Системы цифрового телевидения для тех, кто хочет понять: кодирование, исправляющее ошибки

Часть 7. Начало в №№ 6…9/2020; 1,2/2021

Константин Гласман

Декодирование кодов БЧХ: алгоритм Питерсона-Горенстейна-Цирлера

Коды БЧХ, являясь циклическими, могут декодироваться с использованием конструкций декодеров на базе регистров сдвига, созданных для циклических кодов. Но для кодов БЧХ были разработаны специальные алгоритмы, обладающие лучшими характеристиками. Один из наиболее важных и широко применяемых – Питерсона-Горенстейна-Цирлера [9].

На основе общего подхода, описанного в предыдущей части (№ 2/2021, стр. 44), сформулируем задачу декодирования с использованием расширения поля применительно к кодам БЧХ. Пусть x(z) – кодовое слово, полученное на выходе канального кодера. Многочлен, описывающий принятое слово на входе декодера, можно записать в виде

$$y(z) = x(z) + e(z),$$

где x(z) – кодовый многочлен, соответствующий кодовому слову, e(z) – многочлен ошибок.

Вычислим значения этого многочлена на элементах расширенного поля Галуа $GF(q^m)$ в точках, которые являются корнями порождающего многочлена $g(z)$, то есть в точках $\alpha_1, \alpha_2, \ldots, \alpha_{2t}$. Такое вычисление дает компоненты синдрома:

$$S_j = y(\alpha^j), \quad j = 1, 2, \ldots, 2t.$$

Кодовый многочлен в этих точках равен нулю: $x(\alpha^1) = x(\alpha^2) = \ldots x(\alpha^{2t}) = 0$, так как в этих точках равен нулю порождающий многочлен. Поэтому:

$$Sj = y(\alpha j) = x(\alpha j) + e(\alpha j) = e(\alpha j), \quad j = 1, 2, \ldots, 2t.$$

Используя выражение для многочлена ошибок в форме $e(z) = e_{n-1}z^{n-1} + e_{n-2}z^{n-2} + \ldots + e_1 z + e_0$, находим

$$S_j = e_{n-1}\alpha^{j(n-1)} + e_{n-2}\alpha^{j(n-2)} + \ldots + e_1\alpha^j + e_0 = \sum_{l=0}^{n-1} e_l \alpha^{jl}, \qquad (43)$$

где $j = 1, 2, \ldots, 2t$.

Как было уже отмечено в предыдущей части, компоненты синдрома S_j не являются коэффициентами синдромного многочлена, но дают эквивалентную информацию. Итак, получена система 2t уравнений, которая содержит только величины, определяемые ошибками. Многочлен ошибок можно будет рассчитать, если решить эти уравнения относительно величин e_i. Код спроектирован для исправления не более чем t ошибок, поэтому полагаем, что не более t коэффициентов в выражении (43) отличны от нуля. Предположим, что в действительности произошло v ошибок, причем $0 \le v \le t$, и что этим ошибкам соответствуют позиции $\{i_1, i_2, \ldots, i_v\}$ и величины ошибок $\{e_{i1}, e_{i2}, \ldots, e_{iv}\}$.

Многочлен ошибок можно переписать в виде

$$e(z) = e_{i_1}z^{i_1} + e_{i_2}z^{i_2} + \ldots + e_{i_v}z^{i_v}.$$

Позиции $\{i_1, i_2, \ldots, i_v\}$ и величины ошибок $\{e_{i1}, e_{i2}, \ldots, e_{iv}\}$ неизвестны. Для исправления ошибок эти числа должны быть найдены с использованием компонентов синдрома. Первый компонент синдрома представляет собой значение принятого многочлена в точке α:

$$S_1 = y(\alpha) = x(\alpha) + e(\alpha) = e(\alpha) = e_{i_1}\alpha^{i_1} + e_{i_2}\alpha^{i_2} + \ldots + e_{i_v}\alpha^{i_v}.$$

Полученное выражение можно переписать с использованием упрощенных обозначений:

$$S_1 = Y_1X_1 + Y_2X_2 + \ldots + Y_vX_v,$$

где $Y_l = e_{i_l}$ – величина ошибки в позиции , i_l, … $X_l = \alpha^{i_l}$ – элемент поля, который можно ассоциировать с позицией ошибки i_l и который можно назвать локатором ошибки в позиции i_l. Так как порядок элемента α равен n, то все локаторы рассматриваемой конфигурации ошибок различны.

