является простой импликантой, то она содержится в ядре покрытия. Отсюда, если в п. 4 не изменилась ни одна строка матрицы, переходим к п. 8.

8. Матрица Q''' задает булевую функцию f, минимизированную с учетом ее теоретико-структурных свойств.

Используя метод семантического эквивалентирования, установим, насколько удалена полученная ТДНФ функции $f(x_1, x_2, x_3, x_4)$ от ТДНФ функции f, соответствующей минимальному структурному графу.

Найдем все тупиковые ДНФ функции f. Для этого построим импликантную таблицу и определим ее покрытия. Каждую ТДНФ функции f задаем в виде мографа. Выделим запрещенные фигуры типов А и Б и построим семантические таблицы. Затем найдем и оценим их покрытия. В результате окажется, что тупиковая ДНФ, полученная с помощью предложенного алгоритма минимизации, свободного от перебора всех тупиковых ДНФ, соответствует абсолютно минимальному решению.

Рассмотрим структурную минимизацию системы булевых функций $F(X)$. В этом случае каждой многовыходной простой импликанте (МПИ), как и простой импликанте при минимизации одной булевой функции, соответствует полный подграф, вершины которого взвешены идентификатором этой МПИ в мографе G^M. Учет распределения запрещенных фигур в рассматриваемом случае может быть оценен выражением (6.1). При этом производные вычисляют только для дуг, соединяющих вершины, взвешенные буквами m_i и m_j для которых $p(m_i) = p(m_j) = 0$. Для простоты опустим в выражении (6.1) постоянную частотную составляющую и будем оценивать МПИ по следующей формуле:

$$c(I) = \frac{1}{r(r-1)}\left(\sum_{i=1}^{r-1}\sum_{j=i+1}^{r}\frac{f_i + f_j}{f_{ij}}\right). \quad (6.2)$$

Предложим следующую процедуру минимизации систем булевых функций с учетом их теоретико-структурных свойств, основанную на применении оптимизирующего функционала.

1. Задаем систему булевых функций $F(X) = \{f_1(X), f_2(X), \ldots, f_k(X)\}$ множествами M_i^1, M_i^0. При этом

$$f_i(X) = \begin{cases} 1 \text{ на элементах } M_i^1, \\ 0 \text{ на элементах } M_i^0. \end{cases}$$

2. Находим одним из известных способов системы все МПИ булевых функций и заносим их в список I.

3. Строим матрицу Q, каждому столбцу которой соответствует первичный терм, строке — конституента единицы (импликанта) функции $f_i(X) \in F(X)$ и

$$q_{kl} = \begin{cases} 1, \text{ если } x_k \text{ входит в } l\text{-ю конституенту (импликанту);} \\ 0 \text{ в противном случае.} \end{cases}$$

4. Определяем обязательные МПИ в списке *I*. Если они имеются, то переходим к п. 5, вычеркивая обязательные импликанты из списка *I*. В противном случае переходим к п. 7.

5. Корректируем матрицу *Q*, заменяя конституенты единицы, покрываемые вычеркнутыми МПИ, этими МПИ.

6. Если любая строка матрицы *Q* представляет собой МПИ, то переходим к п. 9, в противном случае — к п. 7.

7. По матрице *Q* строим частотную матрицу отношений $F = Q^{\mathrm{T}} \times Q$.

8. Оцениваем каждую МПИ; вычисляя значение *c(I)*. Выбираем МПИ с минимальным значением $c_{\text{мин}}$ *(I)*. Выбранную МПИ вычеркиваем из списка *I* и переходим к п. 5.

9. Матрица *Q* задает минимизированную систему булевых функций *F(X)* с учетом теоретико-структурных свойств.

Пример 6.6. Задана (табл. 6.11) система булевых функций $F_X = \{f_1(X), f_2(X), f_3(X)\}$.

Таблица 6.11

1.

x_4	x_3	x_2	x_1	f_1	f_2	f_3
0	0	0	0	1	0	1
0	0	0	1	0	0	1
0	0	1	0	0	0	1
0	0	1	1	1	0	1
0	1	0	0	1	0	1
0	1	0	1	0	0	1
0	1	1	0	0	0	0
0	1	1	1	1	0	0
1	0	0	0	1	1	0
1	0	0	1	1	0	0
1	0	1	0	0	1	1
1	0	1	1	1	0	1
1	1	0	0	1	1	1
1	1	0	1	1	0	1
1	1	1	0	0	1	1
1	1	1	1	1	0	1

2. Список *I* многовыходных простых импликант системы *F(x)* следующий:

$I_1 = 0\ 0\ -\ -$ (3), $I_2 = 0\ -\ 0$ (3),

$I_3 = -\ -\ 0\ 0$ (1), $I_4 = -\ 0\ 1\ -$ (3), $I_5 = -\ 1\ 0\ -$ (3),

$I_6 = 1\ -\ 0\ -$ (1), $I_7 = 1\ -\ -\ 0$ (2), $I_8 = -\ -\ 1\ 1$ (1),
$I_9 = 1\ -\ -\ 1$ (1), $I_{10} = 1\ -\ -\ 1\ -$ (3), $I_{11} = 1\ 1\ -\ -$ (3),
$I_{12} = 1\ 1\ -\ 1$ (1, 3), $I_{13} = 1\ -\ 1\ 1$ (1, 3), $I_{14} = 1\ 1\ -\ 0$ (2, 3),
$I_{15} = 1\ 1\ 0\ -$ (1, 3), $I_{16} = 1\ -\ 1\ 0$ (2, 3), $I_{17} = -\ 0\ 1\ 1$ (1, 3),
$I_{18} = 1\ -\ 0\ 0$ (1, 2), $I_{19} = -\ 1\ 0\ 0$ (1, 3), $I_{20} = 0\ -\ 0\ 0$ (1, 3),
$I_{21} = 1\ 1\ 0\ 0$ (1, 2, 3).

В скобках указаны номера функций, рабочие точки которых покрывает соответствующий интервал.

3. Составляем матрицу Q, в которой каждая конституента единицы повторяется столько раз, сколько она входит в булевы функции.

4. Обязательной МПИ является импликанта I_8

5. Заменяем в матрице Q строки 0 0 11, 0 111, 10 11, 1111, соответствующие конституентам булевой функции f_1, на многовыходную простую импликанту I_8. В результате получаем матрицу

$$Q^{(1)} = \begin{Vmatrix} \bar{x}_4 & x_4 & \bar{x}_3 & x_3 & \bar{x}_2 & x_2 & \bar{x}_1 & x_1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \end{Vmatrix}.$$

6. Строки матрицы с 1-й по 22-ю не являются МПИ. Переходим к п. 7.

7 Частотная матрица отношений имеет вид

$$F = \begin{array}{c|cccccccc|c} & x_1 & \bar{x}_1 & x_2 & \bar{x}_2 & x_3 & \bar{x}_3 & x_4 & \bar{x}_4 & \\ \hline & 9 & 0 & 4 & 5 & 4 & 4 & 5 & 3 & x_1 \\ & 0 & 14 & 5 & 9 & 7 & 7 & 9 & 5 & \bar{x}_1 \\ & 4 & 5 & 9 & 0 & 3 & 5 & 6 & 3 & x_2 \\ & 5 & 9 & 0 & 14 & 8 & 6 & 8 & 6 & \bar{x}_2 \\ & 4 & 7 & 3 & 8 & 11 & 0 & 8 & 3 & x_3 \\ & 4 & 7 & 5 & 6 & 0 & 11 & 6 & 5 & \bar{x}_3 \\ & 5 & 9 & 6 & 8 & 8 & 6 & 14 & 0 & x_4 \\ & 3 & 5 & 3 & 6 & 3 & 5 & 0 & 8 & \bar{x}_4 \end{array}$$

8. Оцениваем каждую МПИ, вычисляя значение *c(I).* Для импликанты I_3 = — — 0 0 (1) эта оценка минимальна и равна 1,55. Выбираем I_3= — — 0 0 (1) и переходим к п 5.

5. В матрице $Q^{(1)}$ заменяем конституенты единицы, покрытые импликантой I_3 этой импликантой В результате получаем матрицу

$$Q^2 = \begin{array}{|cccccccc|} \hline 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ \hline \end{array}$$

6. Матрица $Q^{(2)}$ содержит строки, не являющиеся МПИ. Переходим к п. 7 и т. д.

В результате получена матрица

$$\begin{array}{c} \bar{x}_4\ x_4\ \bar{x}_3\ x_3\ \bar{x}_2\ x_2\ \bar{x}_1\ x_1 \\ \left\| \begin{array}{cccccccc} 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \end{array} \right\| \end{array},$$

которая соответствует решению

$$F(X) = x_1x_2f_1 \vee \bar{x}_1\bar{x}_2f_1 \vee \bar{x}_2x_4f_1 \vee \bar{x}_1x_4f_2 \vee$$
$$\vee\ \bar{x}_2\bar{x}_4f_3 \vee x_2x_3f_3 \vee x_3x_4f_3.$$

Определим удаленность полученного решения от минимального. Для этого зададим каждую ТДНФ этой системы мографом и определим распределение запрещенных фигур. Семантически эквивалентируя полученные мографы мографами, интерпретируемыми в категориях структурных графов, получаем, что синтезированная ТДНФ системы булевых функций *F(X)* без перебора всех эквивалентных ТДНФ соответствует абсолютно минимальному решению.

6.4. Характеризация разложения графа переходов в частичное декартово произведение

Сложность и надежность автомата во многом определяются кодами внутренних состояний. Одной из актуальных является задача минимизации связей между элементами памяти. Будем считать два элемента памяти *несвязными,* если функция возбуждения одного из них не зависит от состояния другого элемента памяти, и наоборот. В случае несвязности элемента памяти со всеми остальными элементами его функция возбуждения определяется состоянием этого элемента и входным вектором:

$$\varphi_i = \varphi_i(z_i^+,\ X).$$

Будем характеризовать память автомата ее связностью

$$S = \sum_{i=1}^{s} \alpha_i, \quad /$$

где s — общее количество элементов памяти; α_i - количество элементов памяти, отличных от i-го элемента, значения которых необходимы для вычисления функции возбуждения i-го элемента памяти.

Значение связности памяти S, равное нулю, означает, что элементы памяти функционально несвязны и функция возбуждения любого элемента памяти определяется его значением и входным вектором.

Рассмотрим задачу разложения произвольного графа переходов в частичное декартово произведение n функционально несвязных друг с другом сомножителей, каждый из которых соответствует подавтомату. Функциональная связность между блоками возникает при нарушении детерминированности хотя бы в одном из графов переходов G_i. Для описания ситуаций нарушения детерминированности введем граф сцепления $G_{сц}$, каждой вершине которого взаимно однозначно соответствует внутреннее состояние автомата, ребру - пара сцепленных состояний, причем каждое ребро взвешено входными векторами, которые сцепляют соответствующие состояния автомата.

Два состояния S_α, S_β называются *сцепленными,* если найдутся: набор X_i, который переводит состояние S_α в S_γ, $S_\alpha \xrightarrow{X_i} S_\gamma$, и набор X_j, $X_i \subset X_j$, который переводит S_β в S_δ, $S_\beta \xrightarrow{X_j} S_\delta$, такие, что $S_\gamma \neq S_\delta$ и (S_α, S_γ) и (S_β, S_δ) не образуют петли одновременно.

Состояние графа переходов $G = \langle V, (U, X)\rangle$ после его разложения в частичное декартово произведение $G = \prod_i G_i$

$G_i = \langle V_i, (U_i, X)\rangle$

можно охарактеризовать вектором, i-му разряду которого соответствует состояние i-го подавтомата. При разложении каждый подавтомат характеризуется допустимым количеством состояний $|V_i|$. Очевидно, что

$\prod_i |V_i| \geqslant |V|$.

Сцепленным состояниям должны соответствовать векторы, отличные друг от друга в каждом разряде. В противном случае, если в j-м разряде векторы сцепленных состояний совпадают, при подаче на вход автомата вектора X, по которому они сцеплены, в общем случае будет нарушение детерминированности перехода в этом подавтомате.

Следовательно, построение абстрактной параллельной декомпозиции автомата сводится к многокомпонентной раскраске (мультираскраске) графа сцепления, при которой смежным вершинам сопоставляют спектры красок, отличные друг от друга в каждой компоненте.

Разложение графа переходов в частичное декартово произведение не выводит результирующий граф из класса графов переходов.

Запрещенными фигурами этой семантики являются квазиполные графы.

Теорема 6.4. *Если граф сцепления, построенный для каждого из подавтоматов, не содержит квазиполного графа квазиплотности q + 1,то соответствующий автомат разложим (декомпозируем) в частичное декартово произведение функционально несвязных между собой сомножителей — подавтоматов, число состояний каждого из которых не превышает q.*

Таким образом, квазиполные графы — запрещенные фигуры, характеризующие достаточное условие функциональной несвязности подавтоматов при поиске параллельной декомпозиции управляющего автомата. В дальнейшем этот класс запрещенных фигур будем обозначать $Q_ж$. При этом граф сцепления для первого рассматриваемого подавтомата представляет собой граф сцепления, построенный по графу переходов согласно его определению. Граф сцепления *i*-го подавтомата представляет собой граф сцепления первого подавтомата, в который добавлены ребра, соединяющие вершины с одинаковым спектром красок. Добавление этих ребер необходимо для однозначной идентификации состояний автомата.

Рассмотрим построение абстрактной параллельной декомпозиции автомата на основе найденной семантики на следующем примере. Автомат имеет три входных канала. Граф переходов изображен на рис. 6.22, а.

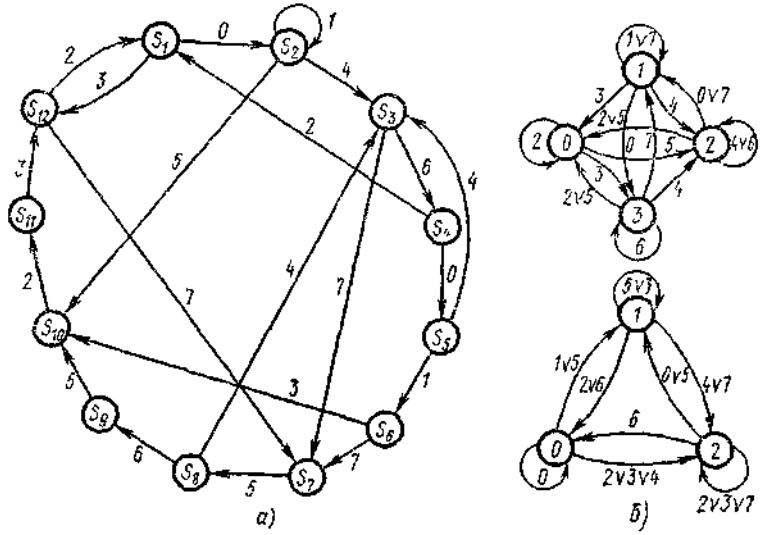


Рис. 6.22

Входные векторы обозначены десятичными эквивалентами соответствующих двоичных наборов. Найдем параллельную декомпозицию автомата в виде двух автоматов с числом внутренних состояний, соответственно равным 3 и 4.

Граф сцепления первого подавтомата приведен на рис. 6.23, *а.*

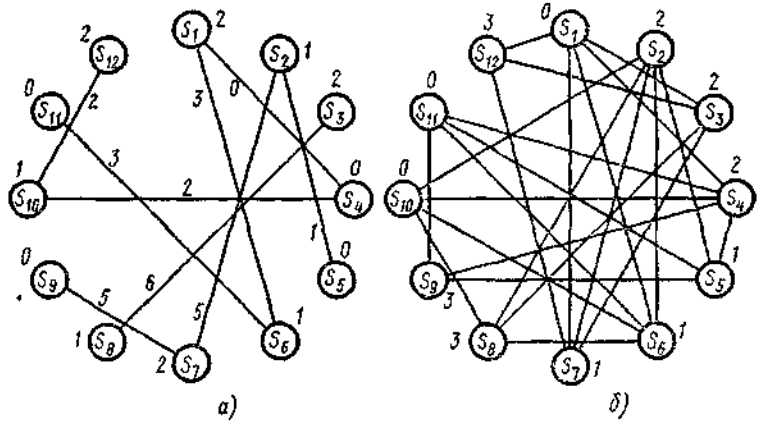


Рис. 6.23

Он не содержит запрещенных фигур; следовательно, возможна раскраска тремя цветами {0, 1, 2}. Каждый цвет соответствует состоянию первого подавтомата. Построим его граф переходов $G_1 = \langle V_1, (U_1, X)\rangle$. В результате раскраски графа $G_{сц}$ множество сосюяний исходного автомата разбивается на три соцветных множества, каждое из которых соответствует состоянию первого подавтомата. Обозначим их соответственно S'_{10}, S'_{11}, S'_{12}. Имеем: $\{S_4, S_5, S_9, S_{11}\} = S'_{10}$; $\{S_2, S_6, S_8, S_{10}\} = S'_{11}$; $\{S_1, S_3, S_7, S_{12}\} = S'_{12}$. Отсюда условиями перехода $\varphi_{i\to j}$ из состояния S'_{1i} в состояние S'_{1j} $(i, j = 0, 1, 2)$ первого блока являются условия перехода из состояния $S_\alpha \in S'_{1i}$ в состояние $S_\beta \in S'_{1j}$, определяемые исходным графом переходов: $\varphi_{0\to 0} = 0$, $\varphi_{0\to 1} = 1 \vee 5$, $\varphi_{0\to 2} = 2 \vee 3 \vee 4$; $\varphi_{1\to 1} = 3 \vee 5$, $\varphi_{1\to 0} = 2 \vee 5$; $\varphi_{1\to 2} = 4 \vee 7$; $\varphi_{2\to 2} = 1 \vee 3 \vee 5$, $\varphi_{2\to 0} = 6$, $\varphi_{2\to 1} = 0 \vee 5$.

