МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего образования
«Нижегородский государственный университет им. Н.И. Лобачевского»

Библиотека Исследовательской Школы
«Колебательно-волновые процессы в природных и искусственных средах»

А.В. Давыдов, А.А. Мальцев

ВВЕДЕНИЕ В ТЕОРИЮ ПОМЕХОУСТОЙЧИВОГО КОДИРОВАНИЯ

Учебно-методические материалы для магистров и аспирантов

Нижний Новгород, 2014

УДК 621.391:519.725

ВВЕДЕНИЕ В ТЕОРИЮ ПОМЕХОУСТОЙЧИВОГО КОДИРОВАНИЯ: Составители: Давыдов А.В., Мальцев А.А. Учебно-методические материалы для магистрантов и аспирантов Исследовательской школы «Колебательно-волновые процессы в природных и искусственных средах». – Нижний Новгород: Нижегородский госуниверситет, 2014. – 123 с.

Учебно-методические материалы предназначены для аспирантов ННГУ, обучающихся по направлению подготовки 03.04.03 – «Радиофизика» и для магистрантов ННГУ, обучающимся по направлению 010300 «Фундаментальная информатика и информационные технологии и по направлению 011800 «Радиофизика». В пособии изложены основные принципы построения современных помехоустойчивых кодов, а также рассматриваются реализация декодирующих и декодирующих устройств. Приводятся примеры практического применения кодов в радиотехнических системах, в том числе в современных системах сотовой связи.

Учебно-методические материалы подготовлены в соответствии с Программой повышения конкурентоспособности Нижегородского государственного университета им. Н.И. Лобачевского среди ведущих мировых научно-образовательных центров на 2013 – 2020 годы и Программой развития Нижегородского государственного университета им. Н.И. Лобачевского как национального исследовательского университета на 2009 -2018 годы.

ВВЕДЕНИЕ

Пособие составлено по материалам курса лекций «Введение в теорию помехоустойчивого кодирования» для магистрантов радиофизического факультета ННГУ. Работа содержит изложение основных принципов построения помехоустойчивых кодов, а также рассматривает реализацию кодирующих и декодирующих устройств. Приводятся примеры практического применения кодов в радиотехнических системах, в том числе в современных системах сотовой связи. Помехоустойчивому кодированию посвящено достаточно много книг и научных публикаций, однако в отечественной литературе в основном рассматриваются вопросы классических методов кодирования. Отличительной особенностью данного пособия является более детальное рассмотрение основных принципов современных методов кодирования и декодирования. Естественно, пособие не охватывает все вопросы теории. Детально не рассматриваются такие вопросы, как построение эффективных кодов, а также методы теоретической оценки помехоустойчивости кодов. Напротив, большее внимание уделяется обзору кодов, применяющихся в практических системах связи, а также описанию методов их декодирования. По мнению автора, понимание процедур декодирования является наиболее важным для разработчика современных систем связи. Автор выражает признательность сотрудникам кафедры бионики и стат. радиофизики Болховской О.В. и Артюхину И.В. за сотрудничество в процессе подготовки учебного пособия.

Глава 1

ОСНОВНЫЕ КОНЦЕПЦИИ И ЗАДАЧИ ПОМЕХОУСТОЙИВОГО КОДИРОВАНИЯ

1.1. ВВЕДЕНИЕ

В последние годы в связи с бурным развитием высокоскоростного Интернета и высокопроизводительной электроники возрастает потребность в построении надежных систем передачи и хранения данных. Одной из основных задач в таких системах является эффективная и безошибочная передача и хранение информации. Долгое время считалось, что наличие шума в канале связи препятствует безошибочной передачи данных. Однако в 1948 году Клод Элвуд Шеннон опубликовал основополагающую работу по теории информации, показывающую возможность передачи данных без ошибок в зашумленных каналах связи. Шеннон показал, что любой канал связи имеет некоторую пропускную способность C. При этом, если скорость передачи данных R не превышает пропускную способность C, то возможно добиться безошибочной передачи информации. Исправление ошибок, возникающих в канале, достигается с помощью схем помехоустойчивого (или канального) кодирования. Стоит отметить, что теорема Шеннона доказывает лишь существование таких кодов, однако не объясняет, как такие коды строить и как декодировать. Теорема Шеннона послужила отправной точкой к построению теории помехоустойчивого кодирования рассматривающей широкий круг задач разработки и анализа схем исправления ошибок.

Не смотря на большие усилия, долгое время существенного прогресса в построении эффективных кодов, позволяющих приблизится к границе Шеннона, не было. В 1970-х годах были найдены некоторые классы кодов с эффективными методами декодирования. Однако помехоустойчивость таких кодов была достаточно далека от границы Шеннона. Несмотря на это, в 1980-х годах в связи с развитием микроэлектроники схемы помехоустойчивого кодирования стали активно внедряться в различные системы передачи данных. Сейчас помехоустойчивое кодирование является неотъемлемой частью любого современного устройства передачи или хранения данных. Стоит отметить, что существенным прорывом в теории кодирования является изобретение в 1993 году класса турбо кодов и методов итеративного декодирования. Турбо коды являются первыми практическими схемами

кодирования, позволяющими передавать информацию со скоростью близкой к пропускной способности канала.

1.2. УПРОЩЕННАЯ МОДЕЛЬ СИСТЕМЫ ПЕРЕДАЧИ ИНФОРМАЦИИ

Изучение основ помехоустойчивого кодирования начнем с рассмотрения упрощенной схемы передачи или хранения данных, показанной на Рис. 1. Выходной сигнал блока источника информации поступает на вход блока кодирования источника, задача которого состоит в преобразовании входного сигнала в двоичную информационную последовательность **u** (информационное слово). Блок помехоустойчивого кодирования преобразует информационную последовательность **u** в кодовую последовательность **c** (кодовое слово). В большинстве случаев кодовая последовательность **c** является также двоичной и содержит в себе некоторую избыточность, необходимую для борьбы с ошибками, возникающими при передаче сигнала в зашумленных каналах. Мерой вносимой избыточности и, как следствие, помехоустойчивости кода является скорость кода R, определяемая как отношение длины информационной последовательности k к длине кодовой последовательности n.

Выходной сигнал блока помехоустойчивого кодирования, как правило, непригоден для прямой передачи по физическому каналу. Для преобразования кодовой последовательности в физический сигнал используется модулятор.

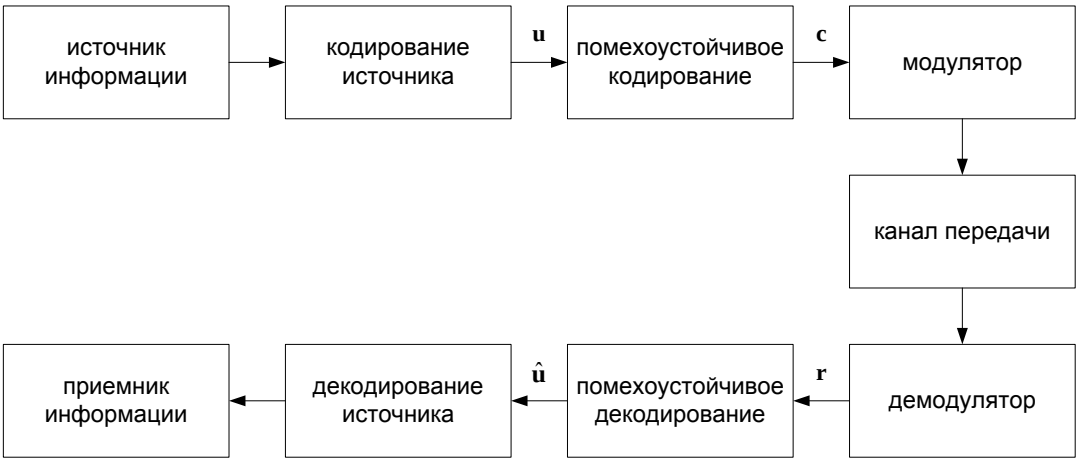


Рис. 1. Упрощенная схема передачи информации

Сигнал с выхода блока модулятора поступает в реальный канал передачи данных, где подвергается различного рода искажениям: частотно-временному замиранию (федингу), многолучевому распространению, аддитивному шуму, и т.д. Далее принятый сигнал поступает на блок демодулятора, преобразующий принятый сигнал в последовательность $\mathbf{r}$, пригодную для декодирования. Блок декодирования на основе входной последовательности осуществляет оценку $\hat{\mathbf{u}}$ переданной информационной последовательности, с использованием избыточности кода. В идеальном случае информационная последовательность $\mathbf{u}$ и ее оценка $\hat{\mathbf{u}}$ на выходе помехоустойчивого декодера должны совпадать, несмотря на возможные ошибки и искажения, возникающие при передаче сигнала. Далее блок декодирования источника преобразует последовательность $\mathbf{u}$ в форму, пригодную для приемника информации.

1.3. ДЕКОДИРОВАНИЕ ПО КРИТЕРИЮ МАКСИМУМА ПРАВДОПОДОБИЯ

Рассмотрим основные стратегии декодирования принятой последовательности. Как отмечалось ранее, задача декодирования заключается в оценке переданной информационной последовательности $\mathbf{u}$ на основе наблюдения $\mathbf{r}$. Поскольку существует взаимнооднозначное соответствие между информационной и кодовой последовательностями, схема декодирования может также проводить оценку кодовой последовательности $\hat{\mathbf{c}}$. Тогда ошибка декодирования E возникает, когда $\mathbf{c} \neq \hat{\mathbf{c}}$ или $\mathbf{u} \neq \hat{\mathbf{u}}$. При этом условная вероятность ошибки декодирования определяется как

$$P(E \mid \mathbf{r}) = P(\mathbf{c} \neq \hat{\mathbf{c}} \mid \mathbf{r}). \tag{1.3.1}$$

Тогда вероятность ошибки может быть получена

$$P(E) = \sum_{\mathbf{r}} P(E \mid \mathbf{r}) P(\mathbf{r}). \tag{1.3.2}$$

Поскольку последовательность $\mathbf{r}$ получена до декодирования, вероятность $P(\mathbf{r})$ не зависит от правила декодирования. В этом случае оптимальным декодированием принятой последовательности, обеспечивающим минимум полной вероятности ошибки $P(E)$,

является правило минимума условной вероятности $P(E \mid \mathbf{r})$, достигаемой при максимуме $P(\mathbf{c} = \hat{\mathbf{c}} \mid \mathbf{r})$.

Используя правило Байеса, условную вероятность $P(\mathbf{c} \mid \mathbf{r})$ можно записать в виде

$$P(\mathbf{c} \mid \mathbf{r}) = \frac{P(\mathbf{r} \mid \mathbf{c})P(\mathbf{c})}{P(\mathbf{r})}. \tag{1.3.3}$$

Если информационные последовательности являются равновероятными, то минимум (1.3.3) обеспечивается при максимуме $P(\mathbf{r} \mid \mathbf{c})$. Декодер, осуществляющий поиск кодовой последовательности по критерию

$$\hat{\mathbf{c}} = \arg\max_{\mathbf{c}} P(\mathbf{r} \mid \mathbf{c}) \tag{1.3.4}$$

называется *максимально правдоподобным* (МП) декодером, а оценка кодовой последовательности $\hat{\mathbf{c}}$ – *максимально правдоподобной*.

Заметим, что в литературе может также встречается модифицированный вариант декодирования по критерию максимума правдоподобия

$$\hat{c}_n = \arg\max_{c_n \in \{0,1\}} P(\mathbf{r} \mid c_n), \tag{1.3.5}$$

где поиск решения производится для каждого символа кодовой последовательности в отдельности.

Таким образом, для обеспечения минимума вероятности ошибки декодирования на приемнике, предпочтительным является использование МП декодера. Однако, как мы увидим позже, использование такого приемника на практике является затруднительным.

Рассмотрим канал без памяти, для которого вычисление вероятности $P(\mathbf{r} \mid \mathbf{c})$ можно упростить следующим образом

$$P(\mathbf{r} \mid \mathbf{c}) = \prod_i P(r_i \mid c_i). \tag{1.3.6}$$

Поскольку логарифмическая функция является монотонно возрастающей, поиск максимума выражения (1.3.5) можно также проводить после логарифмирования. Логарифм условной вероятности можно записать как сумму

$$\log\big(P(\mathbf{r} \mid \mathbf{c})\big) = \sum_i \log(P(r_i \mid c_i)). \tag{1.3.7}$$

Рассмотрим декодирование по критерию МП на примере двоичного симметричного канала с вероятностью ошибки $P(r_i \mid c_i) = p$, $r_i \neq c_i$ (см. Рис. 2).

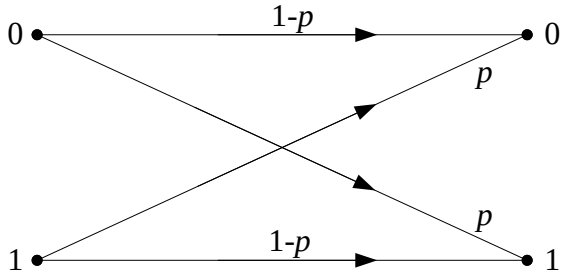


Рис. 2. Модель двоичного симметричного канала

Пусть $d(\mathbf{r},\mathbf{c})$ задает число позиций, в которых последовательности $\mathbf{c}$ и $\mathbf{r}$ отличаются. Тогда логарифм условной вероятности равен

$$\log\big(P(\mathbf{r}\mid\mathbf{c})\big)=d(\mathbf{r},\mathbf{c})\log\left(\frac{p}{1-p}\right)+n\cdot\log(1-p). \tag{1.3.8}$$

При вероятности ошибки $p<1/2$ первое слагаемое в сумме является отрицательной величиной, т.к. $\log\big(p/(1-p)\big)<0$. Поскольку второе слагаемое не зависит от $\mathbf{c}$, максимально правдоподобная оценка для двоичного симметричного канала может быть найдена поиском кодовой последовательности, имеющей минимальное расстояние $d(\mathbf{r},\mathbf{c})$. В литературе расстояние $d(\mathbf{r},\mathbf{c})$ получило название расстояния Хэмминга. При этом максимально правдоподобное декодирование эквивалентно декодированию по критерию минимума расстояния Хэмминга.

Другим популярным критерием декодирования принятой последовательности $\mathbf{r}$ является критерий *максимума апостериорной вероятности* (МАВ). Оценка для данного метода осуществляется путем поиска кодовой последовательности $\mathbf{c}$, удовлетворяющей уравнению

$$\hat{\mathbf{c}}=\arg\max_{\mathbf{c}} P(\mathbf{c}\mid\mathbf{r}). \tag{1.3.9}$$

Максимизация апостериорной вероятности может проводиться независимо для каждого символа последовательности $\mathbf{c}$. В этом случае уравнение МАВ записывается для каждого символа следующим образом

$$\hat{c}_i=\arg\max_{c_i\in\{0,1\}} P(c_i\mid\mathbf{r}). \tag{1.3.10}$$

Для равновероятных кодовых символов $P(c_i)=1/2$ можно показать, что критерии МП и МАВ эквивалентны. Действительно, используя правило Байеса, легко получить

$$\hat{c}_n = \arg\max_{c_n \in \{0,1\}} P(c_n \mid \mathbf{r}) = \arg\max_{c_n \in \{0,1\}} P(\mathbf{r} \mid c_n) \frac{P(c_n)}{P(\mathbf{r})}, \quad (1.3.11)$$

Поскольку второй множитель выражения (1.3.11) является постоянной величиной, не зависящей от c_n, решение (1.3.10) удовлетворяет уравнению (1.3.11).