Вычисляя значения принятого многочлена при всех степенях примитивного элемента α^j (здесь $j = 1, 2\ldots, 2t$), получаем $2t$ компонентов синдрома и, соответственно, систему из $2t$ уравнений относительно неизвестных локаторов $X_1, X_2, \ldots, X_v$ и неизвестных величин ошибок $Y_1, Y_2, \ldots, Y_v$:

$$S_1 = Y_1X_1 + Y_2X_2 + \ldots + Y_vX_v$$
$$S_2 = Y_1X_1^2 + Y_2X_2^2 + \ldots + Y_vX_v^2$$
$$\ldots$$
$$S_{2t} = Y_1X_1^{2t} + Y_2X_2^{2t} + \ldots + Y_vX_v^{2t}.$$

Подробное описание алгоритма решения этой системы уравнений можно найти в литературе [9]. В соответствии с алгоритмом сначала определяется фактическое число ошибок, потом – локаторы ошибок и в завершение – величины ошибок. Это позволяет найти многочлен ошибок и выполнить оценку кодового многочлена, переданного по каналу связи.

Коды Рида-Соломона

Коды Рида-Соломона являются важным и широко применяемым подмножеством кодов БЧХ. Поле символов $GF(q)$ совпадает с полем локаторов ошибок $GF(q^m)$, то есть при конструировании кодов Рида-Соломона параметр $m = 1$. Длина кода в этом случае будет равна $n = (q^m-1) = (q-1)$. Так как поле символов и поле локаторов совпадают, то все минимальные многочлены линейны. Минимальный многочлен $f(\alpha)$ над полем Галуа $GF(q)$ для элемента α равен $(z-\alpha)$. Это позволяет записать порождающий многочлен кода как:

$$g(z) = (z-\alpha)(z-\alpha^2)\ldots(z-\alpha^{2t}). \qquad (44)$$

Степень порождающего многочлена всегда равна $2t$, поэтому параметры кода Рида-Соломона связаны соотношением $(n-k) = 2t$. Это означает, что для исправления одной ошибки надо добавить два проверочных символа. Минимальное расстояние кода равно конструктивному расстоянию: $d^* = d = n-k+1$.

В качестве примера найдем порождающий многочлен для кода Рида-Соломона над полем Галуа $GF(q) = GF(16)$, который исправляет две ошибки [10]. Представление элементов поля Галуа $GF(16)$ дано в таблице в предыдущей части (№ 2/2021, стр. 46). В соответствии с выражением (44) и заданным числом исправляемых ошибок порождающий многочлен должен содержать четыре множителя:

$$g(z) = (z-\alpha)\,(z-\alpha^2)\,(z-\alpha^3)\,(z-\alpha^4) = z^4 + \alpha^{13}z^3 + \alpha^6z^4 + \alpha^3z + \alpha^{10}.$$

Длина кодового слова равна $n = q-1 = 15$. Степень порождающего многочлена равна $n-k = 4$. Поэтому длина информационного слова равна $k = 11$. Каждый символ поля Галуа $GF(16)$ имеет длину 4 бита, поэтому информационное слово имеет длину 44 бита, а кодовый символ имеет длину 60 битов.

Код Рида-Соломона использовался в качестве внешнего кода в системах телевидения DVB первого поколения. Это код над полем Галуа из 256 элементов $GF(2^8) = GF(256)$. Поле Галуа представляет собой расширение поля Галуа из 2 элементов с помощью примитивного простого многочлена:

$$p(z) = z^8 + z^4 + z^3 + z^2 + 1.$$

Код проектировался для исправления $t = 8$ ошибок в кодовом слове, поэтому порождающий многочлен кода состоит из 16 множителей:

$$g(z) = (z + \alpha^0)(z + \alpha^1)(z + \alpha^2)\ldots(z + \alpha^{15}),$$

где $\alpha = 02H$ – примитивный элемент поля $GF(256)$.