Граф сцепления, соответствующий второму блоку, изображен на рис. 6.23,б. Он не содержит запрещенных фигур; следовательно, исходный автомат реализуем в виде двух параллельно работающих функционально несвязных подавтоматов. Раскраска второго графа сцепления разбивает множество состояний исходного автомата на следующие четыре множества:

$S_{20} = \{S_1, S_{10}, S_{11}\}$, $S_{21} = \{S_5, S_6, S_7\}$, $S_{22} = \{S_2, S_3, S_4\}$, $S_{23} = \{S_8, S_9, S_{12}\}$

Функции перехода для состояний второго подавтомата имеют следующий

$\varphi_{0\to0} = 2$, $\varphi_{0\to2} = 0$, $\varphi_{0\to3} = 3$, $\varphi_{1\to1} = 1 \vee 7$, $\varphi_{1\to0} = 3$, $\varphi_{1\to2} = 4$, $\varphi_{1\to3} = 5$, $\varphi_{2\to2} = 4 \vee 6$, $\varphi_{2\to0} = 2 \vee 5$, $\varphi_{2\to1} = 0 \vee 7$, $\varphi_{3\to3} = 6$, $\varphi_{3\to0} = 2 \vee 5$, $\varphi_{3\to1} = 7$, $\varphi_{3\to2} = 4$.

Разложение заданного графа переходов на функционально несвязные сомножители приведено на рис. 6 22,б.

Построим абстрактную декомпозицию автомата, используя семантику разложения графа переходов в произведение сомножителей. Пусть задан граф переходов G (рис. 6.24, а), который необходимо разложить в произведение $G = G_1 \times G_3$, где G_1 и G_2 — графы переходов, мощность носителя каждого из которых не превышает 3.

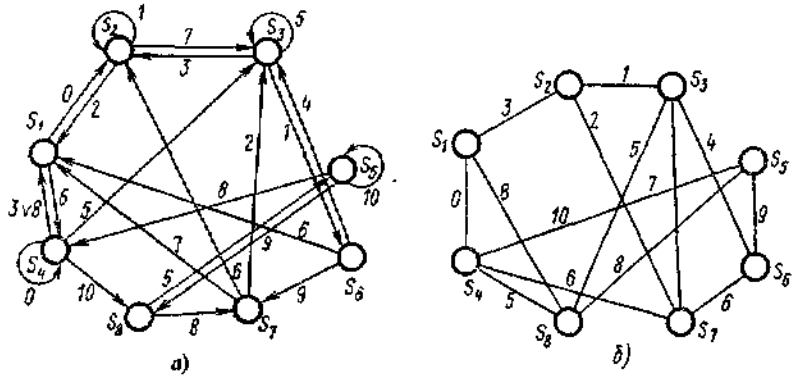


Рис. 6.24

Построим граф сцепления $G_{сц}$ и произведем его раскраску по первой компоненте (рис. 6.24,б). Для этого выделим все пустые подграфы $\{S_2, S_4, S_6\}$, $\{S_3, S_4\}$, $\{S_1, S_6\}$, $\{S_1, S_3, S_5\}$, $\{S_1, S_2, S_7\}$, $\{S_2, S_5\}$, $\{S_2, S_6, S_8\}$, $\{S_7, S_8\}$ графа сцепления.

Покрываем столбцы строками таблицы (табл. 6.12), в которой каждой строке взаимно однозначно соответствует пустой подграф, столбцу — вершина, а на пересечении i-й строки и j-го столбца стоит 1, если j-я вершина содержится в носителе i-го пустого подграфа, и 0 в противном случае.

Таблица 6. 12

Носители пустых подграфов	Вершины							
	S_1	S_2	S_3	S_6	S_5	S_4	S_8	S_7
$\{S_1, S_5, S_7\}$	1	0	0	0	1	0	0	1
$\{S_1, S_3, S_5\}$	1	0	1	0	1	0	0	0
$\{S_1, S_6\}$	1	0	0	1	0	0	0	0
$\{S_2, S_4, S_6\}$	0	1	0	1	0	1	0	0
$\{S_2, S_6, S_8\}$	0	1	0	1	0	0	1	0
$\{S_2, S_5\}$	0	1	0	0	1	0	0	0
$\{S_3, S_4\}$	0	0	1	0	0	1	0	0
$\{S_7, S_8\}$	0	0	0	0	0	0	1	1

Число допустимых состояний графа G_1 не превышает трех, следовательно, мощность покрытия этой таблицы также не должна быть больше трех. Запрещенных фигур по первой компоненте (квазиполных графов квазиплотности 4) граф сцепления не содержит; следовательно, такие покрытия существуют, поскольку хроматическое число графа равно его квазиплотности. Эти покрытия имеют следующий вид $\pi_1 = \{\{S_2, S_4, S_6\}, \{S_1, S_3, S_5\}, \{S_7, S_8\}\}$; $\pi_2 = \{\{S_1, S_5, S_7\}, \{S_2, S_6, S_8\}, \{S_3, S_4\}\}$.

Для определенности выберем первое покрытие. Ему соответствуют раскраска графа сцепления, представленная на рис. 6 25, *а.*

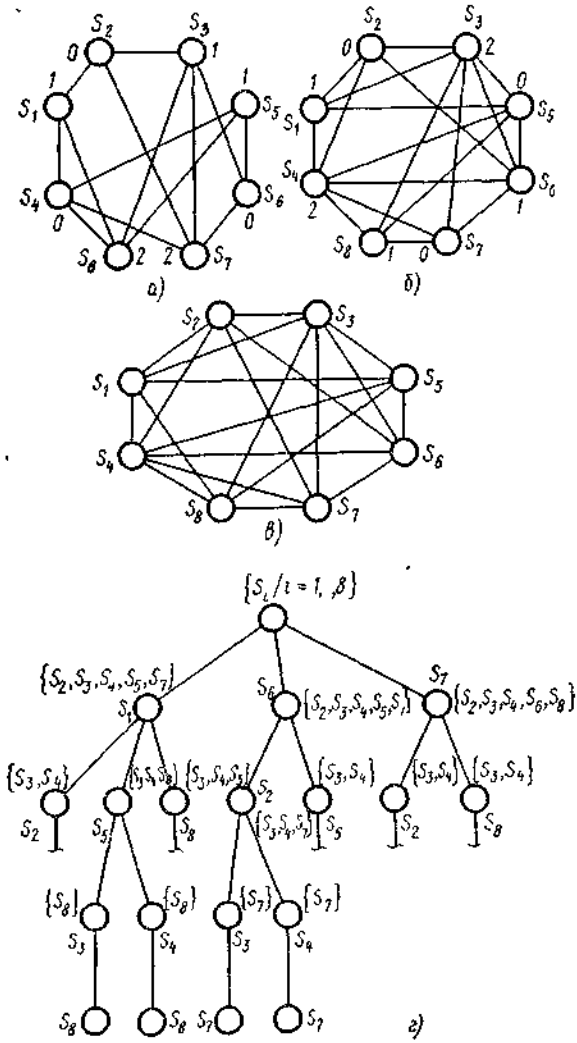


Рис. 6. 25

Согласно этой раскраске, функции переходов φ_{ijk}, где i - номер компоненты разложения, j - состояние (краска), из которого осуществляется переход, k - состояние (краска), в которое осуществляется переход первого сомножителя графа переходов G_1

, имеют следующий вид:

$\varphi_{100} = 0 \vee 1, \varphi_{101} = 2 \vee 3 \vee 4 \vee 5 \vee 6,$

$\varphi_{102} = 9 \vee 10, \varphi_{110} = 1 \vee 3 \vee 7 \vee 8 \vee 0, \varphi_{111} =$

$= 5 \vee 10, \varphi_{112} = 4 \vee 9, \varphi_{120} = 6, \varphi_{121} = 2 \vee 5 \vee 7, \varphi_{122} = 8.$

Для того чтобы произведение графов удовлетворяло условию автоматности, перед раскраской графа сцепления по второй компоненте необходимо соединить ребрами вершины, соцветные по первой компоненте (рис 6.25,б). Чтобы установить, содержит ли полученный граф (рис. 6.25, в) запрещенные фигуры, выделим полные подграфы плотности 4. Опять воспользуемся приведенным выше алгоритмом, который модифицируем так, чтобы в ярусе располагались несмежные вершины, причем взвешивались вершинами, смежными с ними. Пути, длина которых меньше четырех, обрываем (рис. 6.25, *г).* Кроме выделенных полных подграфов плотности 4, носители которых имеют соответственно вид

$\{S_1, S_3, S_5, S_8\}$, $\{S_1, S_4, S_5, S_8\}$, $\{S_2, S_3, S_6, S_7\}$, $\{S_2, S_4, S_6, S_7\}$, граф сцепления при раскраске по второй компоненте содержит еще шесть запрещенных фигур - квазиполных графов квазиплотности 4 (рис. 6.26).

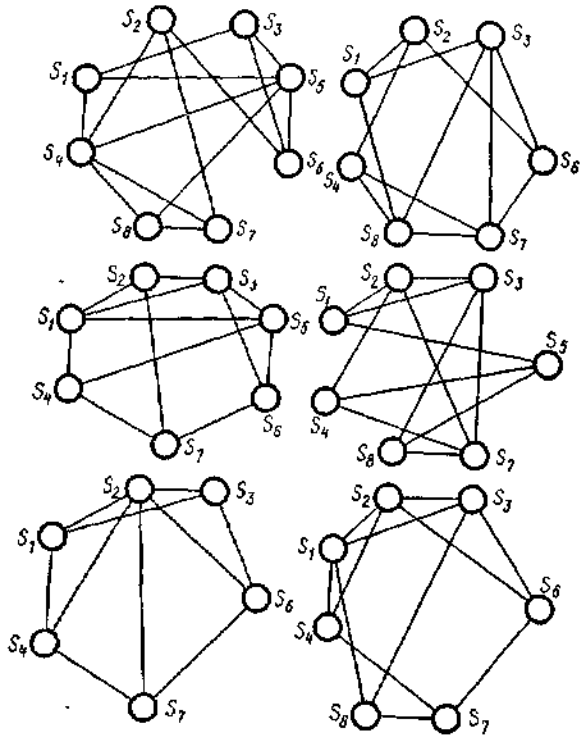


Рис. 6. 26

Построим семантическую таблицу (табл. 6.13). Каждой ее строке взаимно однозначно соответствует ребро запрещенной фигуры, столбцу — запрещенная фигура и

$$(i, j) = \begin{cases} 1, \text{ если } i\text{-е ребро содержится в } j\text{-й фигуре;} \\ 0 \text{ в противном случае.} \end{cases}$$

Таблица 6.13

Ребра	Квазиполные графы квазиплотности 4									
$\{S_1, S_4\}$	1	0	1	0	1	0	1	1	0	0
$\{S_1, S_5\}$	1	0	1	0	1	1	0	0	0	1
$\{S_1, S_8\}$	1	0	0	1	0	0	0	1	0	1
$\{S_4, S_8\}$	1	0	1	1	0	0	0	0	0	0
$\{S_4, S_5\}$	1	0	1	0	1	1	0	0	0	0
$\{S_5, S_8\}$	1	0	1	0	0	1	0	0	0	1
$\{S_2, S_7\}$	0	1	1	0	1	1	1	0	1	0
$\{S_2, S_3\}$	0	1	0	0	1	1	1	1	0	0
$\{S_2, S_6\}$	0	1	1	1	0	0	1	1	1	0
$\{S_3, S_7\}$	0	1	0	1	0	1	0	0	0	0
$\{S_6, S_7\}$	0	1	0	1	1	0	1	1	1	0
$\{S_3, S_6\}$	0	1	1	1	1	0	1	1	0	0
$\{S_2, S_4\}$	0	0	1	1	0	1	1	1	1	0
$\{S_3, S_8\}$	0	0	0	1	0	1	0	1	0	1
$\{S_3, S_5\}$	0	0	1	0	1	0	0	0	0	1
$\{S_5, S_6\}$	0	0	1	0	1	0	0	0	0	0
$\{S_4, S_7\}$	0	0	1	1	1	1	1	1	1	0
$\{S_1, S_3\}$	0	0	1	1	1	1	1	1	0	1
$\{S_7, S_8\}$	0	0	1	1	0	1	0	1	0	0
$\{S_1, S_2\}$	0	0	0	1	1	1	1	1	0	0
$\{S_4, S_6\}$	0	0	0	0	0	0	0	0	1	0

Третья и седьмая строки образуют минимальное покрытие. Следовательно, при удалении из графа сцепления соответствующих им ребер $\{S_1, S_8\}$ и $\{S_2, S_7\}$ возможна раскраска графа тремя цветами (см. рис. 6.25,б). В результате получаем двухкомпонентную раскраску графа сцепления (см. рис. 6.25, а, б):

$S_1 - (1, 1)$, $S_2 - (0, 0)$, $S_3 - (1, 2)$,

$S_4 - (0, 2)$, $S_5 - (1, 0)$, $S_6 - (0, 1)$, $S_7 - (2, 0)$, $S_8 - (2, 1)$.

Удаление ребер $\{S_1, S_8\}$ и $\{S_2, S_7\}$ означает, что состояния S_1 и S_8 исходного графа переходов не должны быть сцеплены входным вектором 8 (см. рис. 6.25, *а),* а состояния S_2 и S_7 — вектором 2. Для расцепления этих векторов введем в соответствующих четырех переходах дополнительные разряды, по которым эти векторы будут отличаться. Это возможно осуществить либо используя связи между компонентами (в данном случае состояние первой компоненты), либо организуя специальный развязывающий блок памяти.

В первой компоненте спектра состояниям S_1 и S_8 сопоставлены соответственно краски *1* и *2.* Для расцепления этих состояний вектор **8**, взвешивающий переход из состояния S_1, расширяем до вектора $\mathbf{8}' = \mathbf{8} \cdot \tilde{S}_{11}$; вектор **8**, взвешивающий переход из состояния S_8, расширяем до вектора $\mathbf{8}'' = \mathbf{8} \cdot \tilde{S}_{12}$, где $\tilde{S}_{11}$, $\tilde{S}_{12}$- значения разрядов, в

которых отличаются коды красок 1 и *2* первой компоненты спектра (рис. 6.27, *а*).

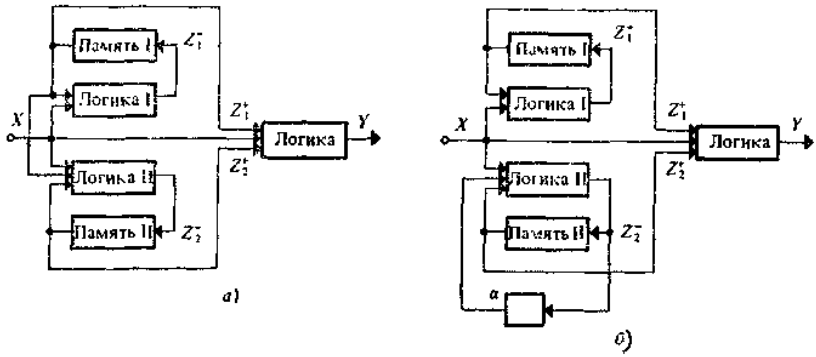


Рис. 6.27

Для расцепления состояний S_2 и S_7 соответственно имеем

$\mathbf{2'} = \mathbf{2} \cdot \tilde{S}_{10}$; $\mathbf{2''} = \mathbf{2} \cdot \tilde{S}_{12}$.