Вернемся к теореме Шеннона, задающей теоретическую верхнюю границу скорости безошибочной передачи C для заданной передаваемой мощности сигнала, уровня шума и занимаемой полосы. Если скорость передачи R не превышает C, то существует код с длиной кодового слова n, такой что вероятность ошибки

$$P(E) \le 2^{-nE(R)}, \quad (1.3.12)$$

где $E(R)$ положительная функция аргумента R. Легко видеть, что произвольно маленькая вероятность ошибки может быть достигнута путем увеличения длины кодового слова n без понижения скорости передачи. Сохранение скорости передачи R и увеличение длины кодового слова n подразумевает увеличение длины информационного слова k. В свою очередь, это приводит к экспоненциальному росту числа кодовых последовательностей (для двоичного кода их число равняется 2^k). Понятно, что в силу высокой вычислительной сложности прямое использование алгоритма декодирования по критерию максимума правдоподобия на практике является затруднительным для кодов с большой длиной кодового слова. С другой стороны, использование кода с малой длиной кодовой последовательности является неэффективным. Таким образом, основными задачами теории кодирования являются

- построение кодов с большой длиной кодового слова, удовлетворяющих условию (1.3.8)
- разработка эффективных методов кодирования и декодирования таких кодов

1.4. ГЕОМЕТРИЧЕСКАЯ ИНТЕРПРЕТАЦИЯ ПОМЕХОУСТОЙЧИВОГО КОДИРОВАНИЯ

Помехоустойчивый код, как правило, описывают тремя параметрами: длиной кодовой и информационной последовательностей и минимальным расстоянием кода, задающим меру различия двух наиболее похожих кодовых слов. Минимальное расстояние кода определяется как

$$d_{\min} = \min_{\mathbf{c}_i \neq \mathbf{c}_j} d(\mathbf{c}_i, \mathbf{c}_j). \quad (1.4.1)$$

Для кода с минимальным расстоянием $d_{\min}$ число ошибок t, которое может исправить код, удовлетворяет неравенству

$$d_{\min} \geq 2t+1, \quad (1.4.2)$$

Таким образом, минимальное расстояние кода является параметром, определяющим его помехоустойчивость. При этом задачей построения эффективного кода является оптимальное размещение кодовых слов в пространстве произвольных последовательностей, с целью увеличения $d_{\min}$. Стоит отметить, что иногда удается исправить конфигурацию из числа ошибок, которые не удовлетворяют неравенству (1.4.2). Однако исправление данного числа ошибок не является гарантированным, т.к. зависит от кодового слова и конфигурации ошибок. Иллюстрация параметров кода и декодирования по минимуму расстояния показана на Рис. 3.

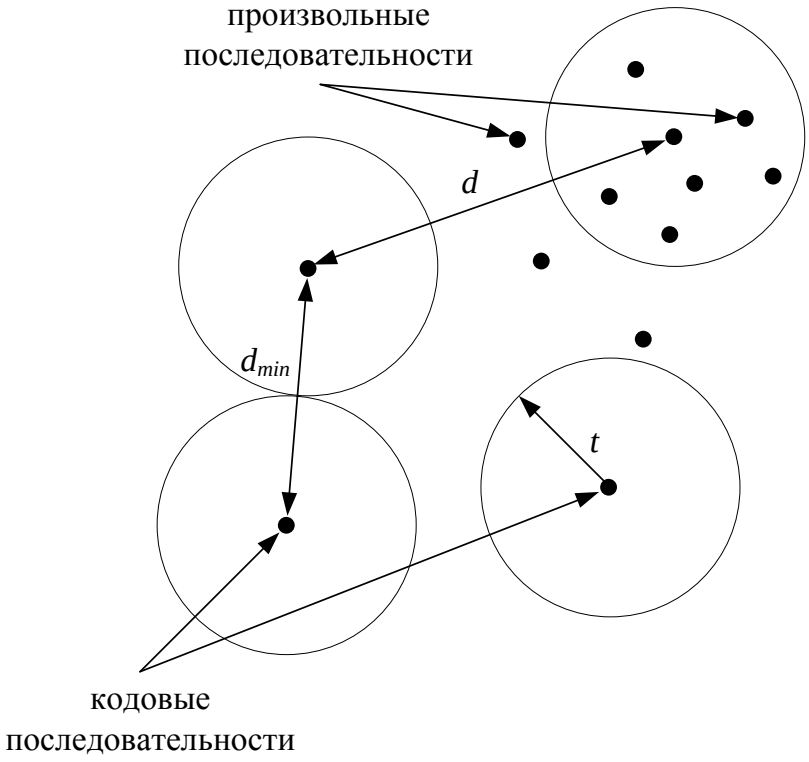


Рис. 3. Иллюстрация декодирования по минимуму расстояния

Отметим, что оптимальное размещение кодовых слов в кодовом пространстве тесно связано с теоремой Шеннона. Для иллюстрации приведем нестрогую геометрическую

интерпретацию результата этой теоремы (см. Рис. 4). Предположим, что имеется помехоустойчивый код, отображающий информационные последовательности длины k в кодовые последовательности длины n, при этом $n > k$. Общее число разрешенных кодовых последовательностей для двоичного кода равняется 2^k. Пусть P_s - средняя передаваемая мощность на бит. При этом кодовые последовательности передаются в канале с аддитивным гауссовским шумом мощности σ^2. Тогда принятая последовательность также является гауссовской со средним значением, равным переданному кодовому слову, и дисперсией $n\sigma^2$.

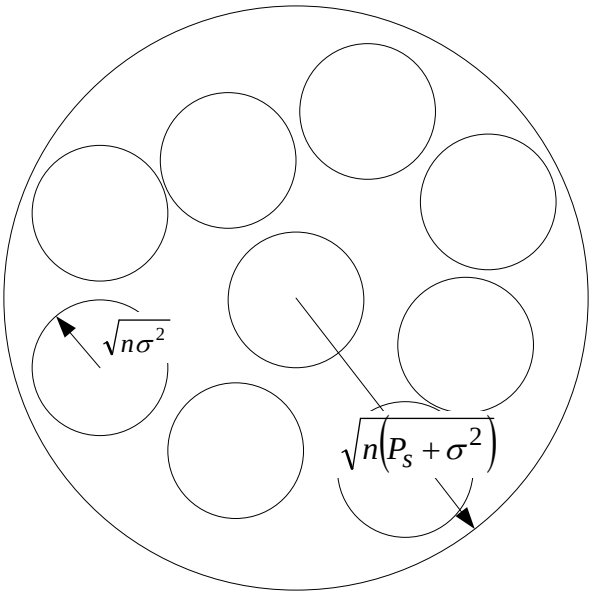


Рис. 4. Упаковка кодовых сфер

Для большой длины кодового слова принятая последовательность будет лежать на сфере радиуса $\sqrt{n\sigma^2}$ с центром, соответствующим переданной кодовой последовательности. В свою очередь, данная сфера будет лежать в сфере большего радиуса $\sqrt{n\left(P_s+\sigma^2\right)}$. Объем n-мерной сферы радиуса r задается выражением

$$V_n = A_n \cdot r^n, \tag{1.4.3}$$

где A_n масштабирующий коэффициент, зависящий от размерности пространства. Тогда максимальное число неперекрывающихся сфер, которые можно упаковать в сферу большего радиуса равно

$$M = \frac{A_n \cdot \left(n \cdot \left(P_s + \sigma^2\right)\right)^{n/2}}{A_n \cdot \left(n\sigma^2\right)^{n/2}}. \qquad (1.4.4)$$

Логарифмируя выражение (1.4.4), получим максимальное число информационных бит, которые можно передать с помощью кодовых последовательностей

$$k = \log_2(M) = \frac{n}{2}\log_2\left(1 + \frac{P_s}{\sigma^2}\right). \qquad (1.4.5)$$

В этом случае максимальная скорость передачи будет равна

$$\frac{k}{n} = \frac{1}{2}\log_2\left(1 + \frac{P_s}{\sigma^2}\right). \qquad (1.4.6)$$

Легко видеть, что выражение (1.4.6) совпадает с классическим выражением, полученным Шенноном, для пропускной способности канала с аддитивным белым гауссовским шумом.

1.5. ПРОСТЕЙШИЕ МЕТОДЫ ПОМЕХОУСТОЙИВОГО КОДИРОВАНИЯ

Двоичный код с проверкой на четность является простейшим кодом с высокой скоростью кода. Для кода с проверкой на четность к заданным k информационным битам добавляется один проверочный бит так, чтобы полное число единиц в кодовой последовательности было четным. Этот код имеет минимальное расстояние равное двум, и позволяет обнаруживать, но не исправлять ошибки.

Простейшим кодом позволяющим обнаруживать и исправлять ошибки является двоичный код с повторением. Данный код отображает каждый информационный бит в кодовую последовательность длины n, каждый элемент которой равен информационному биту. Число ошибок, которые позволяет исправлять данный код, равно $t = \lfloor (n-1)/2 \rfloor$. Вероятность ошибки для такого кода может быть записана как

$$P(E) = \sum_{i=t+1}^{n} P_i, \qquad (1.5.1)$$

где P_i определяет вероятность возникновения i ошибок. Для двоичного канала без памяти вероятность появления i ошибок на длине кодового слова n равна

$$P_i = C_n^i p^i (1-p)^{n-i}, \qquad (1.5.2)$$

где p вероятность ошибки. Тогда суммарная вероятность ошибки равна

$$P(E) = C_n^{t+1}\left(\frac{p}{1-p}\right)^{t+1}(1-p)^n + C_n^{t+2}\left(\frac{p}{1-p}\right)^{t+2}(1-p)^n + \ldots + C_n^n\left(\frac{p}{1-p}\right)^n(1-p)^n. \qquad (1.5.3)$$

Средние вероятности ошибок, полученных с помощью выражения (1.5.3) для различных скоростей кода, приведены в Таблице 1.

Таблица 1. Вероятность ошибки декодирования для кода с повторениями

Скорость кода (R)	Вероятность ошибки $P(E)$
1	10^{-2}
1/3	$3\ 10^{-4}$
1/5	10^{-6}
1/7	$4\ 10^{-7}$
1/9	10^{-8}
1/11	$5\ 10^{-10}$

При увеличении длины кодового слова n возрастает число ошибок t, которые позволяет исправлять код. При этом уменьшается вероятность ошибки $P(E)$. Однако скорость кода также стремится к нулю $R = 1/n \to 0$. Очевидно, что коды с повторениями не гарантируют безошибочную передачу на скоростях, близких к пропускной способности канала.

Глава 2

ЛИНЕЙНЫЕ БЛОКОВЫЕ КОДЫ

2.1. ОПРЕДЕЛЕНИЕ ЛИНЕЙНЫХ БЛОКОВЫХ КОДОВ

В этой главе мы рассмотрим класс линейных блоковых кодов. Будем считать, что элементы кодовой последовательности линейного кода принадлежат конечному полю $GF(q)$[1]. Линейность кода облегчает поиск эффективных кодов и позволяет получить существенные упрощения в процедурах кодирования и декодирования.

Линейный блоковый код C длины n есть подпространство над $GF^n(q)$. Размерностью k кода будем называть размерность этого подпространства. Кодовые слова линейного блокового кода это последовательности длины n над $GF(q)$, где каждый элемент последовательности является элементом из множества $GF(q)$

$$\mathbf{c} = (c_1, c_2, \ldots, c_n), \quad c_i \in GF(q) \tag{2.1.1}$$

Поскольку кодовые слова являются элементами подпространства, то линейная комбинация двух кодовых слов также является кодовым словом.

$$\mathbf{c} = \alpha \cdot \mathbf{c}_1 + \beta \cdot \mathbf{c}_2 = \left(\alpha \cdot c_1^1 + \beta \cdot c_1^2, \alpha \cdot c_2^1 + \beta \cdot c_2^2, \ldots, \alpha \cdot c_n^1 + \beta \cdot c_n^2\right), \quad c_i, \alpha, \beta \in GF(q) \tag{2.1.2}$$

В частности нулевая последовательность

$$\mathbf{c}_0 = (0,0,\ldots,0) \tag{2.1.3}$$

также является кодовым словом. Поскольку код C является векторным пространством, то он имеет базис, образующий это пространство. Размерность базиса образующего подпространство линейного блокового кода равна k. Обозначим базис подпространства, образующего код C (т.е. совокупность k линейно независимых векторов), как $\{\mathbf{g}_0, \mathbf{g}_1, \ldots, \mathbf{g}_{k-1}\}$. Тогда любое кодовое слово $\mathbf{c}$ может быть представлено как линейная комбинация базисных векторов

[1] Детальный обзор теории конечных полей выходит за рамки данной работы, однако некоторые полезные сведения приводятся в Главе 4.1.

$$\mathbf{c} = \sum_{i=0}^{k-1} u_i \cdot \mathbf{g}_i, \quad u_i \in GF(q). \tag{2.1.4}$$

Объединяя элементы u_i в вектор строку $\mathbf{u} = (u_0, u_1, \ldots, u_{k-1})$, а векторы $\mathbf{g}_i$ в матрицу $\mathbf{G}$ выражение (2.1.4) может быть переписано в матрично-векторном виде

$$\mathbf{c} = \mathbf{u} \cdot \mathbf{G}. \tag{2.1.5}$$

Вектор $\mathbf{u}$ будем называть информационным, а вектор $\mathbf{c}$ кодовым. Пусть C является линейным блоковым (n,k) кодом, определяемый базисом $\{\mathbf{g}_0, \mathbf{g}_1, \ldots, \mathbf{g}_{k-1}\}$. Тогда $k \times n$ матрица

$$\mathbf{G} = \begin{bmatrix} \mathbf{g}_0 \\ \mathbf{g}_1 \\ \vdots \\ \mathbf{g}_{k-1} \end{bmatrix}, \tag{2.1.6}$$

называется *порождающей матрицей* кода C.

Поскольку код C является подпространством $GF^n(q)$, он имеет нулевое (ортогональное), пространство размерности $n-k$. Будем называть это подпространство *дуальным кодом* и обозначать $C^{\perp}$. Очевидно, что $C^{\perp}$ является подпространством и имеет порождающую матрицу $\mathbf{H}$ размерности $n-k \times n$. Поскольку нулевое подпространство состоит из векторов, которые ортогональны каждому вектору исходного подпространства, то для каждого кодового вектора должно выполняться условие

$$\mathbf{c}\mathbf{H}^T = \mathbf{0}. \tag{2.1.7}$$

Матрица $\mathbf{H}$, задающая базис нулевого подпространства линейного кода C, называется *проверочной матрицей* кода. Выражение (2.1.7) может быть использовано в качестве проверки последовательности $\mathbf{v}$. Если выражение (2.1.7) выполняется, то вектор принадлежит кодовому подпространству, а значит, является кодовой последовательностью. Заметим, что для данного кода, матрицы $\mathbf{G}$ и $\mathbf{H}$ могут быть заданы несколькими способами.

Матрично-векторное определение кода дает существенное преимущество для практической реализации кодирования. Например, (64,32) двоичный код (кодовые слова являются последовательностями элементов из множества $GF(2)$) имеет $2^{32} = 4.29 \cdot 10^9$ кодовых слов длины 64. Для хранения всех кодовых слов потребовалось бы ~35 Гбайт памяти. Однако, если код (64,32) является линейным, то для хранения порождающей матрицы кода требуется всего 256 байт памяти.

Дальнейшее упрощение процедуры кодирования возможно при использовании *систематического кода*, у которого каждое кодовое слово содержит информационную последовательность. Оставшиеся символы кодового слова называются *проверочными символами*. Порождающая матрица линейного систематического кода имеет специальную форму. Для систематического кода содержащего информационную последовательность в начале кодового слова порождающая матрица кода имеет вид

$$\mathbf{G} = \left[\mathbf{I} \mid \mathbf{P}\right] \tag{2.1.8}$$

где $\mathbf{I}$ - единичная матрица размерности $k \times k$, и $\mathbf{P}$ матрица размерности $k \times n-k$, определяющая проверочные символы (биты проверки на четность). Проверочная матрица систематического кода также имеет определенный вид. Например, для систематического кода вида (2.1.8) проверочная матрица имеет следующую структуру

$$\mathbf{H} = \left[-\mathbf{P}^T \mid \mathbf{I}\right]. \tag{2.1.9}$$

Легко проверить условие ортогональности базисных векторов

$$\mathbf{G}\mathbf{H}^T = \left[\mathbf{I} \mid \mathbf{P}\right]\begin{bmatrix} -\mathbf{P} \\ \mathbf{I} \end{bmatrix} = -\mathbf{P} + \mathbf{P} = \mathbf{0}. \tag{2.1.10}$$

Рассмотрим пример систематического линейного кода с порождающей матрицей

$$\mathbf{G} = \begin{pmatrix} 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 \end{pmatrix}$$

Используя свойство (2.1.9) проверочная матрица данного кода может быть записана в следующем виде

$$\mathbf{H} = \begin{pmatrix} 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \end{pmatrix}.$$

Вектору информационной последовательности $\mathbf{u} = (u_0, u_1, u_2)$ соответствует кодовое слово $\mathbf{c} = (u_0 + u_2, u_0 + u_1, u_1 + u_2, u_0, u_1, u_2)$. На Рис. 5 представлена схема кодирования информационной последовательности для такого кода.