Длина оригинального систематического кода равна $n = (q-1) = 255$. Число проверочных символов $(n-k) = 2t = 16$. Длина информационного слова равна $k = n-2t = 239$. Таким образом, оригинальный код записывается как $(255, 239)$-код Рида-Соломона над полем Галуа $GF(256)$. Число элементов поля равно 256, поэтому каждый символ и информационного и кодового слова представляет собой 8 битов или один байт.

Код применялся к пакетам транспортного потока длиной 188 байтов, которые поступали на вход канального кодера. Перед кодированием слово длиной 188 байтов дополнялось 51 нулевым байтом в старших разрядах до 239 байтов, которые и подвергались кодированию с использованием $(255, 239)$-кода Рида Соломона. После кодирова-


ISE Live
& Online.
В этом году ISE идет к вам.
Четыре события. Четыре города.
Fira de Barcelona – 1-2 июня
MAC Forum, Мюнхен – 8-9 июня
RAI, Амстердам – 15-16 июня
Evolution, Лондон – 23-24 июня
Powered by
ise digital.
cisco
А еще ISE digital – фестиваль
контента и общения.
integrated
systems
europe
A joint venture partnership of
AVIXA
CEDIA
Регистрируйтесь на www.ISEUROPE.org

ния 51 байт в старших разрядах систематического кодового слова удалялся. В канал связи поступали слова длиной *255-51 = 204* байта. Поэтому внешний код записывался в спецификациях стандарта как укороченный *(204, 188)*-код Рида Соломона.

Коды с малой плотностью проверок на четность

Концепция кодов с малой плотностью проверок на четность

Основной параметр кодирования с целью исправления ошибок в канале связи – длина кодового слова *n*. Если канал необходимо использовать эффективно, то есть передача данных должна производиться со скоростью, близкой к пропускной способности канала, и при малой вероятности ошибки, то значение *n* должно быть большим и очень большим. Главное препятствие на пути достижения этой цели – сложность оборудования и большое время вычислений, необходимых для кодирования и декодирования.

Коды с малой плотностью проверок на четность – некоторый специальный случай кодов с проверками на четность. Кодовые слова в коде с проверками на четность образуются комбинированием блоков двоичных информационных символов с блоками проверочных символов. Каждый проверочный символ есть сумма некоторой заранее указанной совокупности информационных символов. Правила построения таких символов удобно задавать с помощью порождающих и проверочных матриц. Порождающие матрицы используются при кодировании, проверочные – при декодировании.

Кодирование для кодов с проверками на четность выполняется сравнительно просто. Но реализация декодирования сталкивается с гораздо большими трудностями, особенно при больших длинах кодового слова. Поэтому целью многих исследований было отыскание специальных классов таких кодов с проверками на четность, для которых существует приемлемый метод декодирования. Коды, которые более просты в отношении алгоритма декодирования, объема памяти и числа операций, являются более предпочтительными даже в том случае, если они имеет большую вероятность ошибки.

Наибольшее различие между кодами с малой плотностью проверок на четность и классическими кодами с проверками на четность заключается в способе декодирования. Классические блоковые коды обычно декодируются с использованием алгоритмов максимального правдоподобия. Декодирование по максимуму правдоподобия было описано в предшествующих разделах. Этот метод удобен, он минимизирует вероятность ошибки. Однако практическое применение декодеров, работающих по максимуму правдоподобия и сравнивающих принятую последовательность со всеми кодовыми словами, представляет большие затруднения, если длина кодового слова велика. Это связано с тем, что объем таблицы декодирования растет экспоненциально с увеличением длины блока.

Коды с малой плотностью проверок на четность декодируются итеративными методами с использованием графического представления проверочной матрицы. Поэтому при создании кодов с малой плотностью проверок на четность в центре внимания находятся свойства проверочной матрицы.

Коды с малой плотностью проверок на четность, часто называемые кодами Галлагера по имени их изобретателя [11, 12], основаны на остроумной математической идее. Проверочная матрица кодов с малой плотностью проверок на четность состоит в основном из нулей и содержит небольшое число единиц. Код с малой плотностью проверок на четность в классическом варианте, предложенном Галлагером, – это код, проверочная матрица которого имеет размер *m×n* и содержит небольшое фиксированное число единиц *c* в каждом столбце и небольшое фиксированное число единиц *r* в каждой строке. Такой код, в котором длина блока равна *n*, называют *(n, c, r)*-кодом.