Специальный блок памяти в данном случае представляет один элемент памяти α (рис. 6.27,б), в котором одно из состояний (например, нулевое) сопоставляется состояниям S_1 и S_2, другое - состояниям S_8 и S_7. В этом случае $\mathbf{8'} = 8\tilde{\alpha}$, $\mathbf{8''} = 8\alpha$; $\mathbf{2'} = 2\tilde{\alpha}$, $\mathbf{2''} = 2\alpha$.

Значение специального развязывающего блока памяти фиксируется при переходе в состояние, однозначность выхода из которого определяется состоянием этого блока.

В результате получаем следующие функции возбуждения графа переходов, определяющего второй сомножитель разложения:

$\varphi_{200} = 1 \vee 6 \vee 10$, $\varphi_{201} = 2' \vee 7 \vee 9$, $\varphi_{202} = 2'' \vee 3 \vee 8$, $\varphi_{210} = 0 \vee 5 \vee \vee 8'' \vee 9$, $\varphi_{211} = 6$, $\varphi_{212} = 3 \vee 4 \vee 8'$, $\varphi_{220} = 7$, $\varphi_{221} = 1 \vee 4 \vee 6$, $\varphi_{222} = 0 \vee 5$.

Полученная абстрактная параллельная декомпозиция заданного автомата со связными сомножителями приведена на рис. 6.28.

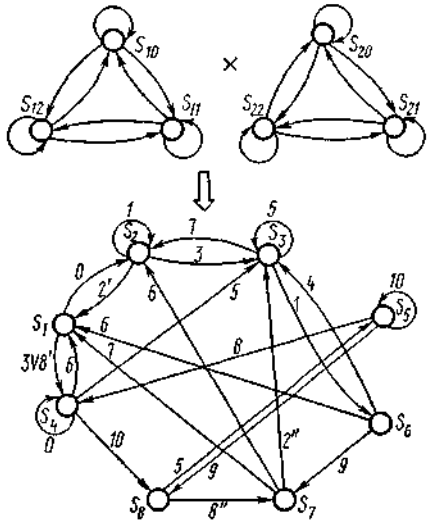


Рис. 6.28

Для ответа на вопрос, возможно ли разложение рассматриваемого автомата в частичное декартово произведение несвязных между собой сомножителей, рассмотрим раскраску графа сцепления с учетом характера переходов сцепленных состояний. Такой учет позволяет не принимать во внимание связь в графе сцепления, если из соответствующих сцепленных состояний переход осуществляется в соцветные вершины. Следовательно, перед расширением входного вектора для расцепления внутренних состояний необходимо проверить возможность устранения этой связи одинаковой раскраской тех состояний, в которые сцепленные состояния переходят.
Построим таблицу переходов заданного автомата (табл. 6.14), каждой строке которой взаимно однозначно соответствует значение входного вектора, столбцу - внутреннее состояние, а в клетке (*i, j)* таблицы находится идентификатор внутреннего состояния, в которое автомат переходит из *j*-го состояния под воздействием *i*-го входного вектора.

Таблица 6.14

x_1	S_1	S_2	S_3	S_4	S_5	S_6	S_7	S_8
0	S_2			S_4				
1		S_2	S_6					
2		S_1					S_3	
3	S_4	S_3		S_3				
4						S_3		
5			S_3					S_5
6				S_1		S_1	S_2	
7			S_2			S_7	S_1	
8	S_4				S_4		S_7	
9					S_8			
10				S_8	S_5			

Несвязное разложение имеет место, если, согласно покрытию семантической таблицы, связность пар $\{S_1, S_8\}$ и $\{S_2, S_7\}_v$ не учитываем из-за возможной соцветности пар вершин $\{S_4, S_7\}$ и $\{S_1, S_3\}$. Состояния S_4 и S_7 сцеплены, и, для того чтобы они были соцветны, необходима соцветность вершин S_1, S_2, в которые эти состояния переходят. Состояния S_1, S_2 сцеплены, и, для того чтобы они были соцветны, необходима соцветность вершин S_3, S_4. Состояния S_3, S_4 не сцеплены; следовательно, могут быть соцветны. Раскрашиваем вершины S_3, S_4 в один цвет. Тогда, согласно свойству транзитивности отношения соцветности, получаем, что каждое из подмножеств $K_0 = \{S_1, S_2, S_8\}$, $K_1 = \{S_3, S_4, S_7\}$, $K_2 = \{S_5, S_6\}$ состоит из соцветных вершин.

При найденной раскраске графа сцепления второй компоненты разложения состояния S_2, S_7 являются несоцветными. Следовательно, и второе противоречие, обусловливающее недетерминированность графа

переходов второго сомножителя, также устранено. Окончательно получаем следующие функции возбуждения второй компоненты разложения:

$$\varphi_{200} = 0 \vee 1 \vee 2,\ \varphi_{201} = 3 \vee 8,\ \varphi_{202} = 5,\ \varphi_{210} = 6 \vee 7 \vee 10,\ \varphi_{211} = 0 \vee 2 \vee 5,\ \varphi_{212} = 1,\ \varphi_{220} = 6 \vee 9,\ \varphi_{221} = 4 \vee 8,\ \varphi_{222} = 10.$$

Таким образом, имеем несвязное разложение заданного автомата, определяемое полученными системами и двухкомпонентной раскраской вида:

$$S_1 - (1, 0),\ S_4 - (0, 1),\ S_7 - (2, 1),\ S_2 - (0, 0),\ S_5 - (1, 2),\ S_8 - (2, 0),\ S_3 - (1, 1),\ S_6 - (0, 2).$$

Вобщем случае для построения параллельной абстрактной декомпозиции абсолютно минимальной связности необходимо оценить каждую раскраску по первой компоненте раскрасками по второй и выбрать многокомпонентную раскраску, удовлетворяющую заданным количествам вершин графов сомножителей и их условиям связности. Разложение графа переходов на n компонент аналогично.

Рассмотрим предельную параллельную абстрактную декомпозицию автоматов, когда подавтоматом является элемент памяти. Семантика этой декомпозиции будет семантикой функциональной связности элементов памяти. С учетом структуры квазиполных графов и двузначности булевой логики рефлексивная семантика функциональной связности элементов памяти автомата определяется следующим утверждением.

Теорема 6.5. *Графы сцепления, не содержащие циклов нечетной длины, определяют кодирование, при котором элементы памяти функционально несвязны.*

Этот критерий позволяет последовательно определять значения разрядов в кодах внутренних состояний, аналогично тому, как это делалось при поиске параллельной абстрактной декомпозиции.

6.5. Характеризация и методы оптимального размещения данных в памяти ЭВМ

Для информационных систем характерны не только большие объемы, но и сложность хранимых данных. **Сложность данных проявляется в том, что они находятся в различных взаимосвязях**. Таким образом, **сложные данные представляются в виде набора некоторых элементарных объектов и совокупности отношений, связывающих объекты данных. Иначе говоря, сложные информационные системы формализуются такими понятиями, как *граф* и *мограф*.**

Это подтверждается богатым практическим опытом построения информационных систем.

Часто возникает необходимость хранить в «чистом виде» графовые структуры, например, при использовании в базах данных моделей данных, основанных на графах. Объекты данных хранятся отдельно от своих связей, которые представляются графом. Известно несколько способов задания графов: с помощью матриц инциденций и смежности, перечислением окрестностей вершин. Можно показать, что последний способ наиболее экономичен при больших размерностях графов, что характерно для практики. Но задание графа перечислением окрестностей равносильно заданию мографа, носителем которого является носитель графа, а словами — окрестности его вершин. Возможны варианты в задании графа, но в любом случае абстракцией представления может быть мограф. Например, в базах данных, основанных на сетевой модели данных (фрагмент сетевой схемы данных приведен на рис. 6.*29,а),* абстракцией данных является ориентированный граф.

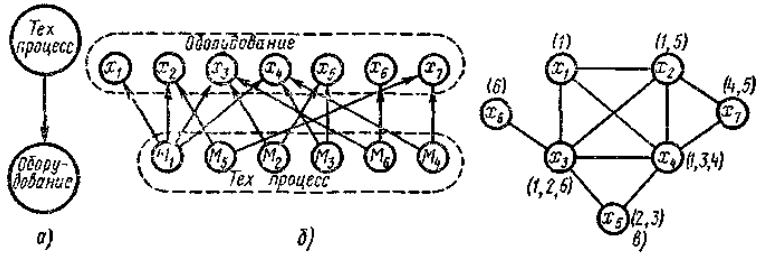


Рис. 6.29

При обработке запросов поиск информации осуществляется в направлении, которое указывается дугами (от данных о процессах к данным об оборудовании), поэтому хранить необходимо только положительные окрестности (окрестности нижнего яруса графа, изображенного на рис. 6.29.б). Эта информация задается мографом, приведенным на рис. 5.29,в.

Как было показало выше, мограф является абстракцией информационно-поисковой системы с фиксированным множеством запросов. Другим примером, когда мотраф представляет информационную систему, является организация файлов с несколькими ключами доступа. Файл представляет собой последо- - вательность записей, состоящих из идентичных полей (записи могут быть неодинаковой длины). Ключом называется поле или множество полей, значения которых идентифицируют записи. Доступ к файлу осуществляется с помощью указания значения ключа. Для ускорения

обработки запросов организуются индексы - таблицы, в которых для каждою значения ключа указываются адреса записей, имеющих это значение. Если индекс хранит адреса всех записей, имеющих данное значение, то такая организация называется инвертированными списками. Информация, хранящаяся в инвертированных списках, представляется мографом, носитель которого образуется множеством адресов записей, а слова — множествами адресов записей, имеющих идентичные значения ключа.

Рассмотрим файл библиотечной информационной системы (рис. 6.30).

адрес	Фамилия И.О.	Название	Выходные данные			Ключевые слова	...
x_1	Самарский А.А.	Теория разностных схем	М	Наука	1977	Математические методы	
x_2	Марчук Г.И.	Методы вычислительной математики	М	Наука	1977	Математические методы, математическое моделирование	
x_3	Макаров И.М. /ред.	Основы автоматизации управления производством	М	Высшая школа	1983	Математические методы, производственные процессы, автоматизация	
x_4	Свами М., Тхуласираман К.	Графы, сети, алгоритмы	М	Мир	1984	Математические методы, теоретико-графовые модели, алгоритмы на графах	
x_5	Брейер М.	Теория и методы автоматизации проектирования вычислительных систем	М	Мир	1977	Автоматизация, алгоритмы на графах	
x_6	Ржевский В.В.	Процессы открытых горных работ	М	Недра	1978	Производственные процессы	
x_7	Питерсон Дж.	Теория сетей Петри и моделирование систем	М	Мир	1984	Математическое моделирование, теоретико-графовые модели	

Рис. 6.30

Он имеет несколько ключей доступа: по фамилии автора, по названию издательства, по ключевому слову и др. Для ускорения доступа файл можно упорядочить по одному из ключей (обычно по фамилии), по другим же ключам записи будут совершенно не упорядочены. В рассматриваемом примере индекс по ключу доступа «ключевое слово» представляется уже ранее приведенным мографом (см. рис.

6.29, в), слову M_1 соответствует значение «математические методы», M_2— «автоматизация» и т. д.

Мограф может задавать не только инвертированные списки, но и другие списковые организации индексов: мультисписки, секционные списки. Наконец, просто двоичная матрица может рассматриваться как матрица инцидент ностей мографа.

Основными критериями при размещении данных в памяти ЭВМ являются минимизация объема памяти и времени доступа. Элементы носителя мографа однозначно соответствуют объектам данных, размещенным в памяти, следовательно, критерий минимизации объема памяти определяет функционал качества φ (Ψ_b) при семантическом эквивалентировании как минимум мощности носителя Ψ_b . Мограф Ψ_b определяет размещение, в котором все слова отыскиваются максимально быстро. Если мограф Ψ_a не допускает такого размещения, то поиск некоторого слова эквивалентен поиску нескольких слов, что равносильно тому, что данное слово расщеплено на несколько слов. Таким образом, критерий минимизации времени доступа определяет функционал качества φ (Ψ_b) как минимум мощности сигнатуры Ψ_b .

В рассмотренных примерах организации данных, представляемых мографом, при условии минимального времени доступа важно минимизировать объем памяти. Возможен и обратный критерий: минимизация времени доступа при неизменном объеме памяти. Он важен, например, в задаче такого упорядочения записей файла (рис. 6.31), при котором быстрее бы осуществлялся поиск по ключу.

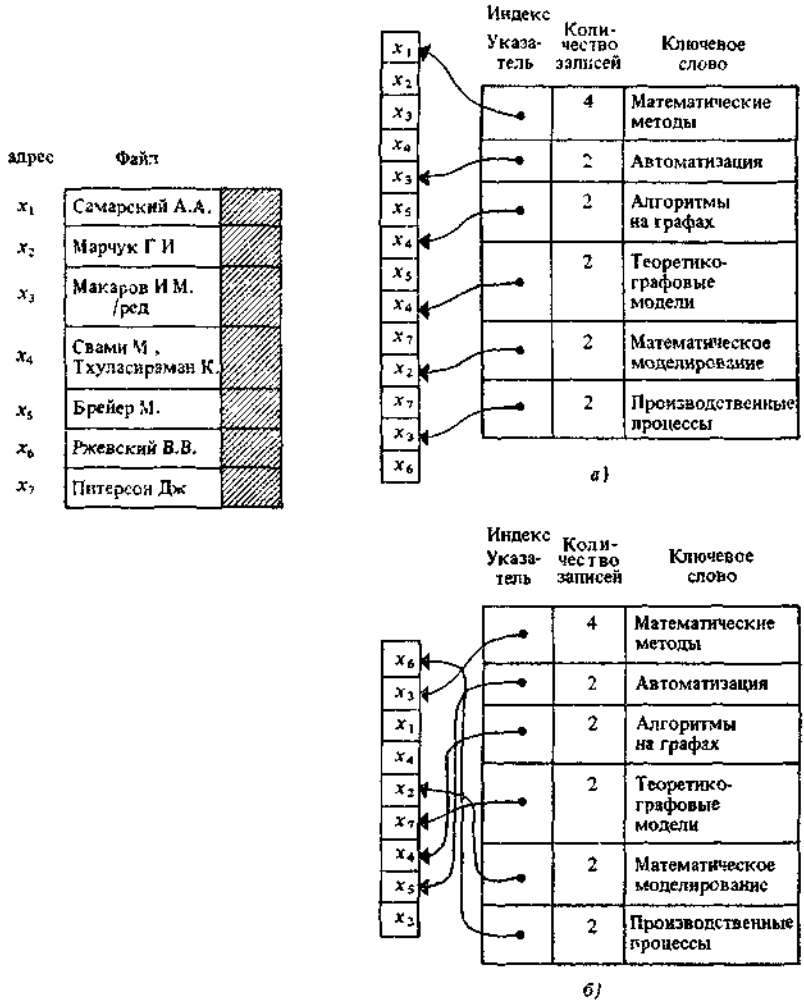


Рис. 6.31

Доступ к объектам данных осуществляется с помощью некоторой стандартной процедуры поиска, реализующей переход от одного элемента памяти к другому. Переход осуществляется однозначно, поэтому эту процедуру поиска можно формализовать функцией осмотра памяти S — частичной функцией на множестве элементов носителя мографа $S\colon X \to X$; $S(x)$ — элемент, к которому осуществляется переход после

элемента *x*. Рассмотрим такое размещение данных и такую функцию осмотра памяти *S* (модель Ψ_b), при которых для каждого слова *M* модели Ψ_b имеется такой элемент $x_0 \in M$, что

$$M = \{x_0,\ S(x_0),\ S^2(x_0),\ \ldots,\ S^{|M|-1}(x_0)\},$$

т. е. каждое слово отыскивается с помощью функции S на основании только информации о мощности слова и начальном элементе x_0. Подобные мографы называются допустимыми. Очевидно, что на практике большинство мографов не являются допустимыми. Для их реализации в памяти ЭВМ необходимо расщепление элементов носителя или расщепление слов. При этом увеличивается либо объем памяти, либо время доступа. Допустимые же мографы представляются в памяти ЭВМ безызбыточным образом и требуют минимального времени доступа.

Таким образом, **задача оптимального размещения данных заключается в преобразовании мографа в допустимый и построении функции осмотра памяти**. При этом функционалом качества является минимальное расширение носителя без изменения сигнатуры или минимальное расширение сигнатуры без изменения носителя. Эта задача имеет графовую интерпретацию. Рассмотрим функцию осмотра памяти $S : X \to X$ допустимого мографа Ψ_b как отношение смежности $S \subseteq X \times X$ вершин в ориентированном графе $G_f = \langle X, S \rangle$, построенном на множестве вершин *X*. Граф G_f является функциональным (*f*-графом), т. е. из каждой вершины его исходит не более одной дуги.