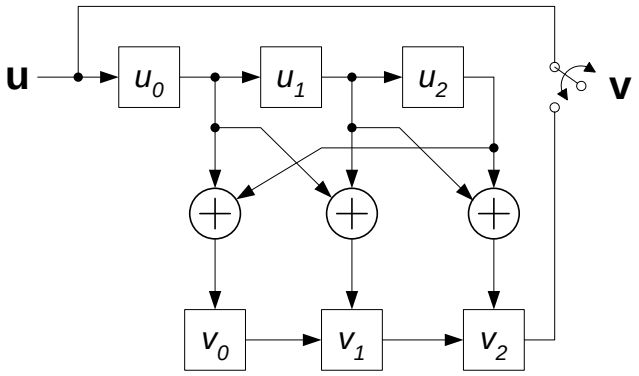


Рис. 5 Схема кодирования (6,3) линейного блокового кода

2.2. СВОЙСТВА ЛИНЕЙНЫХ БЛОКОВЫХ КОДОВ

Линейные коды имеют ряд свойств, которые позволяют упростить процедуру декодирования. Приведем наиболее важные.

Свойство 1. Линейная комбинация двух кодовых слов также является кодовым словом.

Доказательство: Определение линейного кода, как подпространства над $GF^n(q)$, подразумевает, что линейная комбинация двух векторов данного подпространства также принадлежит подпространству, и, следовательно, является кодовым словом.

Введем понятие *веса Хэмминга* $w(\mathbf{c})$ слова $\mathbf{c}$, как число его ненулевых компонент.

Свойство 2. Минимальное расстояние линейного кода есть минимальный вес ненулевого кодового слова.

Доказательство: По определению минимального расстояния кода

$$d_{\min} = \min_{\mathbf{c}_1,\mathbf{c}_2 \in C} d(\mathbf{c}_1,\mathbf{c}_2) = \min_{\mathbf{c}_1,\mathbf{c}_2 \in C} w(\mathbf{c}_1 - \mathbf{c}_2) \qquad (2.2.1)$$

Поскольку код является линейным, то линейная комбинация кодовых слов $\mathbf{c}_1 - \mathbf{c}_2$ также является кодовым словом. Следовательно, минимальное расстояние кода может быть представлено

$$d_{\min} = \min_{\mathbf{c} \in C, \mathbf{c} \neq 0} w(\mathbf{c}). \qquad (2.2.2)$$

Свойство 3. Пусть код C имеет проверочную матрицу $\mathbf{H}$. Тогда минимальное расстояние кода C равно минимальному числу линейно зависимых столбцов матрицы $\mathbf{H}$.

Доказательство: Пусть $\mathbf{c}$ является кодовым словом кода C. Из определения проверочной матрицы кода следует $\mathbf{Hc}^T = \mathbf{0}$. Если кодовое слово $\mathbf{c}$ имеет вес $w(\mathbf{c})$, тогда произведение $\mathbf{Hc}^T$ есть линейная комбинация $w(\mathbf{c})$ столбцов матрицы $\mathbf{H}$. Следовательно, кодовое слово веса w существует тогда и только тогда, когда $\mathbf{H}$ содержит w линейно зависимых столбцов.

Введем понятие эквивалентных кодов. Два кода (n,k) называются *эквивалентными*, если они имеют одинаковую дистанционную структуру. Заметим, что изменение порядка передачи символов кодового слова не меняют дистанционные свойства кода. Изменение порядка передачи символов кодового слова соответствует перестановке столбцов порождающей матрицы.

Два кода эквивалентны тогда и только тогда, когда порождающие матрицы данных кодов получены одна из другой с помощью следующих преобразований

- Перестановка столбцов
- Элементарные операции над строками

Поскольку строки порождающей матрицы являются базисом линейного подпространства, то линейные операции над строками порождающей матрицы не изменяют кодовое пространство. В частности, с помощью линейных комбинаций строк и перестановок столбцов любой код можно свести к систематическому виду.

2.3. ЭЛЕМЕНТАТРНЫЕ ГРАНИЦЫ ПАРАМЕТРОВ ЛИНЕЙНЫХ БЛОКОВЫХ КОДОВ

Границы связывают в виде неравенства помехоустойчивость кода $d_{\min}$ с параметрами кода (n,k). Простейшей является граница Синглтона, ограничивающая минимальное расстояние кода сверху

$$d_{\min} \le n-k+1 \qquad (2.3.1)$$

В справедливости неравенства (2.3.1) легко убедится при рассмотрении систематического (n,k) кода. Для такого кода существуют слова с одним ненулевым информационным символом и $n-k$ проверочными символами. Такое кодовое слово не может иметь вес, больший $\underbrace{n-k}_{\text{проверочные}} + \underbrace{1}_{\text{систематические}}$. Таким образом, минимальный вес кода не может быть больше $n-k+1$.

Граница Синглтона дает простое соотношение между избыточностью кода $r = n - k$ (число проверочных символов) и возможным минимальным расстоянием кода (см. Рис. 6). Код, удовлетворяющий границе Синглтона, называется *кодом с максимальным расстоянием*.

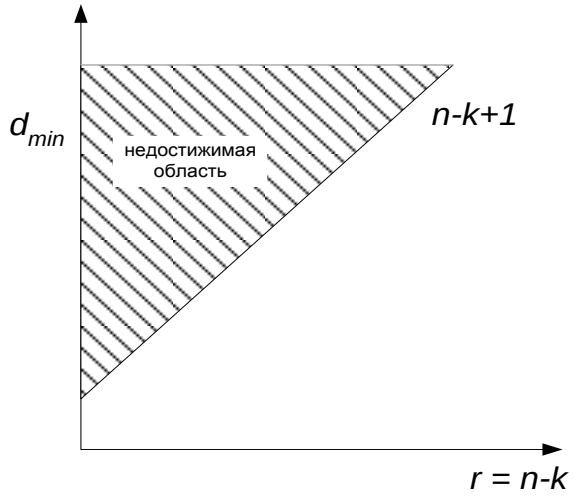


Рис. 6 Граница Синглтона линейных блоковых кодов

Другой границей задающей соотношение между избыточностью и исправляющей способностью кода является граница Хэмминга

$$n - k \geq \log_2\left(1 + C_n^1 + C_n^2 + \ldots + C_n^t\right) \qquad (2.3.2)$$

Доказательство справедливости равенства мы предоставляем читателю.

2.4. СТАНДАРТНОЕ РАСПОЛОЖЕНИЕ И СИНДРОМНОЕ ДЕКОДИРОВАНИЕ ЛИНЕЙНЫХ БЛОКОВЫХ КОДОВ

В этом разделе мы рассмотрим метод записи всех возможных последовательностей длины n для эффективного декодирования принятых последовательностей. По определению линейного кода, разность двух кодовых слов также является кодовым словом. Если известно множество последовательностей, лежащих ближе всего к нулевому кодовому слову, то с помощью сдвига начала координат можно определить множество принятых последовательностей, лежащих ближе всего к любому другому кодовому слову.

Пусть минимальное кодовое расстояние кода нечетно и равно $d_{\min} = 2t + 1$. Внутри сферы радиуса t с центром в нулевом кодовом слове находится множество точек

$$S_0 = \{\mathbf{v} \mid d(\mathbf{0}, \mathbf{v}) \le t\}. \quad (2.4.1)$$

Эта сфера содержит все последовательности, которые декодируются в нулевое кодовое слово (декодирования по минимуму расстояния). Внутри сферы радиуса t с центром в кодовом слове $\mathbf{c}$ находится множество точек

$$S_c = \{\mathbf{v} \mid d(\mathbf{c}, \mathbf{v}) \le t\}, \quad (2.4.2)$$

тогда

$$S_c = S_0 + \mathbf{c} = \{\mathbf{v} + \mathbf{c} \mid \mathbf{v} \in S_0\}. \quad (2.4.3)$$

Таким образом, достаточно знать точки лежащие внутри сферы с центром в нулевом кодовом слове. Точки лежащие внутри сферы с центром в любом другом кодовом слове находится простым сдвигом.

Стандартное расположение представляет собой способ описания данных сфер. Пусть заданы $\mathbf{c}_0 = \mathbf{0}, \mathbf{c}_1, \ldots, \mathbf{c}_{q^k - 1}$ кодовые слова. Построим таблицу расположения последовательностей, изображенную на Рис. 7, следующим образом

			сфера декодирования		область декодирования
	$\mathbf{c}_0 = \mathbf{0}$	$\mathbf{c}_1$	$\mathbf{c}_2$		$\mathbf{c}_{q^k-1}$
	$\mathbf{0} + \mathbf{v}_1$	$\mathbf{c}_1 + \mathbf{v}_1$	$\mathbf{c}_2 + \mathbf{v}_1$		$\mathbf{c}_{q^k-1} + \mathbf{v}_1$
смежный класс	$\mathbf{0} + \mathbf{v}_2$	$\mathbf{c}_1 + \mathbf{v}_2$	$\mathbf{c}_2 + \mathbf{v}_2$		$\mathbf{c}_{q^k-1} + \mathbf{v}_2$
	$\mathbf{0} + \mathbf{v}_j$	$\mathbf{c}_1 + \mathbf{v}_j$	$\mathbf{c}_2 + \mathbf{v}_j$		$\mathbf{c}_{q^k-1} + \mathbf{v}_j$
	$\mathbf{0} + \mathbf{v}_{j+1}$	$\mathbf{c}_1 + \mathbf{v}_{j+1}$	$\mathbf{c}_2 + \mathbf{v}_{j+1}$		$\mathbf{c}_{q^k-1} + \mathbf{v}_{j+1}$
	$\mathbf{0} + \mathbf{v}_l$	$\mathbf{c}_1 + \mathbf{v}_l$	$\mathbf{c}_2 + \mathbf{v}_l$		$\mathbf{c}_{q^k-1} + \mathbf{v}_l$

Рис. 7 Стандартное расположение

В первой строке выпишем все кодовые слова. Из всех оставшихся в $GF^n(q)$ последовательностей (последовательностей не являющихся кодовыми), лежащих на расстоянии 1 от нулевой, выберем любую и обозначим как $\mathbf{v}_1$. Во второй строке выпишем $\mathbf{0}+\mathbf{v}_1,\ \mathbf{c}_1+\mathbf{v}_1,\ldots,\mathbf{c}_{q^k-1}+\mathbf{v}_1$. Таким же образом строим следующие строки. На j-ом шаге выберем слово $\mathbf{v}_j$, которое является ближайшим к нулевому и отсутствует в предыдущих строках. Запишем в j-ой строке $\mathbf{0}+\mathbf{v}_j,\ \mathbf{c}_1+\mathbf{v}_j,\ldots,\mathbf{c}_{q^k-1}+\mathbf{v}_j$. Процедура закончится, когда после очередного шага не останется неиспользованных последовательностей. Горизонтальная пунктирная черта соответствует последней последовательности имеющей вес $w(\mathbf{v}_j)=t$. Процедура заполнения таблицы стандартного расположения проиллюстрирована на Рис. 8.

Существует два класса декодеров:

1. *Полные декодеры* сопоставляют каждой принятой последовательности ближайшее кодовое слово.
2. *Неполные декодеры* сопоставляют каждой принятой последовательности ближайшее кодовое слово в сфере радиуса t. В случае если принятая последовательность не попадает ни в одну сферу, декодер отказывается от декодирования.

Таким образом, неполные декодеры могут обрабатывать только те последовательности, которые встречаются до пунктирной линии.

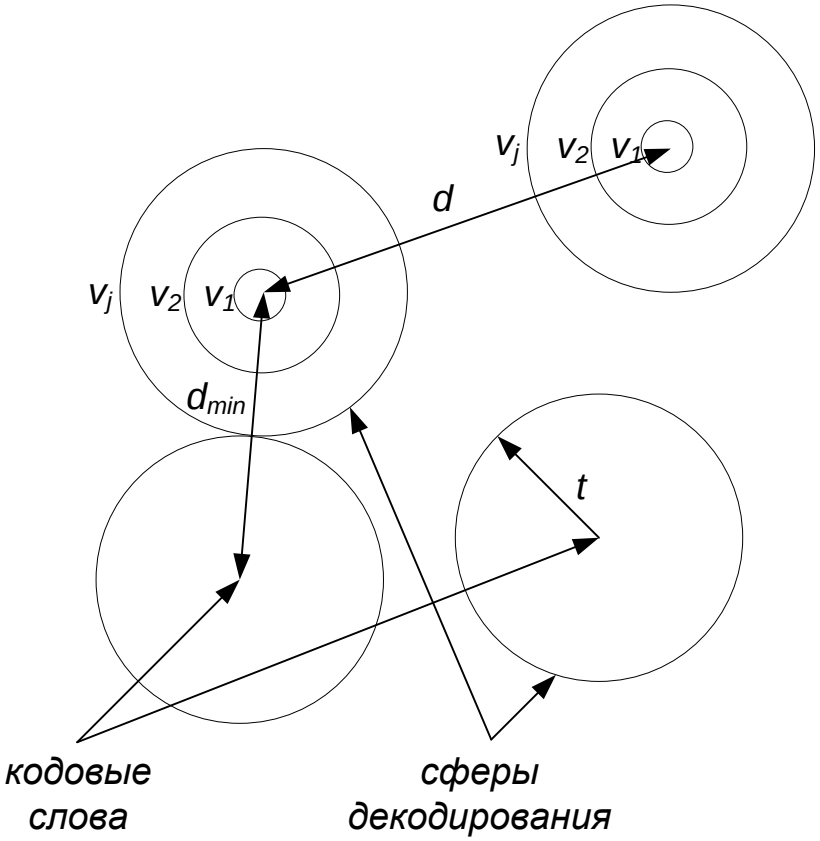


Рис. 8 Иллюстрация к построению стандартного расположения

Пример стандартного расположения для линейного (6,2) блокового кода с порождающей матрицей

$$\mathbf{G} = \begin{bmatrix} 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 1 \end{bmatrix}$$

0	0	0	0	0		1	0	1	1	1		0	1	1	0	1		1	1	0	1	0
0	0	0	0	1		1	0	1	1	0		0	1	1	0	0		1	1	0	1	1
0	0	0	1	0		1	0	1	0	1		0	1	1	1	1		1	1	0	0	0
0	0	1	0	0		1	0	0	1	1		0	1	0	0	1		1	1	1	1	0
0	1	0	0	0		1	1	1	1	1		0	0	1	0	1		1	0	0	1	0
1	0	0	0	0		0	0	1	1	1		1	1	1	0	1		0	1	0	1	0
–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
0	0	0	1	1		1	0	1	0	0		0	1	1	1	0		1	1	0	0	1
0	0	1	1	0		1	0	0	0	1		0	1	0	1	1		1	1	1	0	0

Очевидно, что для больших значений n и k составлять такую таблицу не практично. Однако, таблицу можно упростить, сохраняя только первый столбец и восстанавливая остальные столбцы по мере необходимости. Это можно сделать, вводя понятие *синдрома ошибок*

$$\mathbf{s} = \mathbf{v}\mathbf{H}^T . \qquad (2.4.4)$$

Тогда все векторы смежного класса будут иметь один синдром, присущий данному смежному классу. Действительно, если $\mathbf{v}$ и $\mathbf{v}'$ две принятые последовательности, принадлежащие одному смежному классу, то $\mathbf{v} = \mathbf{c}_i + \mathbf{y}$ и $\mathbf{v}' = \mathbf{c}_j + \mathbf{y}$ для некоторого $\mathbf{y}$ и кодовых слов $\mathbf{c}_i$ и $\mathbf{c}_j$. Синдромы для принятых векторов равны соответственно

$$\mathbf{s} = \mathbf{v}\mathbf{H}^T = \mathbf{y}\mathbf{H}^T , \qquad \mathbf{s}' = \mathbf{v}'\mathbf{H}^T = \mathbf{y}\mathbf{H}^T . \qquad (2.4.5)$$

Следовательно $\mathbf{s} = \mathbf{s}'$. Обратно, допустим, что $\mathbf{s} = \mathbf{s}'$, тогда $(\mathbf{v} - \mathbf{v}')\mathbf{H}^T = \mathbf{0}$, и поэтому разность $\mathbf{v} - \mathbf{v}'$ является кодовым словом. Следовательно, $\mathbf{v}, \mathbf{v}'$ принадлежат одному смежному классу.

Поскольку все элементы смежного класса имеют один синдром, то таблицу декодирования можно существенно сократить, оставив лишь первый столбец (соответствующей последовательности ошибок) и синдром.