Проверочная матрица классических линейных кодов имеет линейно независимые строки. Это положение может не выполняться для кодов с малой плотностью проверок на четность. В этом смысле матрица кода с малой плотностью проверок на четность может не быть проверочной матрицей в классическом понимании. Однако простота структуры проверочной матрицы создает возможность найти простые алгоритмы декодирования. Эти алгоритмы сохраняют эффективность при очень больших длинах кодового слова. Галлагером было найдено, что для типичного *(n, c, r)*-кода с малой плотностью проверок на четность при $c \geq 3$ минимальное расстояние линейно возрастает с увеличением длины блока *n* при постоянных *c* и *r* [12].

Проверочная матрица для (16, 3, 4)-кода с малой плотностью проверок на четность

h_{ij}	j 1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
i 1	1	1	1	1												
2					1	1	1	1								
3									1	1	1	1				
4													1	1	1	1
5	1				1				1				1			
6		1				1				1				1		
7			1				1				1				1	
8				1				1				1				1
9	1					1					1					1
10		1					1					1	1			
11			1					1	1					1		
12				1	1					1					1	

Принципы декодирования кодов с малой плотностью проверок на четность

Предложенная Галлагером структура проверочной матрицы определяется следующим образом [12]. Проверочная матрица разбивается по вертикали на *c* подматриц. В каждом столбце каждой подматрицы содержится только одна единица. В первой подматрице все единицы расположены в ступенчатом порядке. Строка с номером *i* содержит единицы в столбцах с номерами с *[(i-1)r+1]*-го по *ir*-й. Остальные подматрицы представляют собой перестановки столбцов первой.

В таблице приведена проверочная матрица для кода с малой плотностью проверок на четность с параметрами *n = 16, c = 3, r = 4*, то есть для *(16, 3, 4)*-кода. Нули в таблице опущены, что позволяет более наглядно отобразить свойства матрицы. В ней можно видеть три подматрицы с размерами 4 строки и 16 столбцов: с первой строки по четвертую включительно, с пятой строки по восьмую и с девятой строки по двенадцатую. В первой подматрице группы из четырех единиц в каждой из четырех строк расположены в ступенчатом порядке. Остальные подматрицы представляют собой перестановки столбцов первой.

Проверочная матрица представляет собой компактное описание проверочных соотношений, подобных формулам (35). Это по сути система линейных однородных уравнений, называемых проверочными уравнениями. Совокупность символов, входящих в проверочное уравнение, называется проверочным множеством. Множество решений этой системы уравнений является множеством кодовых слов.

Галлагер предложил два метода решения системы уравнений и декодирования [12]: жесткий и вероятностный. В первом методе декодер сначала принимает решение о величине каждого принятого символа (нуль или единица). Затем вычисляются проверки на четность. Часть проверочных соотношений может выполняться, часть соотношений может быть неудовлетворенной. Для каждого символа определяется число вхождений в проверочные соотношения, которые оказались неудовлетворенными. Величины символов, которые содержатся в наибольшем числе невыполненных проверочных соотношений, меняются на противоположные. Затем снова вычисляются проверочные соотношения с новыми измененными значениями символов и находятся символы, входящие в наибольшее число неудовлетворенных проверочных соотношений. Процесс повторяется до тех пор, пока не будут выполняться все проверочные соотношения. В этом случае процесс декодирования завершается успешно декодированием принятого слова. Как следует из приведенного описания, метод является итеративным, процесс декодирования носит итеративный характер.

Второй – вероятностный – метод декодирования также носит итеративный характер. Но входными данными на каждом этапе являются не значения символов, а оценки вероятности того, что значения символов равны единице. Процесс повторяется до тех пор, пока не будут выполнены все проверочные соотношения. Тогда процесс декодирования завершается.

Важно отметить, что число операций на символ при каждой итерации не зависит от длины кода.