Теорема 6.6. *Связный f-граф является либо ациклическим, либо содержит точно один цикл.*

Из этого утверждения следует, чго практически важными являются следующие классы *f*-графов и допустимых мографов, представляющихся соответствующими *f*-графами: линейные (пути); циклические (контуры); ациклические (ориентированные деревья). Обычно на практике функция осмотра памяти реализуется либо с помощью указателей, либо переходом к соседнему элементу памяти. Для представления в памяти линейных *f* -графов можно использовать второй вариант реализации функции осмотра памяти. Представление циклических *f* -графов также основано на переходе к соседнему элементу памяти, но с одним исключением: $S(x_n) = x_1$, где x_1, x_n — соответственно первый и последний в размещении элементы. Для представления ациклических *f* -графов необходимы указатели. Однако представление ациклического *f* -графа парами *(x, S(x))* для каждой вершины *x* избыточно (рис. 6.32, *а, б).*

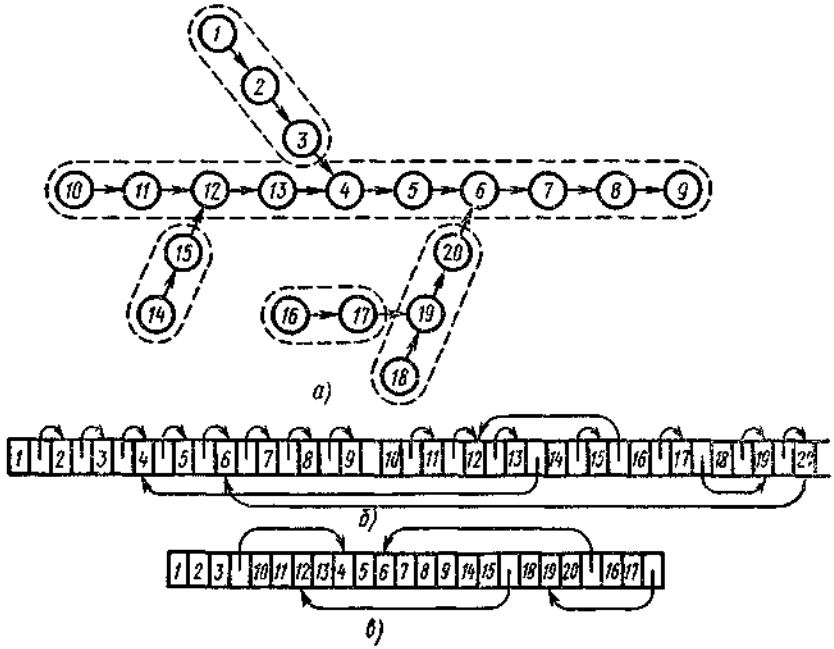


Рис. 6.32

Для безызбыточного представления разобьем *f* -граф на линейные фрагменты, внутри которых в качестве функции осмотра будем использовать переход к соседнему элементу памяти, а связь фрагментов задавать указателями (рис. 6.32, в).
Для решения характеризационных проблем представления графов *f*-/графами различных классов введем отношения подчинения. Граф Ψ_1 подчинен графу Ψ_2 в отношении подчинения $P^1_{\text{п}}$, если он получен из Ψ_2 последовательностью ограничений, сужений, расширений и сверток. **Под ограничением графа понимаем удаление некоторых его слов, под сужением — удаление некоторых элементов носителя, под расширением — введение нового слова, являющегося пересечением имеющихся, под сверткой - склеивание всех вершин некоторого слова с объединением их весов.** Введенные операции иллюстрирует рис. 6.33, *а - д*. Отношение подчинения $P^1_{\text{п}}$ используется для характеризации линейных и ациклических графов.

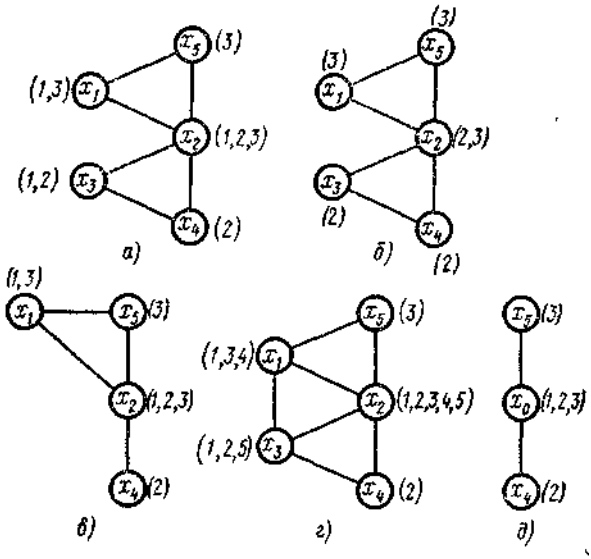


Рис. 6.33

Отношение подчинения $P^1_{\text{п}}$, используется для характеризации линейных и ациклических мографов.
Отношение подчинения $P^2_{\text{п}}$ кроме операций, используемых в $P^1_{\text{п}}$, включает еще одну операцию — **замыкание слова, т. е. замену его на дополнение в множестве элементов носителя мографа** (рис. 6.34), а на **операцию расширения** $P^2_{\text{п}}$ накладывает следующее ограничение: можно включать только слово M, являющееся пересечением слов M_i, M_j таких, что их объединение не образует весь носитель мографа (возможное расширение). Это отношение подчинения используется для характеризации циклических мографов.

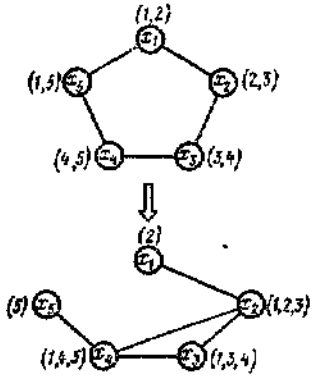


Рис. 6.34

Принцип локальности выполняется для свойств линейности и ацикличности при отношении $P^1_{\text{п}}$ и для свойства цикличности при отношении $P^2_{\text{п}}$.

Теорема 6.7 (теорема В. Л. Торхова). *Запрещенными фигурами для классов допустимости мографов являются следующие мографы:*
1) *для свойства линейности и отношения подчинения* $P^1_{\text{п}}$ — S_3 *и* K_4*;*
2) *для свойства цикличности и отношения подчинения* $P^2_{\text{п}}$ — K_4*;*
3) *для свойства ацикличности и отношения подчинения* $P^1_{\text{п}}$ — S_3, K^*_4, K_5.

Мографы S_3, K_4, K^*_4, K_5 приведены на рис. 6.35, *а —г.*

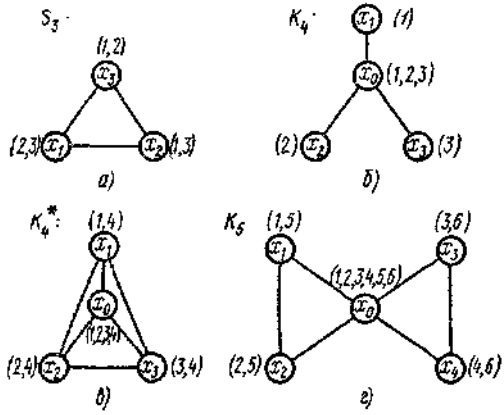


Рис. 6.35

Способы преобразования этих запрещенных фигур в разрешенные заключаются либо в расщеплении элемента носителя x и соответствующем разбиении множества слов $E(x)$, в которые он входил, на два множества $E_1(x)$ и $E_2(x)$ (обозначим этот способ как $x(E_1, E_2)$), либо в расщеплении слова M_i на два M_i' и M_i'' с соответствующим распределением элементов по этим словам (обозначим как $i(M_i', M_i'')$). Способы преобразования запрещенных фигур в принятых обозначениях объединены в табл. 6.15.

Таблица 6.15

Запрещенная фигура	Способ преобразования	
	расширение носителя	расширение сигнатуры
S_3	$x_1(2, 3)$ $x_2(1, 3)$ $x_3(1, 2)$	$1(x_2, x_3)$ $2(x_1, x_3)$ $3(x_1, x_2)$
K_4	$x_0(3, (1, 2))$ $x_0(2, (1, 3))$ $x_0(1, (2, 4))$	$1(x_0, x_1)$ $2(x_0, x_2)$ $3(x_0, x_3)$
K_4^*	$x_1(1, 4)$ $x_2(2, 4)$ $x_3(3, 4)$ $x_0(1, (2, 3, 4))$ $x_0(2, (1, 3, 4))$ $x_0(3, (1, 2, 4))$	$1(x_1, x_0)$ $2(x_2, x_0)$ $3(x_3, x_0)$ $4(x_1, (x_2, x_3, x_0))$ $4(x_2, (x_1, x_3, x_0))$ $4(x_3, (x_1, x_2, x_0))$
K_5	$x_1(1, 5)$ $x_2(2, 5)$ $x_3(3, 6)$ $x_4(4, 6)$ $x_0(1, (2, 3, 4, 5, 6))$ $x_0(2, (1, 3, 4, 5, 6))$ $x_0(3, (1, 2, 4, 5, 6))$ $x_0(4, (1, 2, 3, 5, 6))$	$1(x_1, x_0)$ $2(x_2, x_0)$ $3(x_3, x_0)$ $4(x_4, x_0)$ $5(x_1, (x_0, x_2))$ $5(x_2, (x_0, x_1))$ $6(x_3, (x_0, x_4))$ $6(x_4, (x_0, x_3))$
	$x_0((1, 2, 5), (3, 4, 6))$	

Можно показать, что эти способы базисные, т. е. любой другой способ преобразования является суперпозицией этих способов. Для модели в целом данные способы преобразования запрещенных фигур являются неоднозначными. Семантическое эквивалентирование мографа Ψ_a, в мограф Ψ_b с заданными свойствами допустимости осуществляется с помощью обычной **процедуры: строим семантическую таблицу; находим покрытия; оцениваем эти покрытия с помощью построения специальных графов и их раскраски.** В результате получаем мограф, обладающий заданным свойством допустимости. По нему строим соответствующий *f*-граф. Рассмотрим подробнее алгоритм построения линейного *f*-графа по линейному мографу (будем считать, что мограф связный).

Определим отношение упорядочения P_E на носителе графа такое, что $(x_i, x_j) \in P_E$, если $E(x_i) \subset E(x_j)$ (одноэлементные слова при этом не учитываем). Найдем множество минимальных элементов X_1 отношения упорядочения P_E. Оставим в нем по одному элементу из тех, которые входят в одинаковые слова, а затем только такие элементы x_i, удаление которых вместе с $E(x_i)$ не делает граф несвязным. (Можно показать, что таких элементов всегда не более двух.) Пусть останутся элементы x_i и x_n. Зафиксируем один из них, например x_n, как конечную вершину f-графа. Удалим другой элемент x_i из графа и введем его в f-граф. Если в f-графе уже есть вершины, то соединим предыдущую введенную вершину x_j дугой с x_i. Продолжаем этот процесс до тех пор, пока в графе не останется одна вершина x_n. Поступаем с ней аналогично.

Рассмотрим процесс семантического эквивалентирования графа Ψ_a (см. рис. 5.29, в) в линейный f-граф. Запрещенные фигуры для свойства линейности, присутствующие в Ψ_a приведены на рис. 6.36.

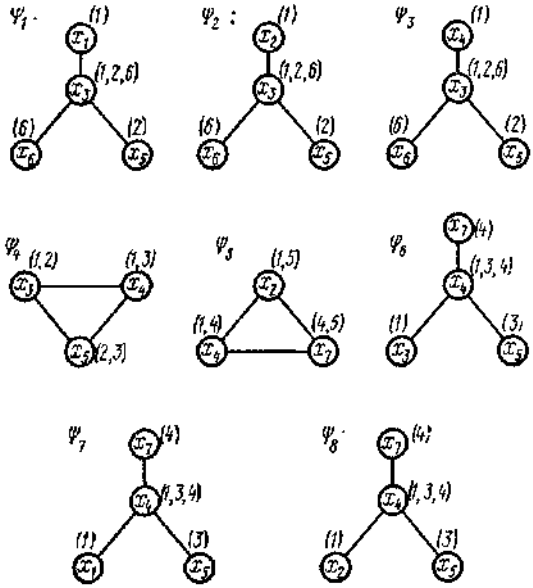


Рис. 6.36

Если критерием является минимизация функционала $\varphi(\Psi_b)$, равного мощности носителя Ψ_b, при условии неувеличения мощности сигнатуры Ψ_a, то можно применять только преобразование, основанное

на расщеплении элементов носителя. Семантическая таблица имеет вид (табл. 6.16):

Табтца 6.16

ψ_1	ψ_2	ψ_3	ψ_4	ψ_5	ψ_6	ψ_7	ψ_8	
1	1	1	1					$x_3(1, (2, 6))$
1	1	1	1					$x_3(2, (1, 6))$
1	1	1						$x_3(6, (1, 2))$
			1					$x_3(1, 2)$
			1					$x_5(2, 3)$
			1					$x_4(1, 3)$
				1				$x_2(1, 5)$
				1				$x_4(1, 4)$
				1				$x_7(4, 5)$
			1	1	1	1	1	$x_4(1, (3, 4))$
			1		1	1	1	$x_4(3, (1, 4))$
				1	1	1	1	$x_4(4, (1, 3))$

Рассмотрим покрытие $\pi = \{x_3(2,(1,6)), x_4(1,(3,4))\}$. Выполнение этих преобразований в Ψ_a приведет к линейному мографу Ψ_b ((рис. 6.37, *а*).

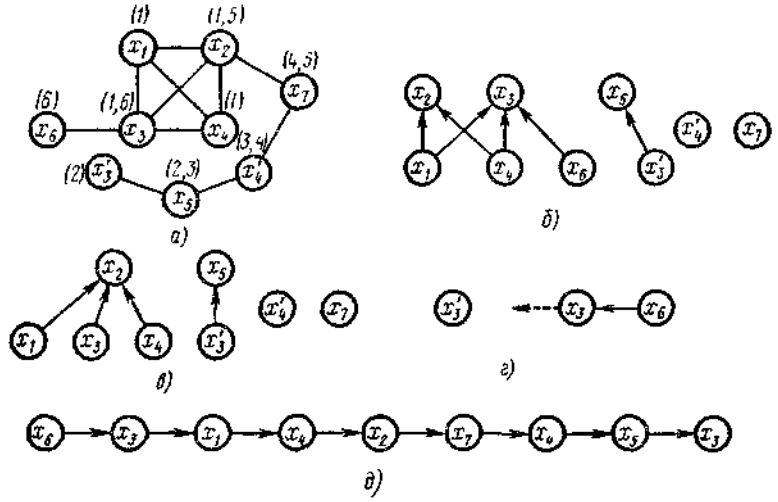


Рис. 6.37

Построим, следуя предложенному алгоритму, линейный *f*-граф, представляющий Ψ_b . Определим отношение упорядочения P_E (рис. 6.37, *б).* Минимальными его элементами являются x_1, x_4, x_6, x'_3. Удаляя из рассмотрения x_1 или x_4, со словом *M* приходим к несвязному мографу. Элемент x'_3 фиксируем как конечную вершину *f*-графа. Вводим в *f*-граф вершину x_6, удаляя этот элемент из Ψ_b . Затем снова строим отношение упорядочения P_E (рис. 6.37, в). Минимальными элементами являются x_1, x_3, x_4, x'_3. Первые три входят в одно множество слов; выбираем элемент x_3, остальные из рассмотрения удаляем. Вводим x_3 в *f*-граф (рис. 6.37, г). Продолжая эту процедуру, строим *f*-граф (рис. 6.37, *д).*

Мограф Ψ_b представляет систему инвертированных списков для файла библиотечной информационной системы (см. рис. 6.30). При использовании обычной списковой организации с линейной функцией осмотра инвертированные списки (см. рис. 6.31, *а)* занимают 14 элементов памяти; при использовании оптимального размещения, представляющего Ψ_b (см. рис. 6.31, б), - 9 элементов. Таким образом, **экономия памяти на инвертированных списках составила приблизительно 35%.**

Если критерием является минимизация функционала $\varphi(\Psi_b)$, равного мощности сигнатуры Ψ_b, при условии неувеличения мощности носителя Ψ_a, то можно применять только преобразования запрещенных

фигур, расщепляющие слова. Семантическая таблица имеет вид (табл 6.17):

Таблица 6.17

ψ_1	ψ_2	ψ_3	ψ_4	ψ_5	ψ_6	ψ_7	ψ_8	
1								$1(x_1, x_3)$
1	1	1	1					$2(x_3, x_5)$
1	1	1						$6(x_3, x_6)$
	1							$1(x_2, x_3)$
		1	1		1			$1(x_3, x_4)$
			1		1	1	1	$3(x_4, x_5)$
				1			1	$1(x_2, x_4)$
					1	1	1	$4(x_4, x_7)$
				1				$5(x_2, x_7)$
						1		$1(x_1, x_4)$

Одним из покрытий, определяющих минимальное решение, является неминимальное (по числу преобразований) покрытие $\pi = \{1(x_1, x_3), 1(x_2, x_3), 1(x_3, x_4), 1(x_2, x_4), 1(x_1, x_4)\}$. Специальный граф для слова М (а все способы расщепляют только его) приведен на рис. 6.38, *а.*

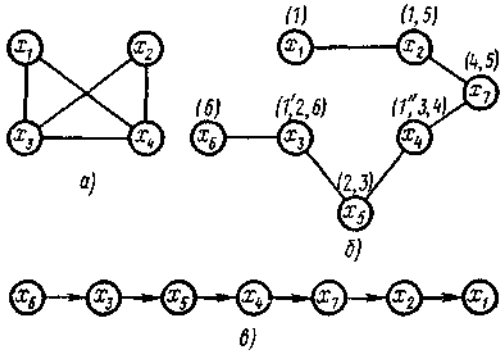


Рис. 6.38

Раскраска его в три цвета определяет минимальное расщепление M_1: $M_1' = \{x_1, x_2\}$, $M_1'' = \{x_3\}$, $M_1''' = \{x_4\}$; после этого мограф становится линейным (рис. 6.38, *б)* и представляется линейным *f*-графом (рис. 6.38, в).