Возвращаясь к примеру (6,2) блокового кода, получим следующую таблицу

e					**s**		
0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	1
0	0	0	1	0	0	1	0
0	0	1	0	0	1	0	0
0	1	0	0	0	1	0	1
1	0	0	0	0	1	1	1
–	–	–	–	–	–	–	–
0	0	0	1	1	0	1	1
0	0	1	1	0	1	1	0

Данная таблица существенно проще стандартного расположения и может быть использована для декодирования принятой последовательности. Например, предположим,

что был принят вектор (10010). Вычислим синдром для данного вектора $(10010)\mathbf{H}^T = (101)$. Данному синдрому соответствует ошибка (01000). Вычитая вычисленную ошибку из принятой последовательности, получим, что переданное слово равно $(10010)-(01000)=(11010)$. Поскольку код является систематическим, то соответствующее информационное слово равно (11).

2.5. КОДЫ ХЭММИНГА

Рассмотрим класс линейных блоковых кодов, введенных Хэммингом в 1950г. Для любого целого $m \geq 3$ существует код Хэмминга имеющий следующие параметры

$$n = 2^m - 1$$
$$k = 2^m - m - 1.$$
$$d_{\min} = 3$$

Проверочная матрица $\left(2^m - 1, 2^m - m - 1, 3\right)$ кода Хэмминга является $m \times 2^m - 1$ матрицей, столбцы которой являются всевозможные ненулевые двоичные последовательности длины m. Очевидно, что проверочная матрица $\left(2^m - 1, 2^m - m - 1, 3\right)$ кода Хэмминга может быть записана в систематическом виде

$$\mathbf{H} = \left[\mathbf{I}_m \mid \mathbf{Q}\right], \tag{2.5.1}$$

где $\mathbf{I}_m$ единичная $m \times m$ матрица, $\mathbf{Q}$ матрица размерности $m \times 2^m - m - 1$ столбцы которой представляют собой все двоичные последовательности веса два или более. Порождающая матрица для такого кода можно представить как

$$\mathbf{G} = \left[\mathbf{Q}^T \mid \mathbf{I}_{2^m - m - 1}\right]. \tag{2.5.2}$$

Рассмотрим пример (7,4) кода Хэмминга. Проверочная и порождающая матрица кода

$$\mathbf{H} = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{bmatrix}$$

$$\mathbf{G} = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Покажем, что минимальное расстояние кода Хэмминга равно 3, т.е. коды Хэмминга позволяют исправлять ошибки единичного веса. Для этого рассмотрим проверочную матрицу кода Хэмминга. Поскольку столбцы проверочной матрицы отличны друг от друга и являются ненулевыми векторами, то минимальное расстояние кода должно быть как минимум равно 3. Далее, поскольку проверочная матрица кода содержит *все* ненулевые двоичные последовательности длины m, то всегда существует столбец равный сумме двух других векторов. Следовательно, если $\mathbf{h}_i, \mathbf{h}_j$ являются столбцами матрицы $\mathbf{H}$, то существует столбец $\mathbf{h}_k$, такой что

$$\mathbf{h}_k = \mathbf{h}_i + \mathbf{h}_j \tag{2.5.3}$$

Из последнего следует, что проверочная матрица кода Хэмминга имеет три линейно зависимых столбца, а значит минимальное расстояние кода равно 3.

Коды Хэмминга являются *совершенными*, поскольку позволяют исправлять *только* ошибки веса $\lfloor (d_{\min} - 1)/2 \rfloor$ и меньше. Другими словами, кодовые сферы совершенного кода покрывают все пространство размерности n, не перекрываясь. Действительно, стандартное расположение для кодов Хэмминга имеет $2^{n-k} = 2^m$ строк, соответствующих $n = 2^m - 1$ ненулевым последовательностям единичного веса.

2.6. МОДИФИКАЦИИ ЛИНЕЙНЫХ БЛОКОВЫХ КОДОВ

Иногда параметры используемого кода (длина информационного или кодового слова, скорость кодирования) не соответствуют требованиям системы передачи информации. В этом случае существуют специальные методы модификации кодов:

- *Расширение кода*: Увеличение длины кодового слова без изменения длины информационного. Данная модификация приводит к увеличению числа проверочных символов и увеличению числа столбцов порождающей матрицы.

- *Выкалывание*: Уменьшение длины кодового слова без изменения длины информационного. Данная модификация приводит к уменьшению числа проверочных символов и уменьшению числа столбцов порождающей матрицы.
- *Пополнение кода*: Увеличение числа информационных символов без изменения длины кодового слова. Данная модификация приводит к увеличению числа строк порождающей матрицы.
- *Код с выбрасыванием*: Уменьшение числа информационных символов без изменения длины кодового слова. Данная модификация приводит к уменьшению числа строк порождающей матрицы.
- *Укорочение кода*: Уменьшение длины кодового слова путем выбрасывания некоторых информационных символов из кодового слова. Данная операция соответствует выбрасыванию некоторых строк и столбцов порождающей матрицы
- *Удлинение кода*: Увеличение длины кодового слова путем добавления информационных символов. Данная операция соответствует добавлению одинакового числа строк и столбцов к порождающей матрице.

Описанные выше модификации показаны на Рис. 9.

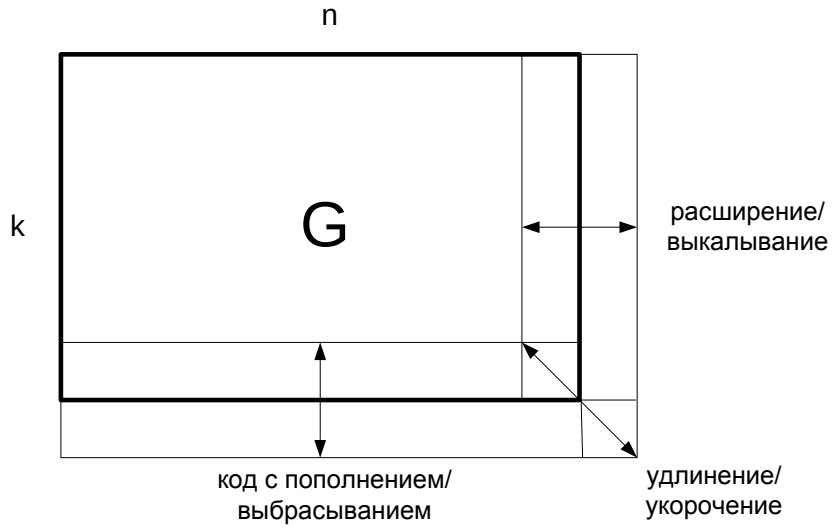


Рис. 9 Модификации линейных блоковых кодов

Глава 3

ЦИКЛИЧЕСКИЕ КОДЫ

3.1. ОПРЕДЕЛЕНИЕ ЦИКЛИЧЕСКИХ КОДОВ

При рассмотрении линейных блоковых кодов мы видели, что предположение о линейной структуре кода существенно упрощает процедуру кодирования и декодирования, а также упрощает способ описания кодов. К сожалению, процедура декодирования по таблице синдромов требует значительных вычислительных затрат и остается затруднительной для реализации в реальных высокоскоростных системах. *Алгебраическая теория кодирования* вводит дополнительные структурные ограничения линейных кодов с целью уменьшения сложности кодирования и декодирования. В частности, циклическая структура некоторых линейных кодов позволяет реализацию процедур кодирования и декодирования с помощью схем на сдвиговых регистрах. Такие коды как Хэмминга, БЧХ (Боуза-Чоудхури-Хоквингема) и Рида-Соломона имеют циклическую структуру и входят в класс циклических кодов, которые мы рассмотрим ниже.

Пусть $\mathbf{c} = (c_0, c_1, \ldots, c_{n-1})$ вектор длины n. Тогда вектор

$$\mathbf{c}^{(i)} = (c_{n-i}, c_{n-i+1}, \ldots, c_{n-1}, c_0, c_1, \ldots, c_{n-i-1}), \qquad (3.1.1)$$

является циклическим сдвигом вектора $\mathbf{c}$ на i позиций вправо. Очевидно, что сдвиг вправо вектора на i позиций эквивалентен сдвигу влево на $n-i$ позиций. Далее будем предполагать, что сдвиг осуществляется вправо, если иного не указано. Отметим также, что сдвиг на $a > n$ позиций эквивалентен сдвигу на $a \bmod n$, где $a \bmod n$ операция взятия остатка от деления a на n.

Линейный (n,k) код C называется *циклическим,* если для каждого кодового слова $\mathbf{c} \in C$ циклический сдвиг $\mathbf{c}^{(i)}$ также является кодовым словом. Для дальнейшего упрощения анализа циклических кодов двоичный вектор $\mathbf{v} = (v_0, v_1, \ldots, v_{n-1})$ будем представлять в виде полинома $v(X) = v_0 + v_1 X + v_2 X^2 + \ldots + v_{n-1} X^{n-1}$, где коэффициенты перед X^i определяются элементами последовательности $\mathbf{v}$. В этом случае существует прямое

соответствие между векторами и полиномами $n-1$ порядка. Обозначим полином ,соответствующий циклическому сдвигу вектора $\mathbf{v}=(v_0,v_1,\ldots,v_{n-1})$ на i позиций как

$$v^{(i)}(X)=v_{n-i}+v_{n-i+1}X+\ldots+v_{n-1}X^{i-1}+v_0X^{i}+v_1X^{i+1}+\ldots+v_{n-i-1}X^{n-1}. \quad (3.1.2)$$

Тогда между полиномами $v^{(i)}(X)$ и $v(X)$ существует алгебраическое отношение. Покажем, что циклический сдвиг полинома может быть получен с помощью следующей операции

$$v^{(i)}(X)=X^{i}v(X)\bmod X^{n}-1, \quad (3.1.3)$$

Для этого умножим полином $v(X)$ на X^{i} и проведем группировку слагаемых

.

$$\begin{aligned}
&X^{i}v(X)=v_0X^{i}+v_1X^{i+1}+\ldots+v_{n-i-1}X^{n-1}+\ldots+v_{n-1}X^{n+i-1}=\\
&=v_{n-i}+v_{n-i+1}X+\ldots+v_{n-1}X^{i-1}+v_0X^{i}+v_1X^{i+1}+\ldots+v_{n-i-1}X^{n-1}+\\
&+v_{n-i}\left(X^{n}-1\right)+v_{n-i+1}X\left(X^{n}-1\right)+\ldots+v_{n-1}X^{i-1}\left(X^{n}-1\right)=\\
&=w(x)\left(X^{n}-1\right)+v^{(i)}(X)
\end{aligned} \quad (3.1.4)$$

Пусть $w(x)=v_{n-i}+v_{n-i-1}X+\ldots+v_{n-1}X^{i-1}$, тогда $v^{(i)}(X)$ является остатком от деления полинома $X^{i}v(X)$ на $X^{n}-1$.

Полученное алгебраическое соотношение легко интерпретировать. Умножение полинома на X^{i} реализует процедуру сдвига элементов вектора. Для циклического сдвига необходимо идентифицировать X^{n} с единицей, т.е. произвести деление по модулю на полином, такой что $X^{n}\equiv 1$, что эквивалентно, $X^{n}-1\equiv 0$. Это можно выполнить с помощью вычисления остатка от деления на полином $X^{n}-1$. Заметим, что для двоичных векторов, коэффициенты полиномов определены над $GF(2)$, и иногда циклический сдвиг записывается через умножение по модулю $X^{n}+1$.

Рассмотрим пример сдвига полинома $v(X)=1+X^{2}+X^{3}+X^{5}+X^{6}$ на одну позицию вправо. Для этого умножим исходный полином на X и разделим его на $X^{7}-1$. Получившийся остаток $v(X)=1+X+X^{3}+X^{4}+X^{6}$ равен циклическому сдвигу $v^{(1)}(X)$ исходного полинома.

3.2. СВОЙСТВА ЦИКЛИЧЕСКИХ КОДОВ

В этом разделе рассмотрим свойства циклических кодов, позволяющих упростить реализацию процедур кодирования и декодирования. Пусть C - линейный циклический (n,k) код, заданный над полем $GF(q)$.

Свойство 1. Существует единственный полином $g(x)\in C$, $g_0\neq 0$, $g_r=1$, такой, что $g(x)$ - полином минимальной степени и $r<n$.

Доказательство: Пусть $g(x)$ является приведенным ($g_r=1$) полиномом минимальной степени r, и существует некий другой кодовый полином $g'(x)$, удовлетворяющий этим требованиям. Тогда $g(x)-g'(x)$, в силу линейности операции, также является кодовым, но имеет меньшую степень. Это противоречит нашему изначальному предположению. Следовательно, $g(x)$ должен быть единственным. Если $g_0=0$, то путем циклического сдвига на одну позицию влево можно получить кодовый полином меньшей степени.

Свойство 2. Каждый кодовый полином $c(x)\in C$ может быть представлен как $c(X)=u(X)g(X)$, где $u(x)$ - полином, заданный над $GF(q)$ и имеющий порядок меньший, чем $n-r$.

Доказательство: Докажем, что полином $v(X)$ является кодовым словом тогда и только тогда, когда делится на порождающий полином $g(x)$ без остатка. Предположим, что $v(X)$ делится без остатка на $g(x)$, тогда $v(X)=u(x)g(x)$, где $u(x)$ – полином порядка $n-r-1$ или менее. Тогда полином

$$v(X)=\left(u_0+u_1X+\ldots+u_{n-r-1}X^{n-r-1}\right)g(X)=u_0g(X)+u_1Xg(X)+\ldots+u_{n-r-1}X^{n-r-1}g(X),$$

также является кодовым словом, поскольку может быть представлен в виде линейной комбинации кодовых слов. Теперь предположим, что $v(X)$ является кодовым словом. Представим полином $v(X)$ в следующем виде

$$v(X)=u(X)g(X)+r(X), \tag{3.2.1}$$

где $r(X)$ полином меньшего порядка, чем $g(x)$ и является остатком от деления. Тогда,

$$r(X)=v(X)-u(X)g(X), \tag{3.2.2}$$

также является кодовым словом. Если полином $r(X)$ отличен от нуля, то он имеет порядок меньше $g(x)$, что противоречит определению $g(x)$ как кодового полинома наименьшей степени. Поскольку каждое кодовое слово есть произведение полинома $g(x)$ на полином $u(x)$, $g(x)$ называется *порождающим полиномом кода*.

Свойство 3. Полином $g(x)$ имеет порядок $r = n - k$.

Доказательство: Из второго свойства следует, что любое кодовое слово $v(X) \in C$ можно представить

$$v(X) = u(X)g(X) \tag{3.2.3}$$

где порядок полиномов $\deg(v(X)) \le n-1, \deg(g(X)) = r$ и $\deg(u(X)) \le n-r-1$. Таким образом, существует q^{n-r} способов выбора коэффициентов информационного полинома $u(X)$. Поскольку, (n,k) код имеет q^k кодовых слов, тогда $r = n - k$.

Свойство 4. Полином $g(x)$ является делителем $X^n - 1$.

Первое доказательство: Порождающий полином $g(X)$ имеет порядок r и $g_0 = 1$, $g_r = 1$. Тогда $X^{n-r}g(X)$ – приведенный полином степени n, которые равен

$$X^{n-r}g(X) = X^n - 1 + g^{(n-r)}(X). \tag{3.2.4}$$

Из определения циклического кода и второго свойства

$$X^{n-r}g(X) = X^n - 1 + u(X)g(X). \tag{3.2.5}$$

После перестановки, получим

$$X^n - 1 = \left(X^{n-r} + u(X)\right)g(X). \tag{3.2.6}$$

Второе доказательство: Предположим, что $g(X)$ не является кратным $X^n - 1$, тогда $X^n - 1$ можно представить в следующем виде

$$X^n - 1 = g(X)h(X) + r(X), \tag{3.2.7}$$

где остаток от деления $r(X)$ и имеет степень меньше степени полинома $g(X)$, а $h(X)$ имеет порядок $n-r$ и меньше. Следовательно

$$r(X) = X^n - 1 - g(X)h(X), \tag{3.2.8}$$

что означает $r(X) = -g(X)h(X) \bmod X^n - 1$, но $-g(X)h(X)$ и, как следствие, $r(X)$ являются кодовыми словами, что противоречит изначальному предположению о полиноме $g(X)$ как кодовом полиноме, имеющим наименьшую степень.

В этой главе мы доказали, что каждый линейный циклический (n,k) код имеет порождающий полином $g(X)$ минимальной степени $r = n - k$ и является кратным полиному $X^n - 1$. Таким образом, при разложении полинома $X^n - 1$ на множители можно получить порождающий полином степени $n - k$ линейного циклического (n,k) кода. Все кодовые слова циклического кода могут быть получены умножением информационного полинома на порождающий полином.