Граф Таннера

Декодирование кодов с малой плотностью проверок на четность часто представляется в графической форме с использованием графа Таннера. Методы Галлагера с использованием графа Таннера применимы к декодированию других кодов с проверками на четность. Для демонстрации метода на более простом примере рассмотрим сначала декодирование (7, 4)-кода Хэмминга, описанного на стр. 46 № 9/2020.

Проверочная матрица (7, 4)-кода Хэмминга представлена в таблице, опубликованной в ч. 4 (№ 9/2020, стр. 480). Проверочные соотношения, используемые при вычислении компонент синдрома по формулам (35), можно записать в форме системы трех линейных однородных проверочных уравнений (уравнениям присвоены номера, которые будут использоваться в дальнейшем):

$$\begin{aligned} y_1 + y_2 + y_3 + y_5 &= 0 \quad (1) \\ y_2 + y_3 + y_4 + y_6 &= 0 \quad (2) \\ y_1 + y_2 + y_4 + y_7 &= 0 \quad (3) \end{aligned} \qquad (45).$$

Рассмотрим применение графа Таннера для описания этого кода. В графе Таннера присутствует два типа вершин: вершины для символов кодовых слов (битовые вершины, или узлы) и вершины для проверочных уравнений (проверочные вершины, или узлы). На рис. 9 показаны фрагменты графа Таннера для *(7, 4)*-кода Хэмминга. На графе присутствует 7 битовых вершин, что соответствует длине слова $n = 7$. Нумерация битовых вершин совпадает с нумерацией битов в кодовом слове. На графе присутствуют три проверочных вершины, что соответствует трем проверочным соотношениям кода ($(n-k) = 3$). Нумерация проверочных вершин совпадает с нумерацией проверочных уравнений в формулах (45). Битовые вершины представлены круглыми узлами. Проверочные вершины представлены квадратными узлами. Ребра графа иллюстрируют взаимодействие битовых и проверочных вершин.

Граф Таннера является двухсторонним. Круглые узлы графа на рис. 9а показывают биты, входящие в проверочные уравнения. Ребра графа показывают, что биты 1, 2, 3, 5 образуют проверочное множество первого проверочного уравнения, биты 1, 2, 4, 7 образуют проверочное множество третьего проверочного уравнения. Рис. 9а фактически иллюстрирует первую и третью строки проверочной матрицы *(7, 4)*-кода (табл. на стр. 48 в № 9/2020). Ребра графа на рис. 9б показывают, что бит 1 входит в первое и третье проверочные уравнения. Он также иллюстрирует, что бит 7 входит в третье проверочное уравнение. Рис. 9а фактически отображает первый и седьмой столбцы проверочной матрицы *(7, 4)*-кода (табл. на стр. 48 в № 9/2020). Остальные ребра на рис. 9 не показаны, чтобы не загромождать рисунок и продемонстрировать основные особенности графа Таннера.

Рис. 10 иллюстрирует декодирование кода Хэмминга с использованием итеративного метода. В качестве принятого слова, которое должно быть декодировано, используется то же слово $y = 0000011$, которое было декодировано по методу наибольшего правдоподобия на стр 46 № 9/2020. Для наглядности линии битов с нулевыми значениями показаны зеленым цветом, линии битов с единичными значениями показаны синим цветом. Ввод принятого слова в декодер может рассматриваться как инициализация процесса, после которого этапы повторяются до тех пор, пока не будут удовлетворены все уравнения.

Сначала вычисляются проверочные соотношения. На графе это можно сделать, подсчитывая сумму числа входящих в проверочные узлы ребер синего цвета. Если число входящих ребер синего цвета является четным, уравнение удовлетворено и проверочное соотношение выполнено (это соответствует арифметике поля Галуа из двух элементов). Если число нечетное, то соотношение не выполнено. Оказываются неудовлетворенными уравнения 2 и 3 (они отмечены красными крестами).

Ребра графа, связывающие проверочные узлы удовлетворенных уравнений с битовыми узлами, показаны зеленым цветом. Ребра графа, связывающие проверочные узлы неудовлетворенных уравнений с битовыми узлами, показаны синим цветом. Определяется число вхождений в проверочные соотношения, которые оказались не выполненными, для каждого символа. Бит 5 не входит в невыполненные соотношения. Биты 1, 3, 6, 7 входят по одному разу. Биты 2 и 4 входят в невыполненные соотношения по два раза. Это максимальное число вхождений в невыполненные соотношения, поэтому значения битов 2 и 4 меняются на противоположные. Битовые узлы 2 и 4 на этом этапе показаны красным цветом.