Решение этой задачи позволяет в соответствии с построенным *f*-графом так упорядочить записи файла библиотечной информационной системы (рис. 6.39), что записи, имеющие идентичное значение ключевого слова, сгруппированы в наименьшее число цепочек. Если в первоначальном файле (рис. 6.30) таких цепочек было 10 (для ключевого слова «математические методы» — 1 цепочка, для ключевого слова «автоматизация» — 2 и т. д.), то в полученном размещении (рис. 5.39) таких цепочек 8 (для ключевого слова «математические методы» — 3, для остальных — по одной). Таким образом, быстродействие при обработке файла в результате оптимизации увеличилось на 20%. Для обоих критериев получено минимальное решение.

адрес	Фамилия И.О. . . . .
x_6	Ржевский В.В.
x_3	Макаров И.М. /ред.
x_5	Брейер М.
x_4	Свами М., Тхуласираман К.
x_7	Питерсон Дж.
x_2	Марчук Г.И.
x_1	Самарский А.А.

Рис. 6.39

Приложение

Список алгоритмов

Ниже приводится **список алгоритмов**, группированный по категориям. Более детальные сведения приводятся в списке структур данных и списке основных разделов теории алгоритмов[1]

Содержание

Комбинаторные алгоритмы

Общие комбинаторные алгоритмы

- Алгоритм Флойда для нахождения циклов (англ.) — находит цикл в итерациях
- Генераторы псевдослучайных чисел:
  - Алгоритм Блюм — Блюма — Шуба
  - Вихрь Мерсенна
  - Метод Фибоначчи с запаздываниями
  - Линейный конгруэнтный метод
  - Инверсный конгруэнтный метод

Генерация комбинаторных объектов

- Алгоритм Робинсона — Шенстеда — генерация перестановок из пар таблиц Юнга
- Алгоритм Нарайаны
- Алгоритм Джонсона — Троттера (*англ.*)

Алгоритмы на графах

- Алгоритм Беллмана — Форда — вычисляет кратчайший путь во взвешенном графе (где некоторые веса рёбер могут быть отрицательны)
- Алгоритм Борувки — находит минимальное остовное дерево в графе

- Алгоритм Брона — Кербоша — нахождение наибольших максимальных независимых по включению множеств вершин графа.
- Алгоритм Флойда — Уоршелла — вычисляет все кратчайшие пути во взвешенном графе
- Алгоритм Дейкстры — вычисляет кратчайший путь в графе с неотрицательными весами рёбер
- Алгоритм Левита — находит кратчайшее расстояние от одной из вершин графа до всех остальных
- Алгоритм Джонсона — вычисляет все кратчайшие пути во взвешенном графе
- Алгоритм Краскала — находит остовный лес минимального веса в графе
- Алгоритм, основанный на источнике (*англ.*) — алгоритм для рисования графа
- Алгоритм Прима — находит остовное дерево минимального веса в связном графе
- Алгоритм Кристофидеса — эвристический приближенный алгоритм для решения метрической задачи коммивояжера на графе.
- Метод ближайшего соседа — «жадный» алгоритм, один из простейших эвристических методов решения задачи коммивояжёра
- Неблокирующий минимальный охватывающий переключатель например, для телефонной связи
- Построение транзитивного замыкания графа (установление факта достижимости вершин)

Алгоритмы нахождения максимального потока

— число вершин, — число рёбер, — наибольшая величина максимальной пропускной способности сети.

- Алгоритм Форда — Фалкерсона (1956) — .
- Алгоритм Эдмондса — Карпа, кратчайших увеличивающихся цепей (1969) — .
- Алгоритм Диница (1970) — .
- Алгоритм Эдмондса — Карпа, локально-максимального увеличения (1972) — .
- Алгоритм Диница 2 (1973) — .
- Алгоритм Карзанова (1974) — .
- Алгоритм Черкаского (1977) — .
- Алгоритм Малхотры — Кумара — Махешвари (1977) — .
- Алгоритм Галила (1980) — .
- Алгоритм Галила — Наамада (1980) — .
- Алгоритм Слейтора — Тарьяна (1983) — .
- Алгоритм Габоу (1985) — .
- Алгоритм Голдберга — Тарьяна (1988) — .
- Алгоритм Ахьюа — Орлина (1989) — .
- Алгоритм Ахьюа — Орлина — Тарьяна (1989) — .
- Алгоритм Кинга — Рао — Тарьяна 1 (1992) — .

- Алгоритм Кинга — Рао — Тарьяна 2 (1994) — .
- Алгоритм Черияна — Хейджрапа — Мехлхорна

  (1996) — .

- Алгоритм Голдберга — Рао (1998) — .
- Алгоритм Кёлнера — Мондры — Спилмана — Тена

  (2010) — .

- Алгоритм Орлина 1 (2012) — .

- Алгоритм Орлина 2 (2012) — , если .

Алгоритмы нахождения максимального паросочетания

- Алгоритм Хопкрофта — Карпа
- Алгоритм Форда — Фалкерсона
- Алгоритм Куна
- Алгоритм Габоу

Алгоритмы поиска

- Алгоритм поиска A* — особый случай поиска по первому наилучшему совпадению; используется эвристика, увеличивающая скорость работы алгоритма
- Алгоритм выбора — модификация алгоритма

  линейного поиска; находит -й по величине элемент в списке;

- Двоичное дерево поиска , в худшем случае — использует бинарное дерево для хранения элементов;
    - Красно-чёрное дерево — использует дополнительный атрибут узла дерева — «цвет»
    - АВЛ-дерево — в каждом узле хранит разницу высот (целое число от −1 до +1)
    - Расширяющееся дерево — вместо дополнительных полей в узлах дерева «расширяющие операции» выполняются при каждом обращении к дереву.
- Двоичный поиск — находит элемент в отсортированном списке
- Интерполяционный поиск (Предсказывающий поиск, Поиск по словарю)
- Линейный поиск — находит элемент в неотсортированном списке
- Локальный поиск (оптимизация)
- Метод штрафов
- Поиск в глубину — проходит граф ветка за веткой
- Поиск в ширину — проходит граф уровень за уровнем
- Поиск по первому наилучшему совпадению (англ. *Best-first search*) — проходит граф в порядке важности, используя очередь приоритетов
- Троичный поиск — находит максимум или минимум функции
- Поиск в хеш-таблице

- Алгоритм Ли (волновой алгоритм) — поиск пути на карте.

Алгоритмы на строках

Алгоритмы поиска строки

- -функция
- Алгоритм Кнута — Морриса — Пратта
- Алгоритм Рабина — Карпа поиска строки
- Алгоритм Бойера — Мура поиска строки
  - Алгоритм Бойера — Мура — Хорспула
  - Первый алгоритм Бойера — Мура — Санди
  - Второй алгоритм Бойера — Мура — Санди
  - Алгоритм Бойера — Мура — Гелила
  - Алгоритм Турбо-БМ
- Алгоритм Ахо — Корасик
- Алгоритм Битапа (также известен как shift-or, shift-and)
- Задача поиска наибольшей общей подпоследовательности
- Задача поиска наибольшей увеличивающейся подпоследовательности
- Задача поиска наикратчайшей общей надпоследовательности (*англ.*)
- Задача поиска наибольшей общей подстроки
- Задача поиска количества подпалиндромов

Алгоритмы вычисления расстояния между строками

- Алгоритм Вагнера — Фишера
- Алгоритм Хешберга

- Алгоритм Ханта — Шиманского
- Алгоритм Укконена — Майерса

Алгоритмы приближенного сравнения строк с шаблоном

- Алгоритм Укконена
- Алгоритм Майерса
- Алгоритм Ву — Менбера

Вычисление характеристических паттернов

- Алгоритм Крочемора поиска всех кратных строк
- Алгоритм Мейна — Лоренца поиска всех кратных строк
- Алгоритм Мейна поиска крайних левых серий
- Алгоритм Колпакова — Кучерова поиска всех серий
- Алгоритм Ли — Смита поиска всех оболочек
- Алгоритм Франека — Смита — Танга поиска всех раппортов

- Алгоритм Шмидта поиска -приближенных раппортов
- Алгоритмы Сима — Илиопулоса — Парка — Смита

  поиска -приближенных периодов

Примерное соответствие

- Расстояние Левенштейна
- Расстояние Хэмминга
- Расстояние Дамерау — Левенштейна
- Алгоритм Нидлмана — Вунша
- Алгоритм Смита — Вотермана (*англ.*)

- Soundex
- Metaphone

Деревья для строковых последовательностей

- Суффиксное дерево
- Алгоритм Мак-Крейта
- Алгоритм Укконена
- Алгоритм Вайнера
- Алгоритм Фрача
- Суффиксный массив

Алгоритмы сортировки

- Bogosort
- Stooge sort
- Timsort — гибридный алгоритм сортировки, сочетающий сортировку вставками и сортировку слиянием

- Наивная сортировка — генерация всех возможных перестановок и проверка на отсортированность
- Блинная сортировка
- Блочная сортировка (также известен как *корзинная сортировка*), ср. с поразрядной
- Быстрая сортировка — с разбиением исходного набора данных на две половины так, что любой элемент первой половины упорядочен относительно любого элемента второй половины; затем алгоритм применяется рекурсивно к каждой половине
- Гномья сортировка — имеет общее с сортировкой пузырьком и сортировкой вставками. Сложность

  алгоритма — .

- Пирамидальная сортировка (Сортировка кучей) — превращаем список в кучу, берём наибольший элемент и добавляем его в конец списка
- Плавная сортировка
- Поразрядная сортировка — сортирует строки буква за буквой.
- Сортировка Бентли — Седжвика (англ. *BeSe sort*) — модификация быстрой сортировки для составных ключей, заключающаяся в делении не пополам, а на три части — в третью попадают одинаковые (по текущему символу) ключи
- Сортировка с помощью двоичного дерева (англ. *Tree sort*)
- Сортировка методом вставок — определяем, где текущий элемент должен находиться в отсортированном списке, и вставляем его туда
- Сортировка методом выбора — наименьшего или наибольшего элемента и помещения его в начало или конец отсортированного списка
- Сортировка перемешиванием (Сортировка коктейлем)
- Сортировка подсчётом — используется диапазон входных данных, подсчитывается число одинаковых элементов (3 варианта)
- Сортировка пузырьком
- Сортировка расчёской
- Сортировка слиянием — сортируем первую и вторую половину списка отдельно, а затем — сливаем отсортированные списки
- Сортировка Шелла — попытка улучшить сортировку вставками
- Топологическая сортировка
- Хитрая сортировка — извлекает из исходной последовательности отсортированные

подпоследовательности, производя их слияние с уже извлечёнными данными
- Цифровая сортировка — то же, что и Поразрядная сортировка.

Алгоритмы слияния

- Простой алгоритм слияния (англ. *Simple Merge algorithm*)
- -мерный алгоритм слияния (англ. *k-way Merge algorithm*)

Минимизация булевых функций

- Алгоритм Куайна
- Алгоритм Мак-Класки
- Карты Карно
- Алгоритм Свободы
- Алгоритм Хва
- Получение простых импликант на основе разбиения по кодам разности
- Метод поразрядного выращивания
- Метод Блейка

Алгоритмы сжатия данных

Алгоритмы сжатия без потерь

- Преобразование Барроуза — Уилера (также известен как англ. *BWT*) — предварительная обработка данных для улучшения сжатия без потерь
- Преобразование Шиндлера (англ. *ST*) — модификация преобразования Барроуза — Уилера

- Алгоритм DEFLATE — популярный свободный алгоритм сжатия (используется в библиотеке zlib)
- Дельта-кодирование — эффективно для сжатия данных с часто повторяющимися последовательностями
- Инкрементное кодирование — дельта-кодирование, применяемое к последовательности строк
- Семейство алгоритмов словарного сжатия Лемпеля — Зива:
    - LZ77 — родоначальник семейства LZ77-алгоритмов
        - LZ77-PM
        - LZFG
            - LZFG-PM
        - LZP
        - LZBW
        - LZSS
            - LZB
            - LZH
            - LZRW1
    - LZ78 — родоначальник семейства LZ78-алгоритмов
        - Алгоритм Лемпеля — Зива — Велча (также известен как англ. *LZW*) — сжатие без потерь
            - LZW-PM
        - LZMW
    - LZMA — сокращение от англ. *Lempel-Ziv-Markov chain-Algorithm*
    - LZO — алгоритм сжатия, ориентированный на скорость
- Алгоритм сжатия PPM
- Кодирование длин серий (Групповое кодирование, также известен как англ. *RLE*) — последовательная

серия одинаковых элементов заменяется на два символа: элемент и число его повторений
- Алгоритм SEQUITUR (*англ.*) — сжатие без потерь, автоматическое адаптивное построение контекстно-свободной грамматики для обрабатываемых данных
- Вейвлет-кодирование на основе вложенных нуль-деревьев (*англ.*) (EZW-кодирование)
- Энтропийное кодирование — схема кодирования, которая присваивает коды символам таким образом, чтобы соотнести длину кодов с вероятностью появления символов
    - Алгоритм Шеннона — Фано — самый простой алгоритм кодирования
    - Алгоритм Хаффмана — алгоритм построения кода при помощи кодовых деревьев
        - Адаптивное кодирование Хаффмана (*англ.*) — техника адаптивного кодирования, основывающаяся на коде Хаффмана
    - Усечённое двоичное кодирование (*англ.*) — используется для однородного вероятностного распределения с конечным алфавитом
    - Арифметическое кодирование — развитие энтропийного кодирования
        - Адаптивное арифметическое кодирование — техника адаптивного кодирования, основывающаяся на арифметическом кодировании
    - Кодирование расстояний (*англ.*) — метод сжатия данных, который близок по эффективности к арифметическому кодированию
- Энтропийное кодирование с известными характеристиками

- Унарное кодирование — код, который представляет число в виде единиц с замыкающим нулём
- дельта|гамма|омега-кодирование Элиаса (англ. *Elias coding*) — универсальный код, кодирующий положительные целые числа
- Кодирование Фибоначчи — универсальный код, который кодирует положительные целые числа в двоичные кодовые слова
- Кодирование Голомба — форма энтропийного кодирования, которая оптимальна для алфавитов с геометрическим распределением
- Кодирование Райса — форма энтропийного кодирования, которая оптимальна для алфавитов с геометрическим распределением

Алгоритмы сжатия с потерями

- Линейное предсказывающее кодирование (*англ.*) — сжатие с потерями, представляющее спектральную огибающую цифрового сигнала речи в сжатом виде
- -закон — стандартный алгоритм компандирования. Применяется в РФ.
- Мю-закон — стандартный алгоритм компандирования
- Фрактальное сжатие — метод, использующий фракталы для сжатия изображений
- Трансформирующее кодирование (*англ.*) — тип сжатия данных для «естественных» данных, таких как аудиосигналы или фотографические изображения

- Векторное квантование — техника, часто используемая в сжатии данных с потерями
- Вейвлетное сжатие — тип компрессии данных, хорошо подходящий для сжатия изображений (иногда также используется для сжатия видео и аудио)

Вычислительная геометрия

- Алгоритм Гилберта — Джонсона — Кёрти — определение наименьшего расстояния между двумя выпуклыми множествами
- Поиск пары ближайших точек (*англ.*) —

  трудоёмкость .
- Поиск диаметра множества точек
- Алгоритм Кируса — Бека — отсечение отрезков выпуклым многоугольником.
- Алгоритм Сазерленда — Ходжмана — отсечение многоугольника.
- Построения контура прямоугольников (стороны параллельны осям координат).
- Нахождение ядра многоугольника
- Регуляризация многоугольника — декомпозиция многоугольника на монотонные части.