3.3. ПОРОЖДАЮЩАЯ МАТРИЦА ЦИКЛИЧЕСКИХ КОДОВ И ПРОЦЕДУРА КОДИРОВАНИЯ

Свойство 2 линейных циклических кодов позволяет получить относительно простую процедуру кодирования, используя порождающий полином. Каждое кодовое слово $c(X)$ может быть получено

$$c(X) = u(X)g(X), \tag{3.3.1}$$

где $u(X)$- полином порядка не выше k. Таким образом, мы можем сопоставить информационные последовательности с полиномами $u(X)$. Процедуру кодирования можно выполнить с помощью схем умножения полиномов, которые в свою очередь достаточно просто реализовать на схемах со сдвиговыми регистрами. Таким образом, порождающий полином позволяет полностью описать код. В частности, можно получить порождающую матрицу кода C. Для этого рассмотрим циклический код с полиномом $g(X)$. Процедуру кодирования можно выполнить с помощью следующей операции

$$\begin{aligned} c(X) = u(X)g(X) &= \left(u_0 + u_1 X + \ldots + u_{k-1} X^{k-1}\right) g(X) = \\ &= (u_0, u_1, \ldots, u_{k-1}) \cdot \left(g(X), Xg(X), \ldots, X^{k-1} g(X)\right) \end{aligned}. \tag{3.3.2}$$

Это приводит к следующей структуре порождающей матрицы циклического кода

$$\mathbf{v} = \mathbf{u}\begin{pmatrix} g_0 & g_1 & \cdots & g_{n-k} & & & & \\ & g_0 & g_1 & \cdots & g_{n-k} & & & \\ & & \ddots & \ddots & \ddots & \ddots & & \\ & & & g_0 & g_1 & \cdots & g_{n-k} & \\ & & & & g_0 & g_1 & \cdots & g_{n-k} \end{pmatrix} \quad (3.3.3)$$

Как уже было сказано выше, процедуру кодирования циклических кодов достаточно просто реализовать на схемах со сдвиговыми регистрами. Результат произведения $c(X) = u(X)g(X)$ является суммой циклических сдвигов полинома $g(X)$ взвешенного коэффициентами информационного полинома. Схема, реализующая умножение полиномов, показана на Рис. 10. Начальное состояние сдвиговых регистров нулевое. Элементы информационной последовательности поступают на вход схемы в порядке убывания индексов бит. После поступления последнего элемента последовательности **u** сдвиговые регистры содержат искомое кодовое слово.

Отметим, что циклические коды, полученные таким способом, не являются систематическими. В следующей главе мы рассмотрим способ получения систематических циклических кодов.

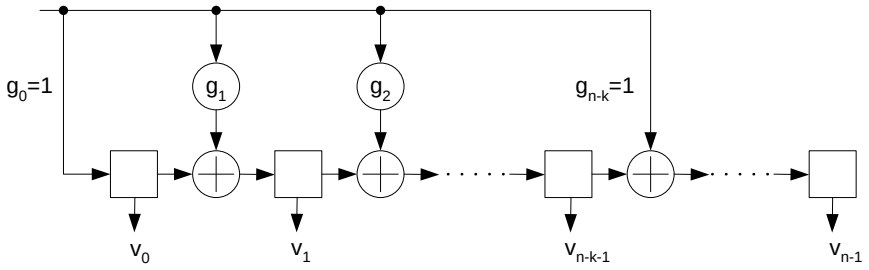


Рис. 10. Реализация процедуры кодирования на сдвиговых регистрах

В качестве примера кодирования рассмотрим (7,4) циклический код с порождающим полиномом $g(X) = 1 + X + X^3$ ($X^7 - 1 = g(X)(1 + X + X^2 + X^4)$). В Таблице 2 приведены все возможные кодовые слова данного кода. Легко видеть, что $g(X)$ является кодовым словом минимального веса. Можно также заметить, что все кодовые слова являются циклическим сдвигом другого и код не является систематическим.

Таблица 2. (7,4) циклический код $g(X)=1+X+X^3$

u	$u(X)$	$c(X)=u(X)g(X)$	**c**
0000	0	0	0000000
0001	X^3	$X^3+X^4+X^6$	0001101
0010	X^2	$X^2+X^3+X^5$	0011010
0011	X^2+X^3	$X^2+X^4+X^5+X^6$	0010111
0100	X	$X+X^2+X^4$	0110100
0101	$X+X^3$	$X+X^2+X^3+X^6$	0111001
0110	$X+X^2$	$X+X^3+X^4+X^5$	0101110
0111	$X+X^2+X^3$	$X+X^5+X^6$	0100011
1000	1	$1+X+X^3$	1101000
1001	$1+X^3$	$1+X+X^4+X^6$	1100101
1010	$1+X^2$	$1+X+X^2+X^5$	1110010
1011	$1+X^2+X^3$	$1+X+X^2+X^3+X^4+X^5+X^6$	1111111
1100	$1+X$	$1+X^2+X^3+X^4$	1011100
1101	$1+X+X^3$	$X+X^2+X^6$	1010001
1110	$1+X+X^2$	$1+X^4+X^5$	1000110
1111	$1+X+X^2+X^3$	$1+X^3+X^5+X^6$	1001011

Схема, реализующая кодирование, показана на Рис. 11. Таблица 3 содержит состояния сдвиговых регистров на каждом этапе процедуры кодирования информационной последовательности $u(X)=1+X^2$

Таблица 3. Содержимое регистров сдвига для $u(X)=1+X^2$

	s_0	s_1	s_2	s_3	s_4	s_5	s_6
Начальное состояние	0	0	0	0	0	0	0
$u_3=0$	0	0	0	0	0	0	0
$u_2=1$	1	1	0	1	0	0	0

$u_1 = 0$	0	1	1	0	1	0	0
$u_0 = 1$	1	1	1	0	0	1	0

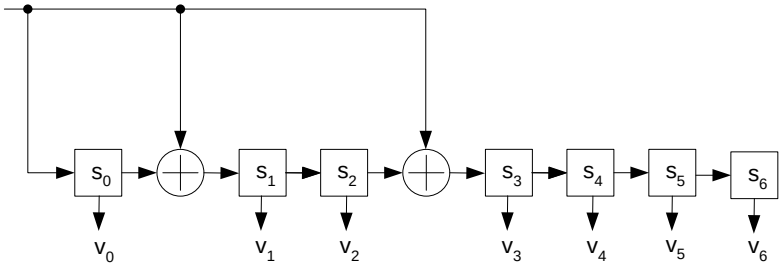


Рис. 11. Схема кодирования (7,4) циклического кода

3.4. СИСТЕМАТИЧЕСКИЕ ЦИКЛИЧЕСКИЕ КОДЫ

Для циклического (n,k) кода с порождающим полиномом $g(X)$ можно найти преобразование информационного слова, достаточного для кодирования в систематическом виде. Пусть задан информационный полином $u(X)$. Умножение данного полинома на X^{n-k} эквивалентно сдвигу вправо на $n-k$ позиций. Таким образом, мы получаем вектор, соответствующий данному полиному и содержащий $n-k$ нулей и k информационных символов: $(0,...,0,u_0,...,u_{k-1})$. Таким образом, нам необходимо найти способ заполнить ненулевые позиции проверочными символами. Если разделить полученный после сдвига полином на $g(X)$, можно получить следующее выражение

$$X^{n-k}u(X) = q(X)g(X) + r(X), \qquad (3.4.1)$$

где $q(X)$ частное, а $r(X)$ остаток от деления. Порядок полинома $r(X)$ меньше чем $n-k$. Из данного равенства можно видеть, что

$$q(X)g(X) = X^{n-k}u(X) - r(X). \qquad (3.4.2)$$

Из свойства 2 циклических кодов следует, что полином $q(X)g(X)$ является кодовым. Тогда $X^{n-k}u(X) - r(X)$ также является кодовым словом. Полином $X^{n-k}u(X)$ содержит $n-k$ нулей. Поскольку максимальный порядок $\deg(r(X)) < n-k$, то элементы векторов

соответствующих $X^{n-k}u(X)$, $r(X)$ не перекрываются. Следовательно, систематическое кодовое слово имеет следующий вид

$$(-r_0,-r_1,...,-r_{n-k-1},u_0,u_1,...,u_k).$$

Процедура систематического кодирования циклических кодов

1. Сдвинуть информационную последовательность $u(X)$ на $n-k$ позиций: $u^{(n-k)}(X)=X^{n-k}u(X)$
2. Разделить результат, полученный в шаге 1, на порождающий полином $g(X)$.
3. Выходное кодовое слово $c(X)=u^{(n-k)}(X)-r(X)$.

Схема, реализующая описанные выше операции, показана на Рис. 12. Операцию деления полинома в данном случае реализует схема на сдвиговых регистрах с обратной связью. В начале процедуры кодирования состояние регистров нулевое, а переключатели находятся в позиции А. Информационная последовательность одновременно поступает на вход и выход схемы кодирования в убывающем порядке индексов. После того как все информационные символы поступили на схему кодирования, переключатели выставляются в позицию В. В этот момент времени сдвиговые регистры содержат результат деления $u(X)$ на полином $g(X)$. Оставшиеся кодовые символы могут быть получены простым тактированием схемы.

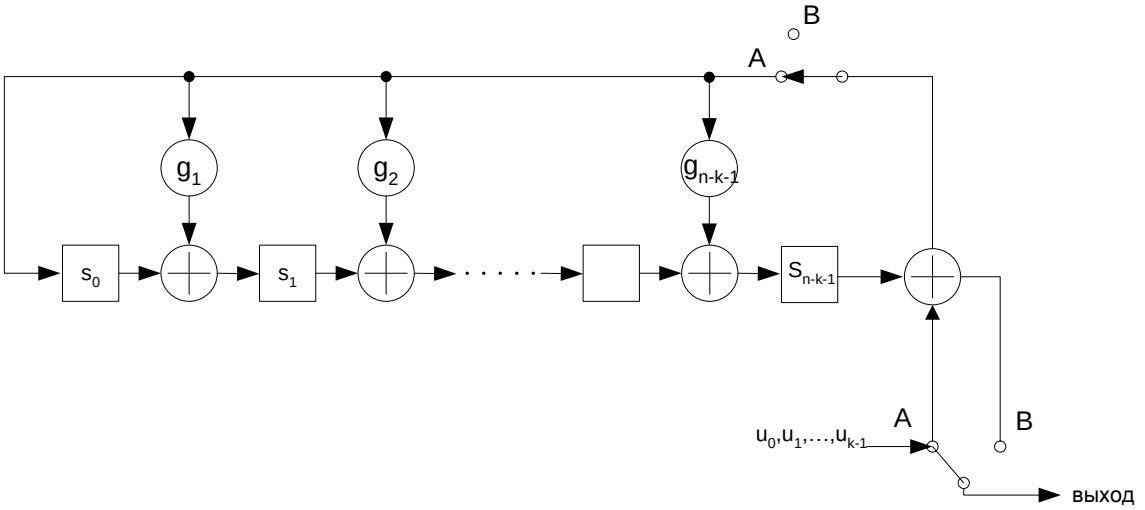


Рис. 12. Схема систематического кодирования

На Рис. 13 изображен систематический кодер для (7,4) циклического кода и порождающим полиномом $g(X)=1+X+X^3$. В таблице 4 приведены выходная последовательность и содержимое регистров сдвига на каждом этапе кодирования информационной последовательности $u(X)=1+X^2$.

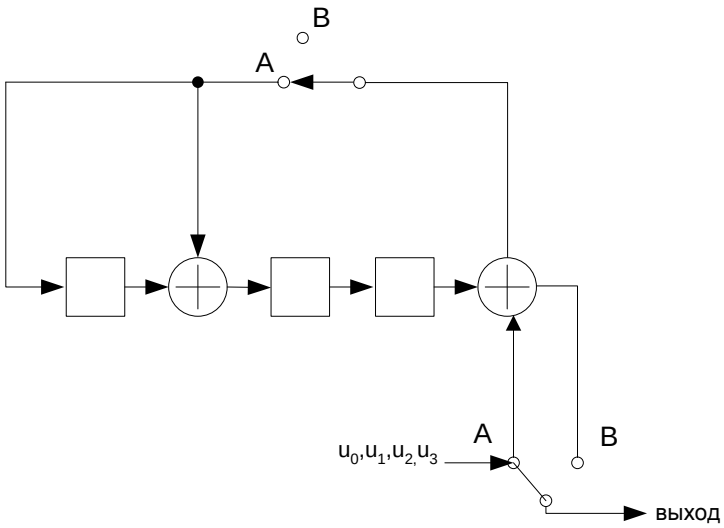


Рис. 13. Систематический кодер (7,4) циклического кода

Таблица 4. Кодирование информационного слова $\mathbf{u}=1010$

Вход	Сдвиговые регистры	Выход
0	000	0
1	110	1
0	011	0
1	001	1
-	000	1
-	000	0
-	000	-

Одним из основных применений систематических циклических кодов в современных системах связи является обнаружение ошибок. Контроль ошибок на приемнике осуществляется с помощью проверки так называемой контрольной суммы (cyclic redundancy check), которая передается вместе с информационной последовательностью. Контрольная

сумма в свою очередь вычисляется с помощью систематического циклического кода. Примеры порождающих полиномов циклических кодов, использующихся в различных стандартах, приведены в таблице 5.

Таблица 5. Примеры порождающих полиномов

Название	**Полином**	**Применение**
CRC-16-CCITT	$X^{16} + X^{12} + X^5 + 1$	Bluetooth, IEEE 802.16m
CRC-24	$X^{24} + X^{23} + X^{18} + X^{17} + X^{14} + X^{11} + X^{10} + X^7 + X^6 + X^5 + X^4 + X^3 + X + 1$	3GPP E-UTRA Rel. 8
CRC-32-IEEE 802.3	$X^{32} + X^{26} + X^{23} + X^{22} + X^{16} + X^{12} + X^{11} + X^{10} + X^8 + X^7 + X^5 + X^4 + X^2 + X + 1$	IEEE 802.11

Для последовательности с длиной, кратной некоторому значению, процедура вычисления проверочных символов может быть существенно ускорена за счет использования таблицы с заранее вычисленным остатком от деления. Рассмотрим кодовый полином $c(X) = u(X) \cdot g(X) = X^{n-k} \cdot u(X) + r(X)$. Пусть длина информационного сообщения кратна α. Расширим исходное сообщение на α бит путем добавления некоторой последовательности. Результирующий информационный полином равен $u'(X) = X^{\alpha} \cdot u(X) + b(X)$, где полином $b(X) = b_0 + b_1 X^1 + ... + b_{\alpha-1} X^{\alpha-1}$ соответствует добавленной последовательности. Проверочная последовательность $r'(X)$ может быть вычислена как остаток от деления $X^{n-k} \cdot u'(X)$ на полином $g(X)$. Подставляя $u'(X)$, получим

$$X^{n-k}\cdot u'(X)=X^{n-k}\cdot\left(X^{\alpha}\cdot u(X)+b(X)\right)=X^{n-k}\cdot b(X)+X^{\alpha}\cdot X^{n-k}\cdot u(X)$$
$$=X^{n-k}\cdot b(X)+X^{\alpha}\cdot\left(a(X)\cdot g(X)+r(X)\right) \quad (3.4.3)$$

Поскольку остаток от деления суммы полиномов равен сумме остатков от деления каждого полинома, получим

$$\left(X^{n-k}\cdot u'(X)\right)\bmod g(X)=\left(X^{n-k}\cdot b(X)+X^{\alpha}\cdot r(X)\right)\bmod g(X). \quad (3.4.4)$$

Упрощая выражение (3.4.4), получим

$$\left(X^{n-k}\cdot b(X)+X^{\alpha}\cdot r(X)\right)\bmod g(X)=$$
$$=\left(\left(b_{\alpha-1}+r_{n-k-1}\right)\cdot X^{n-k-1+\alpha}+\ldots+\left(b_0+r_{n-k-\alpha}\right)\cdot X^{n-k}\right)\bmod g(X)+ \quad (3.4.5)$$
$$+r_{n-k-\alpha-1}X^{n-k-1}+\ldots+r_0X^{\alpha}$$

Равенство задает связь проверочных бит исходного и расширенного информационного сообщения. Для быстрого вычисления первого слагаемого создается таблица остатков от деления всевозможных полиномов с коэффициентами $\{(b_{\alpha-1}+r_{n-k-1}),\ldots,(b_0+r_{n-k-\alpha})\}$ на полином $g(X)$. Данная таблица содержит 2^{α} элементов, каждый элемент которой имеет длину $n-k$ бит. Тогда алгоритм вычисления проверочной суммы может быть записан в следующем виде

1. Инициализация проверочных бит нулевой последовательностью $(r_0,r_1,\ldots,r_{n-k-1})=(0,0,\ldots,0)$
2. Суммирование входных α бит со сдвинутой на $n-k-\alpha$ позиций проверочной последовательностью $[r_{n-k-1},\ldots,r_{n-k-\alpha}]$. Вычисление остатка от деления на $g(X)$ с помощью таблицы
3. Сдвиг регистров, содержащих проверку на четность на α позиций
4. Суммирования результатов шага (2) и (3)

Шаги 2-4 повторяются, пока конец последовательности не достигнут.