Вторая проверка проверочных соотношений показывает, что не выполняются все проверочные соотношения. Максимальное число вхождений в невыполненные соотношения у бита 2 (битовый узел 2 на этом этапе показан красным цветом) – его значение изменяется на противоположное.

Третья проверка показывает, что теперь удовлетворяются все проверочные уравнения. Декодирование завершено. Таким образом, в качестве оценки переданного слова принимается слово $x' = 0001011$.

Описанный процесс итеративного декодирования, состоящий из трех итераций, приведен также в соответствующей таблице. Первым этапом считается инициализация, последним – объявление декодированного слова.

Описанный выше алгоритм декодирования относится к классу алгоритмов с прохождением сообщения (message-passing algorithms), поскольку его выполнение происходит в процессе прохождения принятого слова (принятого сообщения) по ребрам графа Таннера. Все алгоритмы с прохождением сообщения являются итеративными, поскольку сообщение проходит итеративным образом между вершинами битов и вершинами проверок.

Граф рис. 10 и таблицу следует рассматривать только как иллюстрацию декодирования с использованием итеративного метода на простом приме-

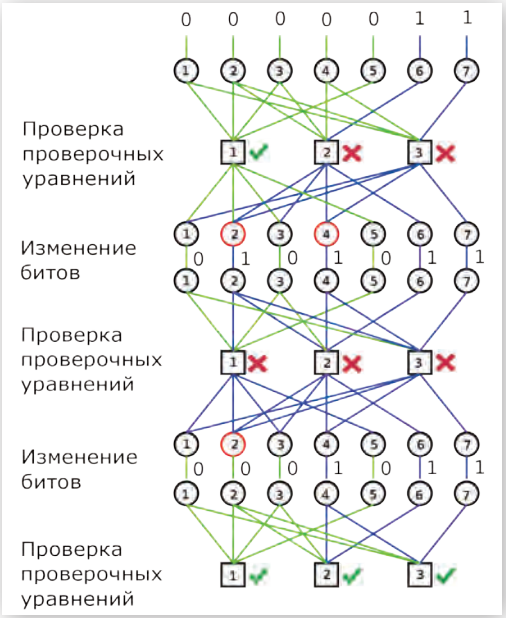

Рис. 10. Декодирование кода Хэмминга с использованием итеративного метода

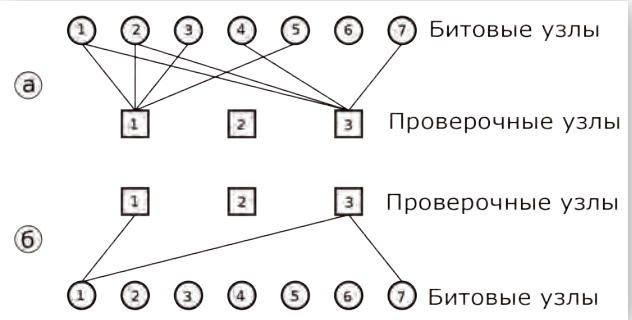

Рис. 9. Фрагменты графа Таннера для (7, 4)-кода Хэмминга

Декодирование слова (7, 4)-кода Хэмминга итеративным методом

Этап	Номер битового узла	1	2	3	4	5	6	7
1	Инициализация (принятое слово)	0	0	0	0	0	1	1
2	Проверяемое слово	0	0	0	0	0	1	1
	Число неудовлетворенных уравнений	1	2	1	2	0	1	1
3	Проверяемое слово	0	1	0	1	0	1	1
	Число неудовлетворенных уравнений	2	3	2	2	1	1	1
4	Проверяемое слово	0	0	0	1	0	1	1
	Число неудовлетворенных уравнений	0	0	0	0	0	0	0
5	Декодированное слово	0	0	0	1	0	1	1