Построение выпуклой оболочки набора точек

- Построение ВП через треугольники —

  трудоёмкость .
- Построение ВП перебором рёбер на

  принадлежность — трудоёмкость .

- Алгоритм сканирования Грэхема — трудоёмкость

  .

- Алгоритм Экла — Туссена — трудоёмкость . Улучшение алгоритма Грэхема.

- Алгоритм Эндрю — трудоёмкость . Улучшение алгоритма Грэхема.

- Алгоритм быстрой оболочки — трудоёмкость ,

  в среднем — .
- Алгоритм Киркпатрика — построение выпуклой оболочки набора точек на плоскости методом «разделяй и властвуй» через мосты. Трудоёмкость

  .
- Построение методом «разделяй и властвуй» через

  построение касательных — трудоёмкость .
- Алгоритм заворачивания подарков (Джарвиса) —

  трудоёмкость , — количество точек в выпуклой оболочке.
- Алгоритм Киркпатрика — Зейделя (*англ.*) —

  трудоёмкость , — количество точек в выпуклой оболочке.

- Алгоритм Чана — трудоёмкость , — количество точек в выпуклой оболочке.

- Инкрементальный алгоритм (fast online hull) — через построение касательных , с помощью сбалансированного дерева — .
- Приближённая выпуклая оболочка снизу (lower approximate hull) — методом полос. Трудоёмкость , где — количество полос.
- Приближённая выпуклая оболочка сверху (upper approximate hull) — методом полос. Трудоёмкость , где — количество полос.
- Алгоритм Ли (выпуклые оболочки) — построение выпуклой оболочки простого многоугольника через отрезание карманов. Трудоёмкость .

Триангуляция

- Триангуляция через поиск диагоналей — ищется диагональ, многоугольник делится на два и далее рекурсивно. Трудоёмкость .
- Триангуляция через отрезание ушей — ищется образующая треугольник диагональ, соседние с треугольником вершины — следующие претенденты на отрезание. Трудоёмкость .
- Триангуляция монотонного простого многоугольника — трудоёмкость .
- Жадная триангуляция — трудоёмкость .

- Оптимальная триангуляция — -полная задача. Суммарная длина всех рёбер минимальна среди всех триангуляций данного множества.

Триангуляция Делоне

- Итеративные алгоритмы построения триангуляции Делоне — трудоёмкость .
- Алгоритмы построения триангуляции Делоне слиянием — трудоёмкость и .
- Алгоритмы прямого построения триангуляции Делоне — трудоёмкость .
- Двухпроходные алгоритмы построения триангуляции Делоне — трудоёмкость и .
- Триангуляция Делоне с ограничениями — трудоёмкость .

Квазитриангуляция

- Алгоритм построения квазитриангуляции

Диаграмма Вороного

- Простой алгоритм построения диаграммы Вороного — трудоёмкость .
- Алгоритм построения диаграммы Вороного через заметающую прямую — трудоёмкость .

- Рекурсивный алгоритм построения диаграммы Вороного — трудоёмкость .

Локализация точки (*англ.*)

- Локализация точки для выпуклого многоугольника — время запроса .
- Локализация точки в звездном многоугольнике — время запроса .
- Алгоритм точки в многоугольнике — проверка принадлежности данной точки простому многоугольнику .
- Метод луча — принадлежность точки простому многоугольнику .
- Метод углов — принадлежность точки выпуклому многоугольнику. Трудоёмкость .
- Метод полос — простой многоугольник. Время запроса , память .
- Метод детализации триангуляции Киркпатрика — простой многоугольник. Время запроса , память .
- Трапецоидальная карта — простой многоугольник. Рандомизированный алгоритм, время запроса , память .

- Метод цепей — простой многоугольник. Время запроса , память .

Пересечения

- Алгоритм Бентли — Оттмана — поиск всех точек пересечения отрезков на плоскости , — количество точек пересечения.
- Алгоритм Чазелла — Эдельсбруннера — пересечение отрезков за .
- Определение наличия пересекающихся отрезков (*англ.*) (алгоритм Шеймоса — Гоя) — трудоёмкость .
- Алгоритм Коэна — Сазерленда — для выпуклых многоугольников. Трудоёмкость .
- Пересечения выпуклых многоугольников — трудоёмкость .
- Алгоритм Шеймоса — Хоуи — для выпуклых многоугольников методом полос. Трудоёмкость .
- Пересечения выпуклых многоугольников с заметающей прямой — трудоёмкость .
- Пересечение звёздных многоугольников — трудоёмкость .
- Пересечение полуплоскостей — трудоёмкость .
- Алгоритм Лианга — Барски (*англ.*)

- Быстрое отсечение (*англ.*)
- Алгоритм Кируса — Бека
- Николла — Ли — Николла (*англ.*)
- Алгоритм Сазерленда — Ходжмана (*англ.*)
- Алгоритм Уайлера — Атертона

Вращающиеся калиперы (*англ.*)

- Поиск диаметра множества точек через вращающиеся калиперы
- Поиск минимального по площади описанного прямоугольника для множества точек (*англ.*)
- Поиск минимального по периметру описанного прямоугольника для множества точек (*англ.*)
- Определение ширины многоугольника
- Построение суммы Минковского двух выпуклых многоугольников
- Поиск максимального расстояния между двумя множествами точек
- Поиск минимального расстояния между двумя выпуклыми многоугольниками
- Построение мостов для двух выпуклых многоугольников
- Построение критических опорных прямых для выпуклых многоугольников

Компьютерная графика

- Алгоритмы построения отрезка — алгоритмы для аппроксимации отрезка на дискретной графической поверхности
    - Алгоритм Брезенхэма — растеризует отрезок линии с заданными координатами начала и конца

  - Алгоритм DDA-линии — чертит точки двухмерного массива в форме прямой линии между двумя заданными точками (использует вычисления с плавающей точкой)
  - Алгоритм Ву — алгоритм растеризации отрезка со сглаживанием
- Алгоритм закраски с затравкой — заполняет соединённый регион многомерного массива указанным значением
- Алгоритм художника — определяет видимые части трёхмерной сцены
- Трассировка лучей — рендеринг реалистичных изображений

- Модель освещения (*англ.*)
  - Затенение по Фонгу — модель освещения и метод интерполяции в трёхмерной компьютерной графике
  - Затенение по Гуро — алгоритм моделирования различных эффектов света и цвета на поверхности объекта в трёхмерной компьютерной графике
  - Модель освещения Блинна — Фонга
  - Модель освещения Кука — Торренса

- Алгоритм сканирующей строки (*англ.*) (англ. *Scanline rendering*) — конструирует образ с помощью перемещения воображаемой линии над образом
- Глобальное освещение — рассматривает прямое освещение и отражение от других объектов
- Алгоритмы интерполяции — конструирование новых точек данных, таких как в цифровом увеличителе

- Интерполяция сплайнами (*англ.*) — уменьшение ошибки феномена Рунге

Компьютерное зрение

- Epitome (*англ.*) — представление образа или видео при помощи меньшего образа или видео

Криптографические алгоритмы

См. также Разделы в криптографии для ***аналитического глоссария***

- Шифрование с симметричным (скрытым) ключом:
  - ГОСТ 28147-89
  - AES (англ. *Advanced Encryption Standard*) — победитель соревнования NIST, также известен как Rijndael
  - Blowfish
  - DES (англ. *Data Encryption Standard*) — иногда, алгоритм DEA (англ. *Data Encryption Algorithm*), победитель соревнования NBS, заменён на AES для большинства применений
  - RC2
  - IDEA (англ. *International Data Encryption Algorithm*)
  - RC4

- Асимметричное шифрование (с публичным ключом)
  - Алгоритм Эль-Гамаля
  - RSA
  - NTRUEncrypt

- Алгоритмы выработки общего ключа
  - Алгоритм Диффи — Хеллмана

- Алгоритмы цифровой подписи:
  - ГОСТ Р 34.10-94 — устаревший российский стандарт цифровой подписи, модификация схемы Эль-Гамаля
  - ГОСТ Р 34.10-2001 — российский стандарт цифровой подписи, основанный на эллиптических кривых
  - DSA (англ. *Digital Signature Algorithm*) — базируется на схеме Эль-Гамаля
  - ECDSA (англ. *Elliptic Curve Digital Signature Algorithm*) — перенос DSA на эллиптические кривые

- Алгоритмы разделения секрета
  - Рюкзак — на данный момент доказана нестойкость схемы
  - Схема Шамира
  - Схема Blakey

- Алгоритмы подбрасывания монеты по телефону

- Доказательство с нулевым разглашением

- Криптографические функции дайджестов сообщений:
  - ГОСТ Р 34.11-94
  - MD5 Резюме сообщения 5 (Message Digest 5) Разработан Рональдом Ривестом (RFC 1321) — существует метод генерации коллизий
  - RIPEMD-160
  - SHA-1

  - НМАС — аутентификация сообщение с помощью хеш-ключа
  - Тигр — обычно используется в Тигровых деревьях хэшей
- Криптостойкие генераторы псевдослучайных чисел
  - Алгоритм Блюма — Блюма — Шуба — базируется на сложности факторизации
  - Алгоритм Ярроу
  - Фортуна (*англ.*) — якобы улучшение алгоритма Ярроу
  - Генерация случайных простых чисел
  - Алгоритм аутентификации сообщений Message authentication algorithm

Цифровая обработка сигналов

- CORDIC — быстрая техника вычисления тригонометрических функций.
- Медианный фильтр для одномерного массива
- Дождевой алгоритм (*англ.*) — уменьшает комплексную историю давлений в расчёте элементарных противодействий для использования в анализе усталости
- Osem — алгоритм для обработки медицинских изображений
- Алгоритм Гёрцеля — может быть использован для декодирования цифр тональных сигналов
- Развеяние Ричардсона — Люси (*англ.*) — алгоритм увеличения резкости образа

Разработка программного обеспечения

- Алгоритмы для восстановления и изоляции повреждённых семантик (*англ.*)
- Алгоритм сравнения Unicode (*англ.*)
- Алгоритм преобразования CHS (*англ.*) — Преобразование между системами адресации диска
- Алгоритм вычисления контрольной суммы (CRC или FCS) Циклическая избыточная сумма (англ. *Cyclic Redunancy Check*), или контрольная последовательность кадра (англ. *Frame Check Sequence*) — вычисление кода проверки.
- Чётность — Проверка четности количества единиц в двоичной записи числа. Позволяет обнаруживать ошибку в одном разряде.
- Алгоритм соединения (СУБД) — реализация операции соединения реляционной алгебры.

Алгоритмы распределённых систем

- Упорядочение Лампорта (*англ.*) — Частичное упорядочение событий в зависимости от *того, что случилось раньше*
- Алгоритм мгновенного снимка (*англ.*) — снимок процесса, записывающий глобальное состояние системы
- Векторное упорядочение (*англ.*) — Полное упорядочение событий
- Алгоритм Марзулло — распределённая синхронизация часов
- Алгоритм пересечений (*англ.*) — другой алгоритм синхронизации часов

Алгоритмы выделения и освобождения памяти

- Сборщик мусора Боема (*англ.*) — «скромный» сборщик мусора
- Дружеское выделение памяти (*англ.*) — алгоритм выделения памяти таким образом, чтобы фрагментация была наименьшей.
- Сборщик мусора с поколениями — быстрые сборщики мусора, которые разделяют память по возрасту
- Пометить и вымести (*англ.*)
- Подсчёт ссылок

Алгоритмы в операционных системах

- Алгоритм банкира (*англ.*) — Алгоритм, использующийся для избежания взаимных блокировок
- Алгоритм замены страницы (*англ.*) — выбор страницы-жертвы при условиях небольшого объёма памяти
- Адаптивный алгоритм замещения кэша (*англ.*): скорость выполнения лучше, чем у LRU
- Часы с адаптивной заменой (*англ.*) (CAR): алгоритм замены страниц со скоростью выполнения, сравнимой с адаптивным алгоритмом замещения кэша
- Алгоритм забияки (*англ.*) — выбор нового лидера среди множества компьютеров
- rsync — алгоритм, использующийся для эффективной передачи файлов между двумя компьютерами

Дисковые алгоритмы-планировщики

- Алгоритм лифта (*англ.*) — дисковый алгоритм планирования, который работает как лифт
- Алгоритм кратчайшего перемещения (*англ.*) — дисковый алгоритм планирования для уменьшения времени поиска

Сетевые алгоритмы

- Алгоритм Карна: получение точных оценок времени распространения пакетов сообщений при использовании TCP/IP
- Алгоритм Лулео (*англ.*): техника эффективного сохранения и поиска в таблицах роутинга
- Нагрузка на сеть (*англ.*)
  - Экспоненциальная задержка (*англ.*)
  - Алгоритм Нагла (*англ.*): улучшение эффективности TCP/IP за счёт объединения пакетов
  - Усечённая бинарная экспоненциальная задержка (*англ.*)
- Шейпинг
  - Алгоритм текущего ведра
  - Алгоритм ведра маркеров (*англ.*)

Алгоритмы синхронизации процессов

- Алгоритм Петерсона
- Алгоритм пекарни Лампорта
- Алгоритм Деккера

Алгоритмы планирования

- Планирование с постоянной скоростью (*англ.*)
- Первый заканчивает раньше (планирование) (*англ.*)
- Честное разделение (планирование) (*англ.*)
- Планирование Round-robin
- Многоуровневая отдача (планирование) (*англ.*) (англ. *Multi level feedback queue*)
- Кратчайшее оставшееся время (планирование) (*англ.*)
- Наименьшее время бездействия (планирование) (*англ.*)
- Списковое планирование (*англ.*) (англ. *List scheduling*)
- Алгоритм высоких вершин (*англ.*)
- Кратчайшая работа следующей
- Кратчайшее оставшееся время (*англ.*)

Генетические алгоритмы

- Выбор пропорционально пригодности (*англ.*) — также известен как **выбор рулеточного колеса**

Медицинские алгоритмы

- Медицинский алгоритм
- Техасский проект медицинских алгоритмов (*англ.*)

Нейронные сети

- Метод обратного распространения ошибки
- Самоорганизующееся отображение (Карты Кохонена, SOM)

- Метод коррекции ошибки
- Метод коррекции с обратной передачей сигнала ошибки

Вычислительная алгебра

- Алгоритм Бухбергера (*англ.*) — находит базис Грёбнера
- Процесс Грама — Шмидта — ортогонализация набора векторов
- Алгоритм пополнения Кнута — Бендикса (*англ.*)
- Алгоритм мультивариационного деления (*англ.*) — для многочленов в некоторых неопределённостях
- Алгоритмы умножения матриц
  - Алгоритм Штрассена — быстрое умножение матриц
  - Алгоритм Копперсмита — Винограда
- Умножение цепных матриц (*англ.*) (англ. *Chain matrix multiplication*)
- Алгоритм Катхилла — Макки — алгоритм уменьшения ширины ленты разреженных симметричных матриц
- Алгоритмы вычисления дискретного преобразования Фурье
  - Быстрое преобразование Фурье
  - Алгоритм БПФ Кули — Туки (*англ.*)
  - Алгоритм БПФ Рэдера (*англ.*)
  - Алгоритм БПФ Блюштейна (*англ.*)
  - Алгоритм БПФ Брууна (*англ.*)
  - Алгоритм БПФ при помощи простых сомножителей (*англ.*)
- Алгоритм нахождения собственного значения матрицы (*англ.*)

- Преобразования Хаусхолдера (-разложение) — вычисление обратной матрицы, собственных векторов и собственных значений матрицы; используется также для решения систем линейных уравнений.
- Решение систем линейных уравнений
  - Метод Гаусса (Гауссово исключение) — стандартный метод решения систем линейных уравнений
  - Структурированное гауссово исключение — применяется, когда матрица системы является разреженной
  - Метод Жордана — Гаусса — модификация метода Гаусса для матричного представления
  - Разложение Холецкого — метод, эффективный для ленточных и разреженных матриц
  - Метод Пранис — Праневича — решение систем линейных уравнений с параллельными вычислениями по компонентам