3.5. ВЫЧИСЛЕНИЕ СИНДРОМА И ДЕКОДИРОВАНИЕ ЦИКЛИЧЕСКИХ КОДОВ

Согласно свойству 4 циклических кодов, порождающий полином $g(X)$ является делителем X^n-1. Как следствие, должен существовать полином $h(X)$ такой что

$X^n - 1 = g(X)h(X)$. Полином $h(X)$ называют проверочным полиномом, поскольку для любого кодового слова $v(X) \in C$ выполняется равенство

$$v(X)h(X) = u(X)g(X)h(X) = u(X)(X^n - 1) = 0 \bmod X^n - 1. \quad (3.5.1)$$

Таким образом, полином является кодовым тогда и только тогда когда $v(X)h(X) = 0 \bmod X^n - 1$. Следовательно, для вычисления синдрома принятой последовательности используется умножение принятой последовательности на проверочный полином $h(X)$.

Пусть принятый вектор представлен в следующем виде $r(X) = u(X)g(X) + e(X)$, тогда

$$h(X)r(X) = h(X)u(X)g(X) + h(X)e(X) = h(X)e(X) \bmod X^n - 1. \quad (3.5.2)$$

Из последнего выражения можно видеть, что полином $h(X)e(X)$ равен нулю, когда $e(X)$ является кодовым словом. Таким образом, $s(X) = h(X)r(X) \bmod X^n - 1$ будет называть синдромом $r(x)$. Поскольку $X^n - 1 = g(X)h(X)$, умножение на $h(X)$ эквивалентно делению на $g(X)$.

Деление на полином $g(X)$ может быть реализовано с помощью схемы представленной на Рис. 14.

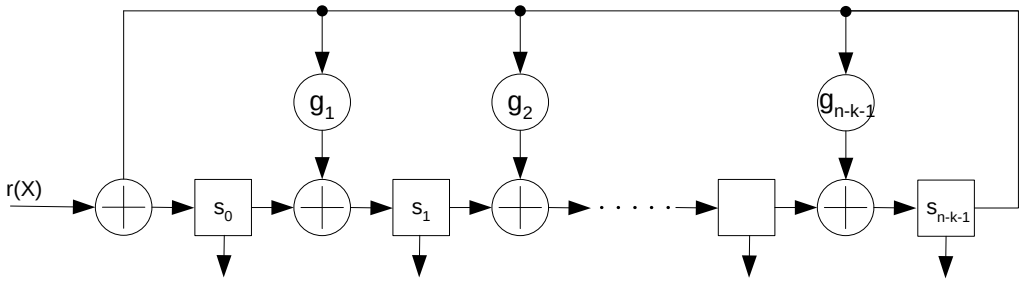


Рис. 14. Схема вычисления синдромов

При рассмотрении линейных блоковых кодов мы видели, что декодирование по максимуму правдоподобия, с использованием таблицы синдромов, требует список, который возрастает с увеличением длины кодового слова. Синдромы циклических кодов имеет определенную структуру позволяющую упростить схемы декодирования. Например, если известен синдром принятого вектора, то можно сравнительно просто вычислить синдром

циклического сдвига данного вектора. Пусть $s(X)$ - синдром принятого вектора $r(X)$. Тогда синдром $s^{(1)}(X)$ вектора $r^{(1)}(X)$ является остатком от деления $Xs(X)$ на $g(X)$, т.е.

$$s^{(1)}(X) = Xs(X) \bmod g(X). \tag{3.5.3}$$

Запишем $X \cdot r(X) = r_{n-1}\left(X^n - 1\right) + r^{(1)}(X)$. Переставляя слагаемые в равенстве и используя $X^n - 1 = g(X)h(X)$, получим

$$r^{(1)}(X) = X\, r(X) - r_{n-1}g(X)h(X). \tag{3.5.4}$$

Далее представим обе части выражения как частное и остаток от деления на $g(X)$

$$c(X)g(X) + d(X) = X(a(X)g(X) + s(X)) - r_{n-1}g(X)h(X), \tag{3.5.5}$$

где $d(X)$ синдром $r^{(1)}(X)$. После перестановки выражение принимает вид

$$\begin{aligned} Xs(X) &= r_{n-1}g(X)h(X) + c(X)g(X) + d(X) - Xa(X)g(X) \\ &= g(X)(r_{n-1}h(X) + c(X) - Xa(X)) + d(X) \end{aligned}. \tag{3.5.6}$$

Таким образом, $d(X)$ является остатком от деления $Xs(X)$ на полином $g(X)$.

Равенство (3.5.3) позволяет упростить процедуру вычисления синдромов циклического сдвигов принятого вектора. Мы уже видели, что вычисление синдрома $r(X)$ может быть выполнено делением на $g(X)$. После вычисления синдрома принятого вектора можно вычислить синдромы полиномов $r^{(1)}(X)$ и т.д., поскольку последующие вычислительные операции данной схемы эквивалентны умножению $s(X)$ на X и делению на $g(X)$.

На Рис. 15 изображена схема вычисления синдрома для (7,4) кода с порождающим полиномом $g(X) = 1 + X + X^3$. В таблице 6 приведено содержимое регистров сдвига после каждого шага работы схемы. На вход поступает последовательность $r = 0010110$. После того как последний бит поступил на вход схемы, последующие шаги работы схемы вычисляют синдромы сдвигов принятого вектора.

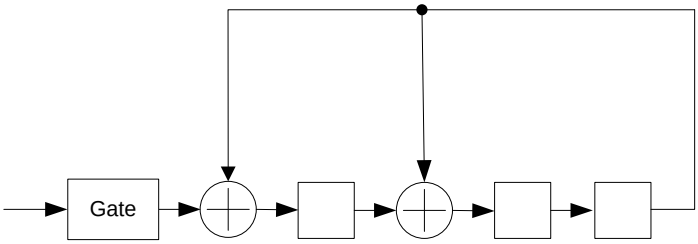


Рис. 15. Пример схемы вычисления синдрома

Таблица 6. Вычисление синдрома (7,4) кода

Вход	Сдвиговые регистры	
0	000	
1	100	
1	110	
0	011	
1	011	
0	111	
0	101	s
-	100	$s^{(1)}$
-	010	$s^{(2)}$

Декодер Меггита, показанный на Рис. 16, использует преимущество теоремы 3 при исправлении ошибок в принятом векторе. Процедура декодирования состоит из трех этапов. На первом шаге, схема вычисления синдромов используется для вычисления $s(X)$. Входная последовательность сохраняется для последующего вычисления с помощью той же схемы полиномов $s^1(X)$, $s^2(X)$ и т.д. Для сохранения входной последовательности используется буфер, который циклически сдвигает принятый вектор синхронно со схемой вычисления синдрома, которая в свою очередь содержит на каждом этапе вычисления соответствующий синдром. Последним этапом вычисления является детектирование ошибок. Это схема комбинационной логики, которая используется для определения из вычисленного синдрома наличия ошибки в правой позиции вектора. Выход данной схемы используется для исправления принятых символов и для обновления содержимого регистров сдвига в схеме вычисления синдромов.

Принцип работы декодера Меггита

1. Подаем принятую последовательность символов на вход для вычисления синдрома $s(X)$ (входы B, C отключены). Принятую последовательность сохраняем в буфере.
2. Подключаем вход C (вход A отключен). Вычисленный синдром поступает на схему детектирования ошибок. Если вычисленный синдром соответствует ошибке, на выходе данной схемы имеем $e_i = 1$ в противном случае $e_i = 0$.
3. Подключаем вход B. Регистры буфера и регистры синдрома синхронно тактируются.
    a. Бит высшего разряда r_{n-1} исправляется и подается на выход. Выходной буфер содержит последовательность $r_{n-1} + e_i, r_0, ..., r_{n-2}$.
    b. Выходной сигнал схемы детектирования ошибок подается также на схему вычисления синдромов. Без данной процедуры регистры схемы вычисления синдрома будут содержать $s^{(1)}(X)$ – синдром циклического сдвига неисправленной последовательности. Нам необходимо вычесть из данного синдрома ошибочную последовательность $e(X) = 1$ (поскольку мы сдвинули принятую последовательность на одну позицию вправо). Синдром $e(X) = 1$ равен $1 \bmod g(X) = 1$. Таким образом, достаточно просто вычесть e_i из младшего бита.
4. Повторить этап 3 необходимое число раз

В качестве примера рассмотрим (7,4) циклический код с порождающим полиномом $g(X) = 1 + X + X^3$. Данный код позволяет исправить ошибку единичного веса и имеет $d_{\min} = 3$. Таким образом, существует единственный исправляемый ошибочный полином с единицей в старшей позиции, т.е. $e(X) = X^6$. Синдром данного полинома $1 + X^2$. На Рис. 17 показан декодер Меггита данного кода.

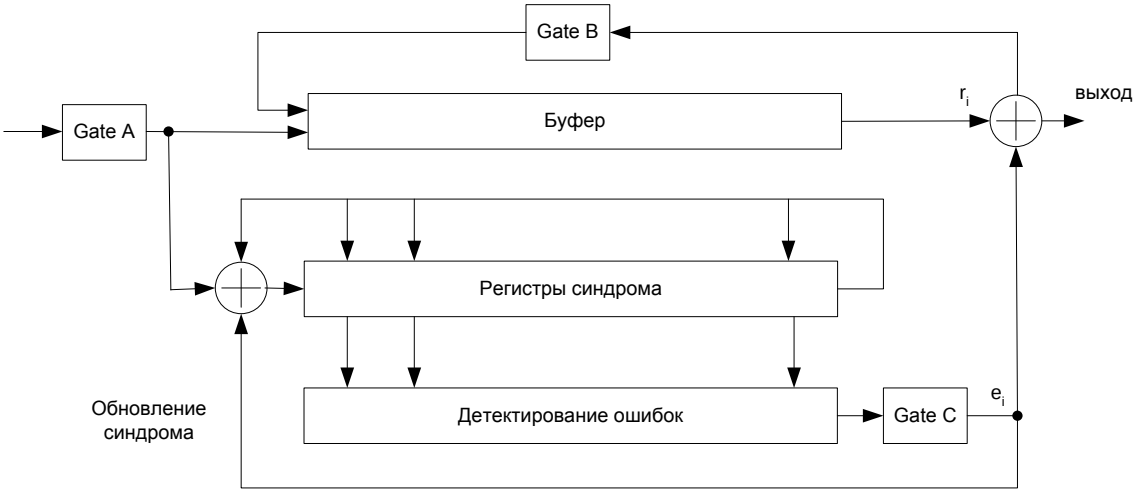


Рис. 16. Декодер Меггита

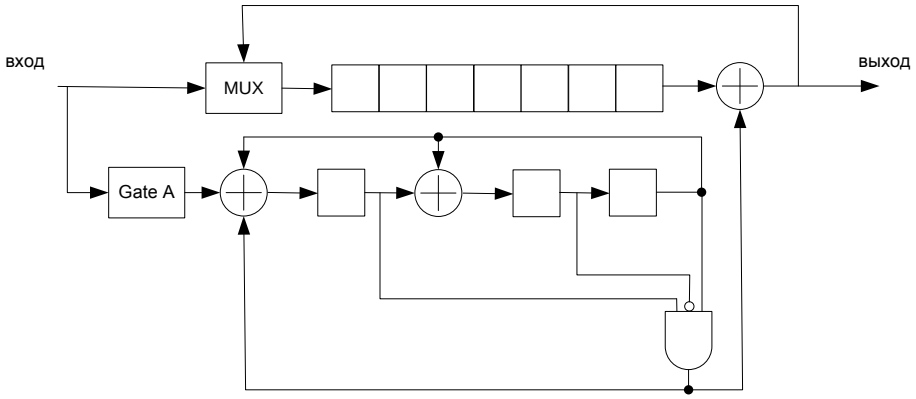


Рис. 17. Декодер Меггита (7,4) циклического кода

3.6. ПРИМЕРЫ ЦИКЛИЧЕСКИХ КОДОВ

Как уже обсуждалось ранее, порождающие полиномы циклических кодов могут быть получены путем разложения полинома $X^n - 1$ на множители. Были найдены достаточно интересные циклические коды, получившие широкое распространение в системах связи и хранения информации: коды Хэмминга, Голея, и БЧХ. Например, (7,4) циклический код с порождающим полиномом $g(X) = 1 + X + X^3$ является кодом Хэмминга. Коды Хэмминга большей длины могут быть также описаны как циклические коды. Другим примером циклического кода является код Голея. Данный код является совершенным (23,12,7) кодом с

исправлением тройных ошибок. Порождающий полином кода Голея задается $g(X)=1+X^2+X^4+X^5+X^6+X^{10}+X^{11}$.

Глава 4

КОДЫ РИДА-СОЛОМОНА

4.1. ОСНОВНЫЕ ПОНЯТИЯ ТЕОРИИ КОНЕЧНЫХ ПОЛЕЙ

Полем называют множество элементов, на котором определены две операции. Одна из них называется сложением и обозначается $x+y$, а другая – умножением и обозначается $x\cdot y$, даже если эти операции не являются обычными операциями сложения и умножения чисел. Для того чтобы множество элементов, на котором заданы операции сложения и умножения, являлось полем, необходимо, чтобы по каждой из этих операций выполнялись все аксиомы групп. Также необходимо выполнение дистрибутивного закона, т.е. для трех любых элементов поля x, y, z равенство $x\cdot(y+z)=x\cdot y+x\cdot z$ должно быть справедливым. Поля с конечным числом элементов q называют полями Галуа по имени их первого исследователя Эвариста Галуа и обозначают $GF(q)$. В зависимости от значения q различают простые или расширенные поля. Поле называют простым, если $q=p$ – простое число. Простое поле образуют числа $\{0,1,...,p-1\}$, а операции сложения и умножения выполняются по модулю p. Наименьшее число элементов, образующих поле, равно 2. Такое поле должно содержать два единичных элемента: 0 относительно операции сложения и 1 относительно операции умножения. Такое поле называется двоичным и обозначается как $GF(2)$. Правила сложения и умножений для элементов определяются как

+	0	1
0	0	1
1	1	0

·	0	1
0	0	0
1	0	1

.

Элементы 0 и 1 как единичные элементы по операции сложения и умножения соответственно не изменяют значения других элементов поля по соответствующей операции. Кроме того, для каждого элемента по операции сложения и для ненулевых элементов по операции умножения имеются обратные. К сожалению, набор целых чисел $\{0,1,...,q-1\}$, где q не является простым числом, и элементарные операции сложения и умножения по модулю

q не образуют конечного поля. Так, например, для $q=16$, среди элементов $\{0,1,...,15\}$ не существует элемента обратного к элементу 2.

Рассмотрим возможность построения поля с элементами в виде последовательностей чисел. Определим условия, при которых последовательности длины m с элементами из поля $GF(p)$, где p - простое число, образуют поле. Рассмотрим последовательности длины 4 с элементами из $GF(2)$. Такие последовательности можно складывать как векторы, а нулевым элементом по операции сложения является 0000. Для задания операции умножения сопоставим каждой последовательности полином от α

Последовательность	Полином
0 0 0 0	0
1 0 0 0	1
0 1 0 0	α
1 1 0 0	$1+\alpha$
0 0 1 0	α^2
1 0 1 0	$1+\alpha^2$
0 0 0 1	α^3
…	…
1 1 1 1	$1+\alpha+\alpha^2+\alpha^3$

Умножение полиномов может дать степень большую, чем 3, т.е. последовательность, не принадлежащую рассматриваемому множеству. Например, полином $\left(1+\alpha+\alpha^3\right)\left(1+\alpha^3\right)=1+\alpha+\alpha^4+\alpha^6$. Для того чтобы свести ответ к полиному степени не более 3, положим, что α удовлетворяет уравнению степени 4, например, $p(\alpha)=1+\alpha+\alpha^4$ или $\alpha^4=1+\alpha$. Тогда $\alpha^5=\alpha+\alpha^2$, $\alpha^6=\alpha^2+\alpha^3$ и $1+\alpha+\alpha^4+\alpha^6=\alpha^2+\alpha^3$, что эквивалентно делению на полином $1+\alpha+\alpha^4$ и нахождению остатка от деления. Таким образом, имеет место аналогия при формировании поля из чисел и последовательностей чисел (полиномов). Эта аналогия распространяется и на то, что для обратимости введенной операции умножения (чтобы система элементов в виде последовательностей длины m или полиномов степени меньшей m, образовывала поле) полином $p(\alpha)$ должен быть неприводим над полем своих

коэффициентов. Поле, образованное полиномами над полем $GF(p)$ (где p простое) по модулю неприводимого полинома $p(x)$ степени m, называется расширением поля степени m над $GF(p)$ или расширенным полем. Оно содержит p^m элементов и обозначается $GF\left(p^m\right)$. Поле, образованное шестнадцатью двоичными последовательностями длины 4, или полиномами степени 3 и менее с коэффициентами из $GF(2)$ по модулю полинома $1+x+x^4$, неприводимого над $GF(2)$, является примером расширенного поля $GF\left(2^4\right)$.