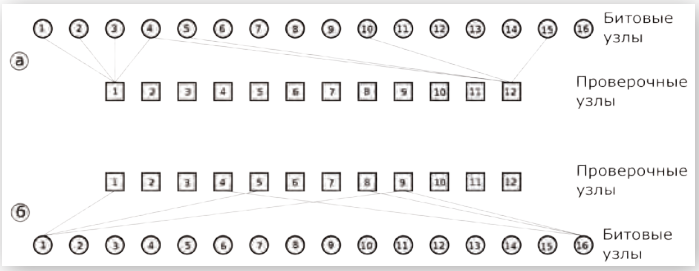

Рис. 11. Фрагменты графа Таннера для декодирования кода с малой плотностью проверок на четность

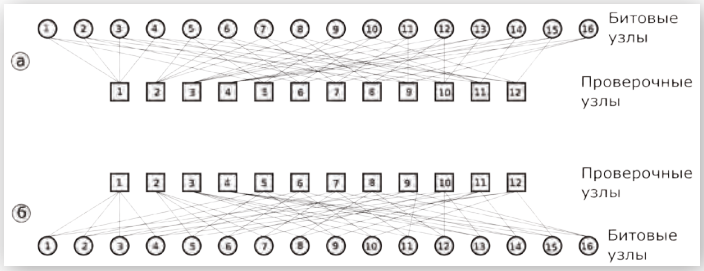

Рис. 12. Полный граф для декодирования кодовых слов

ре. *(7, 4)*-код не является кодом с малой плотностью проверок на четность. В графе есть циклы, которые не позволяют декодировать все возможные принятые слова. Алгоритм «зацикливается», если ошибка произошла в одном из битов слова, несущих проверочные символы (y_5, y_6, y_7). Для исключения циклов должна была бы быть выполнена модификация алгоритма кодирования.

Жесткий алгоритм декодирования кода с малой плотностью проверок на четность

Рассмотрим процесс декодирования с использованием жесткого итеративного метода на примере (16, 3, 4)-кода с малой плотностью проверок на четность, проверочная матрица которого приведена в табл. выше (см. стр. 42). Проверочные соотношения можно записать в форме системы двенадцати линейных однородных проверочных уравнений:

$$\begin{array}{ll} y_1+y_2+y_3+y_4=0 \quad (1) & y_3+y_7+y_{11}+y_{15}=0 \quad (7) \\ y_5+y_6+y_7+y_8=0 \quad (2) & y_4+y_8+y_{12}+y_{16}=0 \quad (8) \\ y_9+y_{10}+y_{11}+y_{12}=0 \quad (3) & y_1+y_6+y_{11}+y_{16}=0 \quad (9) \\ y_{13}+y_{14}+y_{15}+y_{16}=0 \quad (4) & y_2+y_7+y_{12}+y_{13}=0 \quad (10) \\ y_1+y_5+y_9+y_{13}=0 \quad (5) & y_3+y_8+y_9+y_{14}=0 \quad (11) \\ y_2+y_6+y_{10}+y_{14}=0 \quad (6) & y_4+y_5+y_{10}+y_{15}=0 \quad (12) \end{array} \quad (46).$$

На рис. 11 показаны фрагменты графа Таннера, который может быть использован для декодирования кода с малой плотностью проверок на четность. Каждая проверочная вершина соединяется ребрами с четырьмя битовыми узлами, поскольку в каждой строке проверочной матрицы насчитывается четыре единицы. Рис.11а отображает первое и двенадцатое проверочное уравнения, которые соответствуют первой и двенадцатой строке проверочной матрицы в табл. выше (см. стр. 42). Ребра графа соединяют проверочные вершины 1 и 12 с битовыми узлами, которые входят в первое и двенадцатое уравнения. Фрагмент графа рис. 11б показывает, в какие проверочные соотношения входят первый и шестнадцатый биты. К каждому битовому узлу подходят три ребра, поскольку в каждом столбце матрицы три единицы.