Теоретико-числовые алгоритмы

- Целочисленная арифметика (алгоритмы для работы с большими числами)
  - Умножение столбиком больших чисел
  - «Быстрый столбик»
  - Умножение Карацубы — алгоритм быстрого умножения чисел
  - Алгоритм Тоома — Кука (*англ.*) — обобщённый алгоритм умножения Карацубы (известен также как Toom-3)

  - Метод умножения Шёнхаге — Штрассена — более быстрый алгоритм умножения
  - Алгоритм Фюрера — на данный момент самый быстрый алгоритм умножения больших чисел
  - Деление на одноразрядное число (DO)
  - Деление больших чисел (*англ.*)
- Быстрое возведение в степень — вычисляет степени чисел при помощи возведения в квадрат
- Алгоритмы модулярной арифметики
  - Алгоритм Монтгомери — модулярное умножение и возведение в степень
  - Алгоритм нахождения порядка элемента
  - Алгоритм Тонелли — Шенкса — решение квадратичных сравнений
  - Решение систем линейных сравнений
    - С помощью китайской теоремы об остатках
    - Алгоритм Гарнера
- Решение систем линейных уравнений над полем
  - Алгоритм Ланцоша — эффективен над полем характеристики 2
  - Алгоритм Видемана
- Дискретное логарифмирование:
  - В простом конечном поле
    - Алгоритм Шенкса (алгоритм больших и маленьких шагов, англ. *baby-step giant-step*)
    - Алгоритм Полига — Хеллмана — эффективен, если все делители — небольшие
    - ρ-метод Полларда дискретного логарифмирования

        - Алгоритм Адлемана — первый субэкспоненциальный алгоритм дискретного логарифмирования
        - Алгоритм COS (алгоритм Копперсмита — Одлыжко — Шреппеля) — достаточно эффективный субэкспоненциальный алгоритм
        - Решето числового поля — наиболее эффективный на данный момент алгоритм дискретного логарифмирования
    - В произвольном конечном поле
        - Алгоритм исчисления индексов (алгоритм index-calculus) — сведение дискретного логарифмирования в произвольном конечном поле к аналогичной задаче в простом поле
        - Алгоритм Копперсмита — эффективный алгоритм дискретного логарифмирования в конечном поле характеристики 2
- Алгоритмы нахождения наибольшего общего делителя (НОД) двух чисел
    - Алгоритм Евклида
    - Расширенный алгоритм Евклида — также

      решает уравнение , где НОД ,

      НОД НОД
    - Бинарный алгоритм вычисления НОД — эффективный способ вычисления НОД
    - Расширенный бинарный алгоритм — модификация бинарного алгоритма

нахождения НОД, аналогичная расширенному алгоритму Евклида

- Простые числа:
  - Нахождение простых чисел:
    - Перебор делителей
    - Решето Эратосфена
    - Решето Аткина — оптимизированная версия решета Эратосфена
    - Решето Сундарама
  - Тесты простоты — проверка, является ли данное число простым:
    - Детерминированные тесты простоты:
      - Тест на основе малой теоремы Ферма
      - Тест Миллера — модификация теста на основе малой теоремы Ферма; опирается на расширенную гипотезу Римана
      - (N-1)–метод проверки простоты — тест на простоту при известном разложении на

        множители числа ; также используется для построения больших простых чисел
      - (N+1)–метод проверки простоты — тест на простоту при известном разложении на

        множители числа

- Алгоритм Конягина — Померанса — модификация

  -метода
- Алгоритм Ленстры — использует суммы Якоби и некоторые тесты, обобщающие малую теорему Ферма
- Тест Люка — Лемера для чисел Мерсенна
- Тест Пепина для чисел Ферма
- Тест Агравала — Каяла — Саксены — полиномиальный детерминированный тест простоты

- Вероятностные тесты простоты:
  - Тест Соловея — Штрассена — опирается на малую теорему Ферма и свойства символа Лежандра
  - Тест Миллера — Рабина — эффективная модификация теста Соловея — Штрассена

- Факторизация — разложение числа на простые множители:

  - Алгоритмы с экспоненциальной сложностью:
    - Перебор делителей
    - Метод факторизации Ферма

    - -алгоритм Полларда
    - ρ-метод Полларда
    - Метод Лемана

        - Алгоритм Ленстры
    - Алгоритмы с субэкспоненциальной сложностью:
        - Алгоритм Диксона
        - Метод квадратичного решета (англ. *QS*)
        - Специальный метод решета числового поля (англ. *SNFS*)
        - Общий метод решета числового поля (англ. *GNFS*)
        - Факторизация с помощью эллиптических кривых — вероятностный алгоритм факторизации с помощью эллиптических кривых

- Дроби
    - Жадный алгоритм для египетских дробей — преобразует рациональные числа в египетские дроби
- Прочие
    - Алгоритм Шуфа — вычисление порядка группы точек эллиптической кривой
    - Алгоритм Ленстры — Ленстры — Ловаса (*англ.*) (LLL-алгоритм, L^3-алгоритм)

Численные алгоритмы

Основная статья: **Численный анализ**
См. также: Список разделов численного анализа

- Алгоритм де Кастельжо — вычисление кривых Безье

- Методы интерполяции
  - Линейное сглаживание по трём точкам
  - Линейное сглаживание по пяти точкам
  - Нелинейное сглаживание по семи точкам
- Приближенное вычисление решений
  - Метод фальшпозиции (*англ.*) (*False position method, regula falsi method*) — аппроксимирует корни функции
  - Метод бисекции — нахождение нуля (корня) нелинейной функции
  - Метод Ньютона (метод касательных) — нахождение нулей функций с помощью производной
  - Метод секущих (метод хорд) — аппроксимирует корни функции
  - Метод градиентов (градиентный спуск) — аппроксимирует решение системы уравнений
  - Метод сопряжённого градиента
  - Алгоритм Гаусса — Ньютона — алгоритм для решения нелинейных уравнений методом наименьших квадратов
  - Алгоритм Левенберга — Марквардта — алгоритм для решения нелинейных уравнений методом наименьших квадратов
- Танцующие звенья (*англ.*) — нахождение всех решений задачи точного покрытия
- Алгоритм де Бора (*англ.*) — вычисление сплайнов
- Алгоритм Гаусса — Лежандра (*англ.*) — вычисление цифр числа пи
- Алгоритм Кэхэна — более аккуратный метод суммирования чисел с плавающей точкой
- Алгоритм MISER (*англ.*) — моделирование методом Монте-Карло, численное интегрирование
- Функции округления (*англ.*) — классические способы округления чисел

- [[Сдвигающий алгоритм корня -ой степени]] (*англ.*) — извлечение корня цифра за цифрой
- Вычисление квадратного корня:
  - Алгоритм Герона
  - школьный (ручной) алгоритм — аппроксимирует квадратный корень числа

- Вычисление корня -ной степени

Алгоритмы оптимизации

- Линейное программирование
- Симплекс-метод
- «Венгерский метод» — решение задач целочисленного линейного программирования
- Метод Мака решения задачи о назначениях
- Алгоритм имитации отжига
- Метод роя частиц
- Муравьиные алгоритмы
- Метод ветвей и границ
- Дифференциальная эволюция
- Эволюционная стратегия
- Метод Нелдера — Мида (downhill simplex method) — алгоритм нелинейной оптимизации
- Всхождение со случайным перезапуском (*англ.*)
- Стохастическое туннелирование (*англ.*)
- Задача о сумме подмножеств
- Метод перебора
- Метод Фибоначчи поиска экстремума — метод выбора точек для нахождения экстремума функции одной переменной
- Градиентный спуск

- Алгоритм Левенберга — Маркардта — комбинация метода Ньютона и наискорейшего спуска
- Метод Ньютона в оптимизации, основанный на методе Ньютона поиска корня
- Квазиньютоновские методы — методы, основанные на замене матрицы Гессе её приближением.
    - Алгоритм Бройдена — Флетчера — Гольдфарба — Шанно — квазиньютоновский алгоритм нелинейной оптимизации

Грамматический разбор

- Рекурсивный нисходящий парсер — нисходящий

  парсер, подходящий для -грамматик
- LL-парсер — относительно простой алгоритм разбора за линейное время для ограниченного класса контекстно-свободных грамматик
- LR-парсер (*англ.*) — более сложный алгоритм разбора за линейное время для большего класса контекстно-свободных грамматик. Варианты:
    - Парсер первенства операторов (*англ.*)
    - SLR-парсер (*англ.*) (англ. *Simple LR parser*)
    - LALR-парсер (*англ.*) (англ. *Look-ahead LR parser*)
    - Канонический LR-парсер (*англ.*)
- Парсер старьёвщика (*англ.*) — алгоритм разбора за линейное время, поддерживающий некоторые контекстно-свободные грамматики и грамматики разбора выражений
- Алгоритм Коука — Янгера — Касами (алгоритм

  CYK) — -алгоритм для разбора любой контекстно-свободной грамматики

- Алгоритм Эрли — другой -алгоритм для разбора любой контекстно-свободной грамматики
- -парсер — алгоритм для разбора любой контекстно-свободной грамматики, он предназначен для определённых грамматик, на которых он

  действует практически за линейное время и в худшем случае.
- Метод рекурсивного спуска — это один из методов определения принадлежности входной строки к некоторому формальному языку, описанному контекстно-свободной грамматикой
- Алгоритм Рутисхаузера — работает в предположении о полной скобочной структуре выражения
- Алгоритм сортировочной станции

Квантовые алгоритмы

Приложения квантовых вычислений к различным категориям проблем и алгоритмы

- Алгоритм Гровера — позволяет выполнять поиск

  среди вариантов за проверок
- Алгоритм Шора — полиномиальный алгоритм факторизации числа
- Алгоритм Дойча — Йожи — критерий баланса для булевой функции
- Задача Фейнмана

Теория вычислений и автоматов

- Конструирование набора подмножеств (*англ.*) — алгоритм для преобразования недетерминированного автомата в детерминированный
- Алгоритм Тодда — Коксетера — процедура для создания сомножеств

Другие

- Астрономические алгоритмы
- Алгоритмы восстановления фазы волнового фронта (Гершберга-Сакстона, Фиенапа, Бейтса, Альморо и др.)
- Алгоритм Баума — Велша
- Алгоритмы манипуляции битами
  - Алгоритм создания битовой маски
  - Алгоритм обмена при помощи исключающего ИЛИ — обмен значений двух переменных без использования буфера
- Алгоритм вычисления дня недели (Алгоритм Судного Дня)
- Алгоритм Шрейера — Симса (*англ.*)
- Алгоритм Витерби
- Алгоритм Луна — метод проверки правильности идентификационных чисел
- Алгоритм стеммативации (стемминга) — процесс нахождения основы слова
- Алгоритм Риша — нахождение первообразных
- Алгоритм синтеза многосвязной сети
- Алгоритм распространения доверия (алгоритм «sum-product») — алгоритм маргинализации с помощью двунаправленной передачи сообщений на графе, применяемый для вывода на графических вероятностных моделях

- Быстрый метод мультиполей — алгоритм вычисления сил взаимодействия частиц[2]

См. также

- Список структур данных
- Список классов сложности, Класс сложности
- Список основных разделов теории алгоритмов

Примечания

- ↑ В тематическом проекте есть также список терминов, относящихся к алгоритмам и структурам данных, составленный на основе словаря Американского национального института стандартов. Если Вы планируете добавить какой-либо алгоритм в этот список, убедитесь, пожалуйста, что его здесь ещё нет (возможно, алгоритм упоминается под каким-либо альтернативным названием). Внимательно посмотрите, к какой именно категории относится данный алгоритм. В случае, когда из названия не ясно, что именно делает алгоритм, напишите, пожалуйста, краткое описание. Если Вы планируете написать статью про один из алгоритмов, упомянутых в этом списке, пожалуйста, прочитайте сначала руководство «Википедия:Алгоритмы в Википедии (*англ.*)» или посмотрите несколько уже написанных статей, посвящённых алгоритмам.
- ↑ *Barry A. Cipra* The Best of the 20th Century: Editors Name Top 10 Algorithms (англ.) // SIAM News. — 2000. — Vol. 33, no. 4.

Литература

- *Ахо, Альфред, В., Хопкрофт, Джон, Ульман, Джеффри, Д.* Структуры данных и алгоритмы. — Издательский дом «Вильямс», 2000. — 384 с. — ISBN 5-8459-0122-7 (рус.) / ISBN 0-201-00023-7 (англ.).
- *Василенко О.Н.* Теоретико-числовые алгоритмы в криптографии. — Москва: МЦНМО, 2003. — 328 с. — ISBN 5-94057-103-4.
- *Дональд Кнут*. Искусство программирования, том 1. Основные алгоритмы = The Art of Computer Programming, Volume 1. Fundamental Algorithms. — 3-е изд. — М.: «Вильямс», 2006. — 720 с. — ISBN 5-8459-0080-8.
- *Дональд Кнут*. Искусство программирования, том 1, выпуск 1. MMIX — RISC-компьютеры нового тысячелетия = The Art of Computer Programming, Volume 1, Fascicle 1: MMIX — A RISC Computer for the New Millennium. — М.: «Вильямс», 2007. — 160 с. — ISBN 978-5-8459-1163-6.
- *Дональд Кнут*. Искусство программирования, том 2. Получисленные методы = The Art of Computer Programming, Volume 2. Seminumerical Algorithms. — 3-е изд. — М.: «Вильямс», 2007. — 832 с. — ISBN 5-8459-0081-6.
- *Дональд Кнут*. Искусство программирования, том 3. Сортировка и поиск = The Art of Computer Programming, Volume 3. Sorting and Searching. — 2-е изд. — М.: «Вильямс», 2007. — 824 с. — ISBN 5-8459-0082-4.
- *Дональд Кнут*. Искусство программирования, том 4, А. Комбинаторные алгоритмы, часть 1 = The Art of Computer Programming, Volume 4A: Combinatorial

Algorithms, Part 1. — М.: «Вильямс», 2013. — 960 с. — ISBN 978-5-8459-1744-7.

- *Д-р Сидни Фейт.* TCP/IP: Архитектура, протоколы, реализация (включая IP версии 6 и IP Security) = TCP/IP: Arhitecture, Protocols, and Implementation with IPv6 and IP Security. — 2nd. ed. Dr. Sidnie Feit Copyright 1997, 1993 by The McGraw-Hill Companies, Inc. (включая IP версии 6 и IP Security). — 2-е изд. — М.: Издательство «Лори», 2003. — 424 с. — ISBN 5-85582-072-6 (рус.) / ISBN 0-07-021389-5 (англ.).
- *Порублев Илья Николаевич, Ставровский Андрей Борисович.* Алгоритмы и программы. Решение олимпиадных задач. — М.: «Вильямс», 2007. — 480 с. — ISBN 978-5-8459-1244-2.
- *Томас Х. Кормен, Чарльз И. Лейзерсон, Рональд Л. Ривест, Клиффорд Штайн.* Алгоритмы: построение и анализ, 3-е издание = Introduction to Algorithms, Third Edition. — М.: «Вильямс», 2013. — 1328 с. — ISBN 978-5-8459-1794-2.

- *Роберт Седжвик.* Фундаментальные алгоритмы на C. Анализ/Структуры данных/Сортировка/Поиск = Algorithms in C. Fundamentals/Data Structures/Sorting/Searching. — СПб.: ДиаСофтЮП, 2003. — 672 с. — ISBN 5-93772-081-4.
- *Роберт Седжвик.* Фундаментальные алгоритмы на C. Алгоритмы на графах = Algorithms in C. Graph Algorithms. — СПб.: ДиаСофтЮП, 2003. — 480 с. — ISBN 5-93772-082-2.
- *Sanjoy Dasgupta, Christos H. Papadimitriou, Umesh Vazirani.* Algorithms. — The McGraw-Hill Companies, 2006. — 320 с. — ISBN 0-07-352340-2.
- *Ричард Берд.* Жемчужины проектирования алгоритмов. Функциональный подход = Pearls of

Functional Algorithm Design. — ДМК Пресс, 2013. — (Функциональное программирование). — ISBN 978-5-94074-867-0.
- Кормен Т., Лейзерсон Ч., Ривест Р. Алгоритмы: построение и анализ. — М.: МЦНМО, 2001.
- Гэри М., Джонсон Д. Вычислительные машины и труднорешаемые задачи: Пер. с. англ. — М.: Мир, 1982.
- Oded Goldreich. Introduction to Complexity Theory — Weizmann Institute of Science, Israel, 1999.

Кононюк А. Е Дискретно-непрерывная математика. (Алгоритмы). —В 12-и кн. Кн. 10,Ч.1— К.: 2017. — 608 с.