Важнейшим свойством конечных полей является следующее. Множество всех ненулевых элементов конечного поля образует группу по операции умножения, т.е. мультипликативную группу порядка $q-1$. Рассмотрим совокупность элементов мультипликативной группы, образованную некоторым элементом α и всеми его степенями α^2, α^3 и т.д. Так как группа конечна, должно появиться повторение, т.е. $\alpha^i = \alpha^j$. Умножая это равенство на $\left(\alpha^i\right)^{-1} = \left(\alpha^{-1}\right)^i$, получим $1 = \alpha^{j-i}$. Следовательно, некоторая степень элемента α равна 1. Наименьшее положительное число e, такое, что $\alpha^e = 1$, называется порядком элемента α. Совокупность элементов $1, \alpha, \alpha^2, ..., \alpha^{e-1}$ образует подгруппу, поскольку произведение любых двух элементов принадлежит этой совокупности, а элемент, обратный α^j, равен α^{e-j} и тоже входит в эту совокупность. Группа, которая состоит из всех степеней одного из ее элементов, называется *циклической* группой.

Из рассмотренного свойства конечных полей вытекают два важных следствия. Первое из них утверждает, что полином $x^{q-1}-1$ имеет своими корнями все $q-1$ ненулевых элементов поля $GF(q)$, т. е.

$$x^{q-1}-1 = \prod_{\substack{\alpha \in GF(q) \\ \alpha \neq 0}} (x-\alpha) \qquad (4.1.1)$$

В поле $GF(q)$ элемент α, имеющий порядок $e = q-1$, называется *примитивным*. Отсюда следует, что любой ненулевой элемент $GF(q)$ является степенью примитивного элемента. Второе следствие из рассмотренного свойства утверждает, что любое конечное поле $GF(q)$ содержит примитивный элемент, т.е. мультипликативная группа $GF(q)$ является циклической.

В таблице 7 представлены различными способами элементы $GF\left(2^4\right)$.

Таблица 7. Пример поля $GF\left(2^4\right)$

Последовательность	Полином	Степень	Логарифм
0000	0	0	$-\infty$
1000	1	1	0
0100	α^1	α^1	1
0010	α^2	α^2	2
0001	α^3	α^3	3
1100	$1+\alpha$	α^4	4
0110	$\alpha+\alpha^2$	α^5	5
0011	$\alpha^2+\alpha^3$	α^6	6
1101	$1+\alpha+\alpha^3$	α^7	7
1010	$1+\alpha^2$	α^8	8
0101	$\alpha+\alpha^3$	α^9	9
1110	$1+\alpha+\alpha^2$	α^{10}	10
0111	$\alpha+\alpha^2+\alpha^3$	α^{11}	11
1111	$1+\alpha+\alpha^2+\alpha^3$	α^{12}	12
1011	$1+\alpha^2+\alpha^3$	α^{13}	13
1001	$1+\alpha^3$	α^{14}	14

Поле $GF\left(2^4\right)$, представленное в таблице 7, построено по модулю x^4+x+1. Примитивный элемент поля α является корнем этого полинома. Полином, корнем которого является примитивный элемент, называется *примитивным полиномом*. Если в качестве $p(x)$ выбрать примитивный неприводимый полином степени m над полем $GF(2)$, то получим поле $GF\left(2^m\right)$ из всех 2^m двоичных последовательностей длины m.

4.2. КОДЫ РИДА-СОЛОМОНА

Коды Рида–Соломона относятся к недвоичным циклическим кодам, т.е. кодам, символы которых взяты из конечного поля, содержащего $q > 2$ элементов и обозначаемого $GF(q)$, где q – степень некоторого простого числа. Пусть необходимо передать по каналу связи последовательность из M двоичных элементов. Разобьем эту последовательность на блоки по m элементов и обозначим их через некоторые символы $b_0, b_1, ..., b_{N-1}$, где $N = M/m$. Полное число различных значений m-элементных блоков равно $q = 2^m$. Таким образом, передаваемая последовательность представляется в виде некоторой q-ичной последовательности: $b_0, b_1, ..., b_{N-1}$. Некоторая совокупность q-ичных последовательностей образует q-ичный код. Такие коды, как и двоичные коды, могут быть помехоустойчивыми.

Кодовые комбинации q-ичного кода могут быть представлены в виде полиномов с q-ичными коэффициентами – элементами поля $GF(q)$. При этом q-ичные коэффициенты как элементы поля $GF(q)$ являются в рассмотренном примере полиномами с двоичными коэффициентами. Например:

$$B(X) = b_0(z)X^0 + b_1(z)X^1 + ... + b_{n-1}(z)X^{n-1}, \qquad (4.2.1)$$

где $b_i(z) = b_0 z^0 + b_1 z^1 + ... + b_{m-1} z^{m-1}$. Здесь $b_i \in \{0,1\}$, а z – формальная переменная полинома с двоичными коэффициентами.

Кодом Рида-Соломона называют циклический (n,k) код, при $n = q-1$, множество кодовых комбинаций которого представляется полиномами степени $n-1$ и менее с коэффициентами из поля $GF(q)$, где $q > 2$ и является степенью простого числа, а корнями порождающего полинома являются $n-k$ последовательных степеней $\alpha^0, \alpha^1, ..., \alpha^{d-1}$, некоторого элемента $\alpha \in GF(q)$, где d – минимальное кодовое расстояние (n,k)-кода. Обычно считают элемент α примитивным элементом поля $GF(q)$, т.е. все степени α от 1-й до $q-1$ являются всеми различными ненулевыми элементами поля $GF(q)$. Порождающий полином кода имеет степень $n-k = d-1$ и по теореме Безу может быть найден в виде произведения

$$g(x)=\prod_{i=1}^{d-1}\left(x-\alpha^{i}\right). \qquad (4.2.2)$$

В соответствии с теорией циклических кодов, порождающий полином $g(x)$ является делителем $x^{n}-1$ над $GF(q)$. Таким образом, код Рида-Соломона над полем $GF(q)$ имеет длину кодовой комбинации $n=q-1$, число избыточных элементов в ней $n-k=d-1$ и минимальное кодовое расстояние $d=n-k+1$.

При фиксированных n и k не существует кода, у которого минимальное кодовое расстояние больше, чем у кода Рида-Соломона. Этот факт часто является веским основанием для использования кодов. В то же время коды Рида-Соломона всегда оказываются короче всех других циклических кодов над тем же алфавитом.

Рассмотрим примеры кодов Рида-Соломона над расширенным полем $GF\left(2^{2}\right)$. Определяем длину кодовой комбинации: $n=q-1=3$. Зададимся кодовым расстоянием $d=2$. Для его реализации необходима избыточность $n-k=d-1=1$. Если необходимо обеспечить $d=3$, то следует задать избыточность $n-k=2$.

Таким образом, над $GF\left(2^{2}\right)$ можно построить (3,2) код с $d=2$ и (3,1) код с $d=3$. Для (3,2) кода Рида-Соломона порождающий полином на основании определения

$$g(x)=x-\alpha. \qquad (4.2.3)$$

$GF\left(2^{2}\right)$ является расширением поля $GF(2)$, поэтому знак "–" в $g(x)$ следует заменить на "+", как символ операции сложения в $GF(2)$, т.е. следует принять $g(x)=x+\alpha$. Порождающая матрица этого кода имеет вид

$$G=\begin{pmatrix}\alpha & 1 & 0\\ 0 & \alpha & 1\end{pmatrix}.$$

Код (3,2) над $GF\left(2^{2}\right)$ содержит $q^{k}=4^{2}=16$ разрешенных комбинаций. Они имеют вид

0	$0 \quad 0 \quad 0$
1	$\alpha \quad 1 \quad 0$
2	$\alpha^{2} \quad \alpha \quad 0$
3	$1 \quad \alpha^{2} \quad 0$

4	$0 \quad \alpha \quad 1$
5	$\alpha \quad \alpha^2 \quad 1$
6	$\alpha^2 \quad 0 \quad 1$
7	$1 \quad 1 \quad 1$
8	$0 \quad \alpha^2 \quad \alpha$
9	$\alpha \quad \alpha \quad \alpha$
10	$\alpha^2 \quad 1 \quad \alpha$
11	$1 \quad 0 \quad \alpha$
12	$0 \quad 1 \quad \alpha^2$
13	$\alpha \quad 0 \quad \alpha^2$
14	$\alpha^2 \quad \alpha^2 \quad \alpha^2$
15	$1 \quad \alpha \quad \alpha^2$

Каждая комбинация представляет собой полином степени 2 или менее. Выше указаны последовательности коэффициентов каждой из кодовых комбинаций в предположении, что коэффициенты старших степеней находятся справа, т.е. информационные элементы занимают две позиции справа, а избыточный элемент – крайнюю слева.

Проследим формирование избыточных элементов для комбинаций 1.

$$\begin{array}{cc} x & \underline{x+\alpha} \\ \underline{x+\alpha} & 1 \\ \alpha & \end{array}$$

При выполнении действий над элементами поля $GF\left(2^2\right)$ полезно помнить, что оно построено по модулю неприводимого примитивного полинома $p(\alpha)=1+\alpha+\alpha^2=0$.

Введение в кодовой комбинации дополнительной проверки на четность во многих случаях увеличивает минимальное расстояние кода на одну единицу. Для кодов Рида-Соломона это выполняется всегда. Пусть $\mathbf{c}=(c_0,c_1,...,c_{n-1})$ кодовая последовательность кода Рида-Соломона. Введем дополнительную проверку на четность по правилу

$$c_n = -\sum_{i=0}^{n-1} c_i \,. \qquad (4.2.4)$$

Покажем, что минимальный вес кодовой комбинации $\mathbf{c} = (c_0, c_1, ..., c_{n-1})$ минимального веса d при расширении увеличивается до $d+1$. Это возможно при условии, что

$$c(1) = -c_n = \sum_{i=0}^{n-1} c_i \neq 0, \qquad (4.2.5)$$

но $c(X) = u(X)g(X)$, так что $c(1) = u(1)g(1)$. Очевидно $g(1) \neq 0$. Кроме того $u(1) \neq 0$, иначе $c(X)$ делилось бы на $(X-1)$, т.е. уже имело бы вес $d+1$.

Например, расширение (3,2) кода Рида-Соломона с порождающим полиномом $g(x) = x + \alpha$ и $d = 2$ является (4,2) код с $d = 3$. Порождающий полином которого равен $g(x) = (x+\alpha)(x+1) = x^2 + \alpha^2 x + \alpha$, а порождающая матрица имеет вид

$$G = \begin{pmatrix} \alpha & \alpha^2 & 1 & 0 \\ 0 & \alpha & \alpha^2 & 1 \end{pmatrix}$$

4.3. ДЕКОДИРОВАНИЕ КОДОВ РИДА-СОЛОМОНА

Способы кодирования и декодирования кодов Рида-Соломона достаточно хорошо разработаны в теоретическом и реализационном плане. Их основу составляют процедуры исправления стираний, ошибок и совместного исправления ошибок и стираний. Рассмотрим декодирование с исправлением ошибок, известное как алгоритм Форни. Пусть на приемник поступила кодовая последовательность

$$r(X) = c(X) + e(X) \qquad (4.3.1)$$

где $c(X)$ переданная кодовая последовательность, в которой в процессе передачи по каналу связи произошло ν ошибок, отображаемых полиномом $e(X)$

$$e(X)=e_{i_1}X^{i_1}+e_{i_2}X^{i_2}+\ldots+e_{i_\upsilon}X^{i_\upsilon}\,. \qquad (4.3.2)$$

Каждый ненулевой компонент $e(X)$ описывается парой элементов из поля $GF(q)$: $Y_l=e_{i_l}$ – величина ошибок и $X_l=\alpha^{i_l}$ – номер позиции ошибки (локатор ошибки). Для получения синдромов S_j надо найти значения полученного полинома в точках α^j, $j=1,2,\ldots,2t$

$$S_j=r(\alpha^j)=c(\alpha^j)+e(\alpha^j). \qquad (4.3.3)$$

Тогда получим следующую систему из $2t$ уравнений относительно ν неизвестных локаторов ошибок $X_1,X_2,\ldots,X_\nu$ и ν неизвестных величин ошибок $Y_1,Y_2,\ldots,Y_\nu$

$$\begin{aligned}
&S_1=Y_1X_1+Y_2X_2+\ldots+Y_\nu X_\nu\\
&S_2=Y_1X_1^2+Y_2X_2^2+\ldots+Y_\nu X_\nu^2\\
&\vdots\\
&S_{2t}=Y_1X_1^{2t}+Y_2X_2^{2t}+\ldots+Y_\nu X_\nu^{2t}
\end{aligned}\,. \qquad (4.3.4)$$

В силу определения синдрома эта система уравнений должна иметь хотя бы одно решение. Задача декодирования в этом случае состоит в вычислении неизвестных по заданным компонентам синдрома, т.е. в решении системы нелинейных уравнений. Далее рассмотрим два подхода к декодированию кодов Рида Соломона.

Декодер Питерсона – Горенстейна – Цирлера

Рассмотрим полином

$$\Lambda(X)=\prod_{i=1}^{\nu}(1-XX_i)=\Lambda_\nu X^\nu+\Lambda_{\nu-1}X^{\nu-1}+\ldots+\Lambda_1X^1+1, \qquad (4.3.5)$$

известный под названием полинома локаторов ошибок и определяемый как полином, корнями которого являются обратные к локаторам ошибок величины X_l^{-1}. Если коэффициенты этого полинома известны, то для вычисления локаторов ошибок нужно найти его корни. Поэтому попытаемся сначала вычислить по заданным компонентам синдрома коэффициенты $\Lambda_\nu,\Lambda_{\nu-1},\ldots,\Lambda_1$.

Умножим обе части равенства, определяющего этот полином, на $Y_lX_l^{j+\nu}$ и положим $X=X_l^{-1}$. Тогда получим равенство

$$Y_l\left(X_l^{j+\nu}+\Lambda_1 X_l^{j+\nu-1}+\ldots+\Lambda_\nu X_l^{j}\right)=0, \tag{4.3.6}$$

справедливое при каждом l и j. Проведем суммирование равенств по l от 1 до ν. Для каждого j это дает следующее уравнение

$$\sum_{l=1}^{\nu} Y_l X_l^{j+\nu}+\Lambda_1\sum_{l=1}^{\nu} Y_l X_l^{j+\nu-1}+\ldots.+\Lambda_\nu\sum_{l=1}^{\nu} Y_l X_l^{j}=0 \tag{4.3.7}$$

Каждая сумма в левой части равенства является компонентой синдрома, так что уравнение приводится к виду

$$S_{j+\nu}+\Lambda_1 S_{j+\nu-1}+\ldots.+\Lambda_\nu S_j=0 \tag{4.3.8}$$

Пробегая по всем значениям j в диапазоне $1\le j\le \nu$, получим систему линейных уравнений, которую можно записать в матричном виде

$$\begin{bmatrix} S_1 & S_2 & \cdots & S_{\nu-1} & S_\nu \\ S_2 & S_3 & \cdots & S_\nu & S_{\nu+1} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ S_\nu & S_{\nu+1} & \cdots & S_{2\nu-2} & S_{2\nu-1} \end{bmatrix}\begin{bmatrix} \Lambda_\nu \\ \Lambda_{\nu-1} \\ \vdots \\ \Lambda_1 \end{bmatrix}=\begin{bmatrix} -S_{\nu+1} \\ -S_{\nu+2} \\ \\ -S_{2\nu} \end{bmatrix} \tag{4.3.9}$$

Можно показать, что эта система является невырожденной, если произошло ν ошибок. Таким образом, для поиска коэффициентов полинома локатора ошибок, прежде всего, необходимо найти правильное значение ν. В качестве пробного значения выбирается $\nu=t$. Если система уравнений не вырождена, то мы получили правильное значение для ν; если же вырождена, то уменьшим значение ν на единицу и повторим процедуру до тех пор пока не найдем правильного значения для числа ошибок. Далее решая уравнение (4.3.9) получим коэффициенты полинома $\Lambda(X)$. Локаторы ошибок могут быть найдены из корней полинома локаторов ошибок.