Полный граф для декодирования кодовых слов показан на рис. 12. Рис. 12а иллюстрирует проверочные уравнения (с 1-го по 12-е), которые отображают на графе строки проверочной матрицы. Ребра графа соединяют проверочные вершины с битовыми узлами, которые входят в проверочные уравнения. Граф рис. 12б показывает, в какие проверочные соотношения входят биты. Рис. 12б является зеркальной копией рис. 12а. Они оба отображают одну проверочную матрицу. Но граф рис. 12а удобен для этапа, на котором используются строки проверочной матрицы и рассчитываются проверочные соотношения. Граф рис. 12б удобен для этапа, на котором используются столбцы проверочной матрицы и рассчитывается число невыполненных проверочных соотношений.

Алгоритм декодирования реализован на основе графа рис. 12 для двух примеров принятого слова. Но выполнение процесса было бы трудно проследить в форме, подобной схеме рис. 10, при числе битовых вершин 16 и числе проверочных вершин 12. Этапы реализации отображаются в двух следующих таблицах.

Результаты вычислений примера 1 на каждом этапе реализации алгоритма представлены в таблице декодирования слова 1 для принятого слова $y = 1000010110100100$. Вычисление проверочных соотношений на этапе 2 (первая итерация после инициализации) показывает, что все биты кроме пятого и десятого входят в неудовлетворенные проверочные уравнения. Биты 3 и 16 входят в такие соотношения максимальное число раз – по три (эти вершины выделены на этом этапе синим цветом, а измененные значения битов выделены красным цветом). Их значения меняются на противоположные. После следующего этапа удовлетворяются все уравнения, поэтому в качестве оценки посланного кодового слова принимается $x' = 1010010110100101$.

Результаты вычислений для примера 2 на каждом этапе реализации алгоритма представлены в таблице декодирования слова 2 для принятого слова $y = 1000010110100001$. Вычисление проверочных соотношений на первой итерации показывает, что биты 2, 3, 14, 15 входят в неудовлетворенные проверочные уравнения наибольшее число раз – по два раза. Их величины меняются. После второго вычисления проверочных соотношений выясняется, что биты 2 и 15 входят в неудовлетворенные уравнения максимальное число раз – по три. Их значения меняются на противоположные. После третьего этапа удовлетворяются все уравнения. В качестве оценки посланного кодового слова принимается $x' = 1010010110100101$.

Выше был проиллюстрирован жесткий метод декодирования. Он хорошо показывает принцип итеративного подхода к декодированию. Особенностью итеративного подхода является возможность выполнять параллельно операции с символами принятого блока, что позволяет добиться большего быстродействия в сравнении с другими методами. Надо также отметить, что число операций на символ при декодировании итеративным методом кодов с малой плотностью проверок на четность растет самое большее логарифмически с длиной блока.

Окончание следует

Декодирование слова 1 (16, 3, 4)-кода с малой плотностью проверок на четность

Этап	Номер битового узла	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	Инициализация (принятое слово)	1	0	0	0	0	1	0	1	1	0	1	0	0	1	0	0
2	Проверяемое слово	1	0	0	0	0	1	0	1	1	0	1	0	0	1	0	0
	Число неудовлетворенных уравнений	2	1	3	2	0	1	1	2	1	0	2	1	1	2	2	3
3	Проверяемое слово	1	0	1	0	0	1	0	1	1	0	1	0	0	1	0	1
	Число неудовлетворенных уравнений	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	Декодированное слово	1	0	1	0	0	1	0	1	1	0	1	0	0	1	0	1

Декодирование слова 2 (16, 3, 4)-кода с малой плотностью проверок на четность

Этап	Номер битового узла	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	Инициализация (принятое слово)	1	0	0	0	0	1	0	1	1	0	1	0	0	0	0	1
2	Проверяемое слово	1	0	0	0	0	1	0	1	1	0	1	0	0	0	0	1
	Число неудовлетворенных уравнений	1	2	2	1	0	1	1	0	0	1	1	0	1	2	2	1
3	Проверяемое слово	1	1	1	0	0	1	0	1	1	0	1	0	0	1	1	1
	Число неудовлетворенных уравнений	1	3	2	2	1	1	2	0	0	2	1	1	2	2	3	1
4	Проверяемое слово	1	0	1	0	0	1	0	1	1	0	1	0	0	1	0	1
	Число неудовлетворенных уравнений	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	Декодированное слово	1	0	1	0	0	1	0	1	1	0	1	0	0	1	0	1