Формальные языки и грамматики

- Программирование,
- Математика,
- Алгоритмы

- Tutorial

Мотивация

Время от времени на Хабре публикуются посты и

переводные статьи, посвященные тем или иным аспектам теории формальных языков. Среди таких публикаций (не хочется указывать конкретные работы, чтобы не обижать их авторов), особенно среди тех, которые посвящены описанию различных программных инструментов обработки языков, часто встречаются неточности и путаница. Автор склонен считать, что одной из основных причин, приведших к такому прискорбному положению вещей, является недостаточный уровень понимания идей, лежащих в основании теории формальных языков.

Этот текст задуман как популярное введение в теорию формальных языков и грамматик. Эта теория считается (и, надо сказать, справедливо) довольно сложной и запутанной. На лекциях студенты обычно скучают и экзамены тем более не вызывают энтузиазма. Поэтому и в науке не так много исследователей в этой тематике. Достаточно сказать, что за все время, с зарождения теории формальных грамматик в середине 50-х годов прошлого века и до наших дней, по этому научному направлению было выпущено всего две докторских диссертации. Одна из них была написана в конце 60-х годов Алексеем Владимировичем Гладким, вторая уже на пороге нового тысячелетия — Мати Пентусом.

Далее в наиболее доступной форме описаны два основных понятия теории формальных языков: формальный язык и формальная грамматика. Если тест будет интересен аудитории, то автор дает торжественное обещание разродиться еще парой подобных опусов.

Формальные языки

Коротко говоря, формальный язык — это математическая

модель реального языка. Под реальным языком здесь понимается некий способ коммуникации (общения) субъектов друг с другом. Для общения субъекты используют конечный набор знаков (символов), которые проговариваются (выписываются) в строгом временном порядке, т.е. образуют линейные последовательности. Такие последовательности обычно называют словами или предложениями. Таким образом, здесь рассматривается только т.н. коммуникативная функция языка, которая изучается с использованием математических методов. Другие функции языка здесь не изучаются и, потому, не рассматриваются.

Чтобы лучше разобраться в том, как именно изучаются формальные языки, необходимо сначала понять, в чем заключаются особенности математических методов изучения. Согласно Колмогорову и др. (Александров А.Д., Колмогоров А.Н., Лаврентьев М.А. Математика. Ее содержание, методы и значение. Том 1. М.: Издательство Академии Наук СССР, 1956.), математический метод, к чему бы он ни применялся, всегда следует двум основным принципам:

1. Обобщение (абстрагирование). Объекты изучения в математике — это специальные сущности, которые существуют только в математике и предназначены для изучения математиками. Математические объекты образуются путем обобщения реальных объектов. Изучая какой-нибудь объект, математик замечает только некоторые его свойства, а от остальных отвлекается. Так, абстрактный математический объект «число» может в реальности обозначать количество гусей в пруду или количество молекул в капле воды; главное, чтобы о гусях и о молекулах воды можно было

говорить как о совокупностях. Из такой «идеализации» реальных объектов следует одно важное свойство: математика часто оперирует бесконечными совокупностями, тогда как в реальности таких совокупностей не существует.

2. Строгость рассуждений. В науке принято для выяснения истинности того или иного рассуждения сверять их результаты с тем, что существует в действительности, т.е. проводить эксперименты. В математике этот критерий проверки рассуждения на истинность не работает. Поэтому выводы не проверяются экспериментальным путем, но принято доказывать их справедливость строгими, подчиняющимися определенным правилам, рассуждениями. Эти рассуждения называются доказательствами и доказательства служат единственным способом обоснования верности того или иного утверждения.

Таким образом, чтобы изучать языки с помощью математических методов, необходимо сначала выделить из языка его свойства, которые представляются важными для изучения, а затем эти свойства строго определить. Полученная таким образом абстракция будет называться формальным языком — математической моделью реального языка. Содержание конкретной математической модели зависит от того, какие свойства важны для изучения, т.е. что планируется в данный момент выделить и изучать.

В качестве известного примера такой математической абстракции можно привести модель, известную под неблагозвучным для русского уха названием «мешок слов». В этой модели исследуются тексты естественного

языка (т.е. одного из тех языков, которые люди используют в процессе повседневного общения между собой). Основной объект модели мешка слов — это слово, снабженное единственным атрибутом, частотой встречаемости этого слова в исходном тексте. В модели не учитывается, как слова располагаются рядом друг с другом, только сколько раз каждое слово встречается в тексте. Мешок слов используется в машинном обучении на основе текстов в качестве одного из основных объектов изучения.

Но в теории формальных языков представляется важным изучить законы расположения слов рядом друг с другом, т.е. синтаксические свойства текстов. Для этого модель мешка слов выглядит бедной. Поэтому формальный язык задается как множество последовательностей, составленных из элементов конечного алфавита. Определим это более строго.

Алфавит представляет собой конечное непустое множество элементов. Эти элементы будем называть символам. Для обозначения алфавита обычно будем использовать латинское V, а для обозначения символов алфавита — начальные строчные буквы латинского алфавита. Например, выражение `V = {a,b}` обозначает алфавит из двух символов a и b.

Цепочка представляет собой конечную последовательность символов. Например, `abc` — цепочка из трех символов. Часто при обозначении цепочек в символах используют индексы. Сами цепочки обозначают строчными символами конца греческого алфавита. Например, `omega = a1...an` — цепочка из n символов. Цепочка может быть пустой, т.е. не содержать ни одного символа. Такие цепочки будем обозначать греческой буквой эпсилон.

Наконец, формальный язык L над алфавитом V — это произвольное множеств цепочек, составленных из символов алфавита V. Произвольность здесь означает тот факт, что язык может быть пустым, т.е. не иметь ни одной цепочки, так и бесконечным, т.е. составленным из бесконечного числа цепочек. Последний факт часто вызывает недоумение: разве имеются реальные языки, которые содержат бесконечное число цепочек? Вообще говоря, в природе все конечно. Но мы здесь используем бесконечность как возможность образования цепочек неограниченной длины. Например, язык, который состоит из возможных имен переменных языка программирования C++, является бесконечным. Ведь имена переменных в C++ не ограничены по длине, поэтому потенциально таких имен может быть бесконечно много. В реальности, конечно, длинные имена переменных не имеют для нас особого смысла т.к. к концу чтения такого имени уже забываешь его начало. Но в качестве потенциальной возможности задавать неограниченные по длине переменные, это свойство выглядит полезным.

Итак, формальные языки — это просто множества цепочек, составленных из символов некоторого конечного алфавита. Но возникает вопрос: как можно задать формальный язык? Если язык конечен, то можно просто выписать все его цепочки одну за другой (конечно, можно задуматься, имеет ли смысл выписывать цепочки языка, имеющего хотя бы десять тысяч элементов и, вообще, есть ли смысл в таком выписывании?). Что делать, если язык бесконечен, как его задавать? В этот момент на сцену выходят грамматики.

Формальные грамматики

Способ задания языка называет грамматикой этого языка.

Таким образом, грамматикой мы называем любой способ задания языка. Например, грамматика `L = {a^nb^n}` (здесь n — натуральное число) задает язык L, состоящий из цепочек вида `ab, aabb, aaabbb` и т.д. Язык L представляет собой бесконечное множество цепочек, но тем не менее, его грамматика (описание) состоит всего из 10 символов, т.е. конечна.

Назначение грамматики — задание языка. Это задание обязательно должно быть конечным, иначе человек не будет в состоянии эту грамматику понять. Но каким образом, конечное задание описывает бесконечные совокупности? Это возможно только в том случае, если строение всех цепочек языка основано на единых принципов, которых конечное число. В примере выше в качестве такого принципа выступает следующий: «каждая цепочка языка начинается с символов a, за которыми идет столько же символов b». Если язык представляет собой бесконечную совокупность случайным образом набранных цепочек, строение которых не подчиняется единым принципам, то очевидно для такого языка нельзя придумать грамматику. И здесь еще вопрос, можно или нет считать такую совокупность языком. В целях математической строгости и единообразия подхода обычно такие совокупности языком считают.

Итак, грамматика языка описывает законы внутреннего строения его цепочек. Такие законы обычно называют синтаксическими закономерностями. таким образом, можно перефразировать определение грамматики, как конечного способа описания синтаксических закономерностей языка. Для практики интересны не просто грамматики, но грамматики, которые могут быть заданы в рамках единого подхода (формализма или парадигмы). Иначе говоря, на основе единого языка (метаязыка)

описания грамматик всех формальных языков. Тогда можно придумать алгоритм для компьютера, который будет брать на вход описание грамматики, сделанное на этом метаязыке, и что-то делать с цепочками языка.

Такие парадигмы описания грамматик называют синтаксическими теориями. Формальная грамматика — это математическая модель грамматики, описанная в рамках какой-то синтаксической теории. Таких теорий придумано довольно много. Самый известный метаязык для задания грамматик — это, конечно, порождающие грамматики Хомского. Но имеются и другие формализмы. Один из таких них — окрестностные грамматики, будет описан чуть ниже.

С алгоритмической точки зрения грамматики можно подразделить по способу задания языка. Имеются три основных таких способа (вида грамматик):

- Распознающие грамматики. Такие грамматики представляют собой устройства (алгоритмы), которым на вход подается цепочка языка, а на выходе устройство печатает «Да», если цепочка принадлежит языку, и «Нет» — в противном случае.
- Порождающие грамматики. Этот вид устройств используется для порождения цепочек языков по требованию. Образно говоря, при нажатии кнопки будет сгенерирована некоторая цепочка языка.
- Перечисляющие грамматики. Такие грамматики печатают одну за другой все цепочки языка. Очевидно, что если язык состоит из бесконечного числа цепочек, то процесс перечисления никогда не остановится. Хотя, конечно его можно остановить принудительно в нужный момент времени, например, когда будет напечатана нужная цепочка.

Интересным представляет вопрос о преобразовании видов грамматики друг в друга. Можно ли, имея порождающую грамматику, построить, скажем, перечисляющую? Ответ — да, можно. Для этого достаточно генерировать цепочки, упорядочив их, скажем по длине и порядку символов. Но превратить перечисляющую грамматику в распознающую в общем случае нельзя. Можно использовать следующий метод. Получив на вход цепочку, запустить процесс перечисления цепочек и ждать, напечатает ли перечисляющая грамматика эту цепочку или нет. Если такая цепочка напечатана, то заканчиваем процесс перечисления и печатаем «Да». Если цепочка принадлежит языку, то она обязательно будет напечатана и, таким образом, распознана. Но, если цепочка не принадлежит языку, то процесс распознавания будет продолжаться бесконечно. Программа распознающей грамматики зациклится. В этом смысле мощность распознающих грамматик меньше мощности порождающих и перечисляющих. Это следует иметь ввиду, когда сравнивают порождающие грамматики Хомского и распознающие машины Тьюринга.

Окрестностные грамматики

В середине 60-х годов советский математик Юлий Анатольевич Шрейдер предложил простой способ описания синтаксиса языков на основе т.н. окрестностных грамматик. Для каждого символа языка задается конечное число его «окрестностей» — цепочек, содержащих данный символ (центр окрестности) где-то внутри. Набор таких окрестностей для каждого символа алфавита языка называется окрестностной грамматикой. Цепочка считается принадлежащей языку, задаваемому окрестностной грамматикой, если каждый символ этой цепочки входит в

нее вместе с некоторой своей окрестностью.

В качестве примера рассмотрим язык `A = {a+a, a+a+a, a+a+a+a,...}`. Этот язык представляет собой простейшую модель языка арифметических выражений, где роль чисел играет символ «a», а роль операций — символ "+". Составим для этого языка окрестностную грамматику. Зададим окрестности для символа «a». Символ «a» может встречаться в цепочках языка A в трех синтаксических контекстах: вначале, между двумя символами "+" и в конце. Для обозначения начала и конца цепочки введем псевдосимвол "#". Тогда окрестности символа «a» будут следующими: `#a+, +a+, +a#`. Обычно для выделения центра окрестности этот символ в цепочке подчеркивается (ведь в цепочке могут быть и другие такие символы, которые не являются центром!), здесь этого делать не будем за неимением простой технической возможности. Символ "+" встречается только между двух символов «a», поэтому для него задается одна окрестность, цепочка `a+a`.

Рассмотрим цепочку `a+a+a` и проверим, принадлежит ли она языку. Первый символ «a» цепочки входит в нее вместе с окрестностью `#a+`. Второй символ "+" входит в цепочку вместе с окрестностью `a+a`. Подобное вхождение можно проверить и для остальных символов цепочки, т.е. данная цепочка принадлежит языку, как и следовало ожидать. Но, например, цепочка `a+aa` языку A не принадлежит, поскольку последний и предпоследний символы «a» не имеют окрестностей, с которыми они входят в эту цепочку.

Не всякий язык может быть описан окрестностной грамматикой. Рассмотрим, например, язык B, цепочки которого начинаются либо с символа «0», либо с символа «1». В последнем случае далее в цепочке могут идти символы «a» и «b». Если же цепочка начинается с нуля, то

далее могут идти только символы «a». Нетрудно доказать, что для этого языка нельзя придумать никакой окрестностной грамматики. Легитимность вхождения символа «b» в цепочку обусловлена ее первым символом. Для любой окрестностной грамматики, в которой задается связь между символами «b» и «1» можно будет подобрать достаточно длинную цепочку, чтобы всякая окрестность символа «b» не доставала до начала цепочки. Тогда в начало можно будет подставить символ «0» и цепочка будет принадлежать языку A, что не отвечает нашим интуитивным представлениям о синтаксическом строении цепочек этого языка.

С другой стороны, легко можно построить конечный автомат, который распознает этот язык. Значит, класс языков, которые описываются окрестностными грамматиками, уже класса автоматных языков. Языки, задаваемые окрестностными грамматиками, будем называть шрейдеровскими. Таким образом, в иерархии языков можно выделить класс шрейдеровских языков, который является подклассом автоматных языков.

Можно сказать, что шрейдеровские языки задают одно простое синтаксическое отношение — «быть рядом» или отношение непосредственного предшествования. Отношение дальнего предшествования (которое, очевидно, присутствует в языке B) окрестностной грамматикой задано быть не может. Но, если визуализировать синтаксические отношения в цепочках языка, то для диаграмм отношений, в которые превращаются такие цепочки, можно придумать окрестностную грамматику.
Метки:

- формальные языки
- грамматики

Алгоритмы

1,3k авторов, 2,4k публикаций

Литература

Список литературы

1. Бусленко Н.П. Метод статистического моделирования – М.: Статистика, 1970. – 112 с.

2. Демидович Б.П., Марон И.А. Основы вычислительной математики. - М.: Наука, 1966. – 664 с.

3. Епанешников А.М., Епанешников В.А. Программирование в среде TURBO PASCAL 7.0 – М.: Диалог-МИФИ, 1998. – 288 с.

4. Ермаков С.М. Метод Монте-Карло и смежные вопросы – М.: Наука, 1975–472 с.

5. Копченова Н.В., Марон И.А. Вычислительная математика в примерах и задачах. – М.: Наука, 1972. – 367 с.

6. Соболь И.М. Метод Монте-Карло – М.: Наука, 1985. – 80 с.

- *Соболь И. М.* Метод Монте-Карло. — М.: Наука, 1968. — 64 с. — (Популярные лекции по математике). — 79 000 экз.
- Статья «Моделируя жизнь», автор Андрей Тепляков
- *Metropolis, N.*, *Ulam, S.* The Monte Carlo Method, — Journal of the American Statistical Association 1949 **44** № 247 335—341.
- *Alex F Bielajew*. Fundamentals of the Monte Carlo method for neutral and charged particle transport. 2001
- *W. M. C. Foulkes, L. Mitas, R. J. Needs and G. Rajagopal* Quantum Monte Carlo simulations of solids, — Reviews of Modern Physics 73 (2001) 33.
- Статья «Metropolis, Monte Carlo and the MANIAC»
- *Fishman, George S.* Monte Carlo : concepts, algorithms, and applications. — Springer, 1996. — ISBN 0-387-94527-X.
- Vitter's original paper: J. S. Vitter, «Проектирование и анализ динамических кодов Хаффмана», журнал ACM, 34(4), октябрь 1987 г., стр. 825—845.
- J. S. Vitter, «ALGORITHM 673 Dynamic Huffman Coding», ACM Transactions on Mathematical Software, 15(2), June 1989, pp 158–167. Also appears in Collected Algorithms of ACM.
- Donald E. Knuth, «Dynamic Huffman Coding», Journal of Algorithm, 6(2), 1985, pp 163–180.

Кононюк А. Е Дискретно-непрерывная математика. (Алгоритмы). —В 12-и кн. Кн. 10,Ч.1— К.: 2017. — 608 с.

$\dot{-}$