Для определения значений ошибок Y_l, вернемся к уравнениям, определяющим компоненты синдрома. Аналогично уравнениям для коэффициентов полинома $\Lambda(X)$ найдем значения ошибок путем решения ν линейных уравнений.

Описанный декодер Питерсона–Горенстейна–Цирлера предполагает обращение двух матриц размерности $t\times t$. Хотя обращение матриц в конечных полях не приводит к ошибкам округления, вычислительные затраты, особенно для больших значений t, остаются достаточно высокими. В следующих параграфах мы рассмотрим быстрые алгоритмы декодирования, позволяющие избежать обращения матриц. Обращение первой матрицы, используемое для вычисления локаторов ошибок, можно обойти, используя алгоритм

Берлекэмпа-Месси. Обращение второй матрицы, необходимое для вычисления значений ошибок, можно обойти, воспользовавшись процедурой Форни.

Алгоритм Форни

Введем полином значений ошибок

$$\Omega(X) = S(X)\Lambda(X) \bmod X^{2t}\ , \qquad (4.3.10)$$

где $S(X) = \sum_{j=1}^{2t} S_j X^j = \sum_{j=1}^{2t}\sum_{i=1}^{\nu} Y_i X_i^j X^j$ – синдромный полином, для которого известны только $2t$ коэффициентов для поступившей комбинации кода. Здесь $S_j = \sum_{i=1}^{\nu} Y_i X_i^j = e\left(\alpha^j\right)$ – элемент синдрома, α^j – корень порождающего полинома кода.

Равенство (4.3.11) определяет множество из $2t-\nu$ уравнений и называется ключевым уравнением. Это становится понятным, если учесть, что степень $\Omega(X)$ не превышает $\nu-1$ и поэтому справедливо:

$$[\Lambda(X)\cdot S(X)]_{\nu}^{2t-1} = 0 \qquad (4.3.11)$$

где $[A(X)]_m^n = a_m X^m + a_{m+1}X^{m+1} + ... + a_n X^n$. Из равенства необходимо получить ν уравнений для ν неизвестных коэффициентов Λ_k. Эти уравнения являются линейными. Они могут быть решены обычными методами либо с помощью итерационных процедур. После нахождения полинома $\Lambda(X)$ ключевое уравнение позволяет найти неизвестные компоненты вектора $e(X)$ и по ним выходной вектор декодера. Простейшим путем нахождения корней полинома $\Lambda(X)$ является метод проб и ошибок, известный как процедура Ченя. Эта процедура состоит в последовательном вычислении $\Lambda\left(\alpha^j\right)$ для каждого j и проверки полученных значений на нуль. Если величина $\Lambda\left(\alpha^{-k}\right)$ равна нулю, то α^k является взаимным к корню полинома локаторов ошибок и k-ый элемент кодовой комбинации содержит ошибку. Наиболее простым способом вычисления значения $\Lambda(X)$ в точке $\Lambda(\beta)$ является схема Горнера

$$\Lambda(\beta) = (..(((\Lambda_\nu\beta + \Lambda_{\nu-1})\beta + \Lambda_{\nu-2})\beta + \Lambda_{\nu-3})\beta + ... + \Lambda_0).$$

Для вычисления $\Lambda(\beta)$ по схеме Горнера требуется только ν умножений и ν сложений, где ν степень $\Lambda(X)$.

После определения локаторов ошибок с помощью ключевого уравнения находим значения ошибок. Для этого, используя значения сомножителей, входящих в равенство для $\Omega(X)$, перепишем ключевое уравнение следующим образом:

$$\Omega(X)=\left(\sum_{j=1}^{2t}\sum_{i=1}^{\nu}Y_i X_i^j X^j\right)\prod_{i=1}^{\nu}(1-XX_i)\bmod X^{2t}= \\ =\sum_{i=1}^{\nu}Y_i X_i^j X^j\left((1-XX_i)\sum_{j=1}^{2t}X_i^j X^j\right)\prod_{l\neq i}(1-XX_l)\bmod X^{2t} \quad (4.3.12)$$

Выражение в скобках является разложением $\left(1-X^{2t}X_i^{2t}\right)$. Таким образом, получаем

$$\Omega(X)=\sum_{i=1}^{\nu}Y_i X_i^j X^j\left(1-X^{2t}X_i^{2t}\right)\prod_{l\neq i}(1-XX_l)\bmod X^{2t}. \quad (4.3.13)$$

Приводя это выражение по модулю X^{2t}, получаем

$$\Omega(X)=\sum_{i=1}^{\nu}Y_i X_i\prod_{l\neq i}(1-XX_l). \quad (4.3.14)$$

Вычислим полином значений ошибок на позиции l

$$\Omega\left(X_l^{-1}\right)=Y_l X_l\prod_{j\neq l}\left(1-X_j^{-1}X_l\right) \quad (4.3.15)$$

откуда

$$Y_l=\frac{\Omega\left(X_l^{-1}\right)X_l^{-1}}{\prod_{j\neq l}\left(1-X_l^{-1}X_l\right)}. \quad (4.3.16)$$

Найдем производную от полинома локаторов ошибок $\Lambda(X)$:

$$\Lambda'(X)=\sum_{i=1}^{\nu}X_i\prod_{j\neq i}(1-XX_i). \quad (4.3.17)$$

Для l-й позиции получаем

$$\Lambda'\left(X_l^{-1}\right)=X_l\prod_{j\neq l}\left(1-X_l^{-1}X_i\right). \quad (4.3.18)$$

С учетом последнего выражения Y_l значение ошибки на позиции l принимает вид

$$\Lambda'\left(X_l^{-1}\right)= X_l \prod_{j\neq l}\left(1- X_l^{-1}X_i\right). \qquad (4.3.19)$$

Данный способ вычисления значения ошибки известен в литературе как алгоритм Форни. Он позволяет найти значение ошибок по известным полиномам локаторов и значений ошибок. Указанные полиномы вычисляются в результате решения ключевого уравнения.

Алгоритм Берлекэмпа – Месси решения ключевого уравнения

Нахождение решения ключевого уравнения, имеющего минимальную степень, эквивалентно построению регистра сдвига минимальной длины с обратными связями, отображающими $\Lambda(X)$, порождающим первые $2t$ членов $S(X)$. Основное содержание алгоритма Берлекэмпа – Месси формулируется следующим образом. Вначале находят самый короткий регистр сдвига, генерирующий S_1. Далее проверяют, порождает ли этот регистр также S_2. Если порождает, то данный регистр по-прежнему остается наилучшим решением, и нужно проверить, порождает ли он следующие символы синдромного полинома. На каком-то шаге очередной символ уже не будет генерироваться. В этот момент нужно изменить регистр таким образом, чтобы он:

1. правильно предсказывал следующий символ
2. не менял предсказание предыдущих символов
3. увеличивал длину регистра на минимально возможную величину

Процесс вычисления продолжается до тех пор, пока не будут порождены первые $2t$ символов синдрома.

Алгоритм строится на основе итеративной процедуры. При каждой итерации должны сохраняться как полином связей $\Lambda(X)$, так и добавка. Для каждого нового члена $S(X)$ предусматривается проверка правильности предсказания этого символа текущим полиномом связи. Если предсказание правильно, то полином связей не меняется, а добавка умножается на X. Если предсказание неправильно, то изменяют текущий полином связей, прибавляя к нему добавку. После этого проверяют, увеличилась ли длина регистра. Если она не увеличилась, то текущую добавку оставляют. Если длина регистра возрастает, то лучшей добавкой считают предыдущий полином связей. Для недвоичных полей добавку нормируют, чтобы невязка стала равной 1. Далее при каждом исправлении эту нормализованную добавку умножают на значение текущей невязки.

Блок-схема алгоритма Берлекэмпа–Месси приведена на Рис. 18.

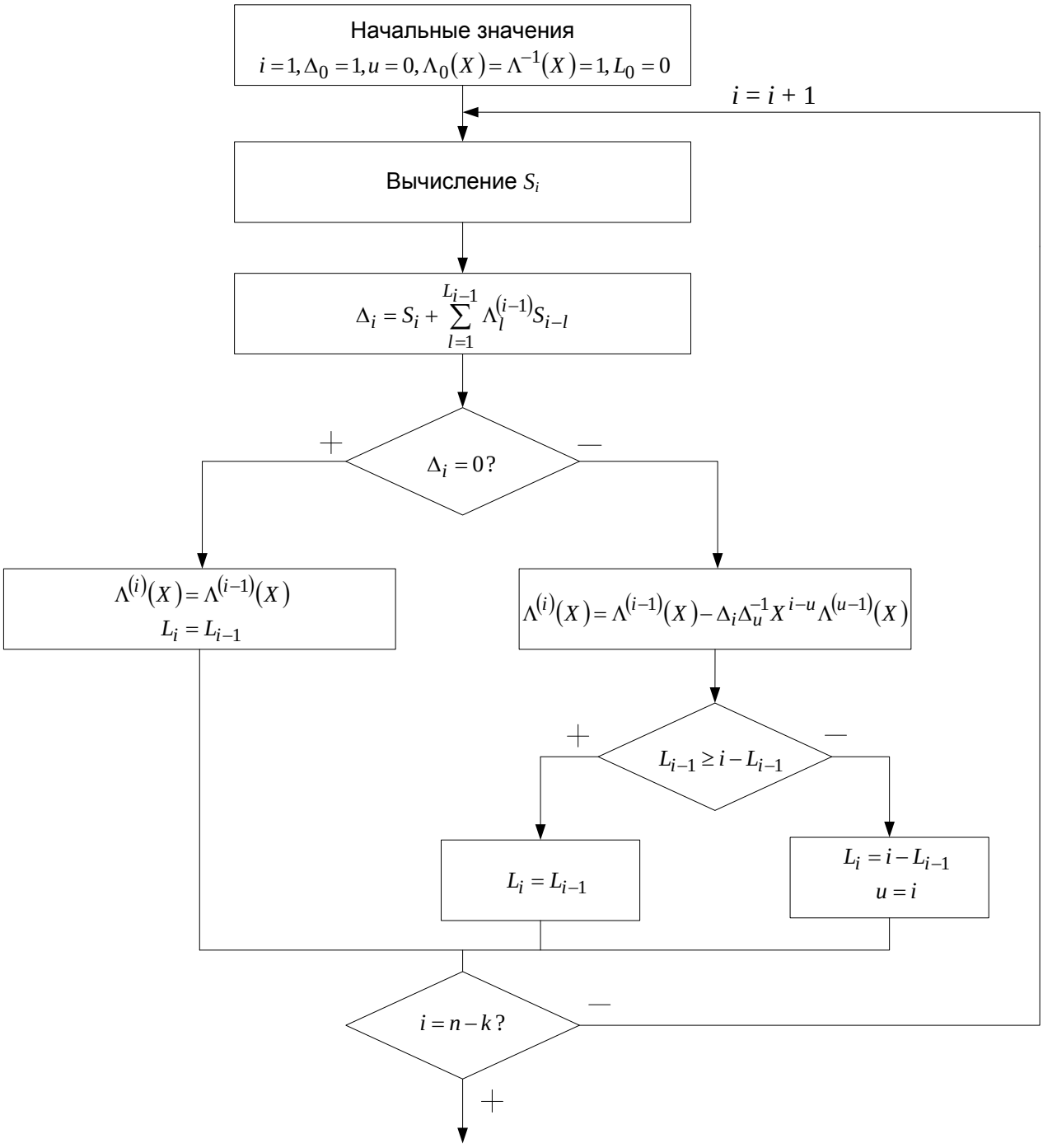


Рис. 18. Блок-схема алгоритма Берлекэмпа–Месси

Алгоритм выполняется за $2t = n - k$ итераций. Для удобства введены следующие обозначения: i - номер итерации, $i = 1,2,...,n-k = 2t$, S_i - компонента полинома синдрома, Δ_i - ошибка в вычислении S_i (невязка), $\Lambda^{(i)}(X)$ полином локаторов ошибок на итерации i, в

соответствии с компонентами которого строится регистр сдвига минимальной длины, L_i - длина регистра сдвига, $\Delta_i\Delta_u^{-1}$ - нормирующий коэффициент добавки.

Процедура декодирования кодов Рида-Соломона, с исправлением ошибок представлена на Рис. 19. Для исправления ошибок декодер выполняет последовательность вычислений, реализующих алгоритм Берлекэмпа–Месси и алгоритм Форни.

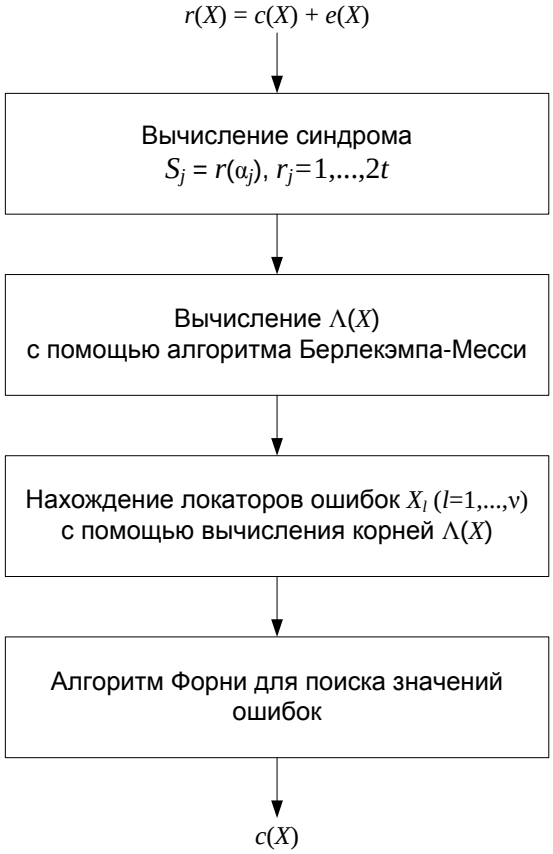


Рис. 19. Блок схема декодирования кодов Рида-Соломона

Рассмотрим код Рида Соломона (15, 9) над полем $GF\left(2^4\right)$. Число избыточных элементов в кодовой комбинации $n-k=6$. Минимальное кодовое расстояние $d=n-k+1=7$. Число гарантировано исправляемых ошибок $t=3$. Корни порождающего полинома $\alpha,\alpha^2,\alpha^3,\alpha^4,\alpha^5,\alpha^6$ элементы $GF\left(2^4\right)$. Порождающий полином

$$g(X)=(X+\alpha)\left(X+\alpha^2\right)\left(X+\alpha^3\right)\left(X+\alpha^4\right)\left(X+\alpha^5\right)\left(X+\alpha^6\right)=$$
$$=X^6+\alpha^6X^5+\alpha^{14}X^4+\alpha^4X^3+\alpha^6X^2+\alpha^9X+\alpha^6.$$

Пусть от передатчика в канал передана комбинация $c(X)=0$, а приемник принял комбинацию

$$r(X)=e(X)=\alpha X^7+\alpha^5X^5+\alpha^{11}X^2.$$

Компоненты синдрома вычисляются по формуле

$$S_j=\alpha\left(\alpha^j\right)^7+\alpha^5\left(\alpha^j\right)^5+\alpha^{11}\left(\alpha^j\right)^2,$$

где $j=1,2,...,6$. Синдромный полином имеет вид

$$S(X)=\alpha^{11}X^6+\alpha^0X^5+\alpha^{13}X^4+\alpha^{14}X^3+\alpha^0X^2+\alpha^{12}X^1.$$

Вычисление полинома локаторов ошибок $\Lambda(X)$ по алгоритму Берлекэмпа–Месси представлено на Рис. 20. В результате решения ключевого уравнения получаем полином локаторов ошибок

$$\Lambda(X)=1+\alpha^{14}X+\alpha^{11}X^2+\alpha^{14}X^3=\left(1+\alpha^7X\right)\left(1+\alpha^5X\right)\left(1+\alpha^2X\right).$$

Обратим внимание, что полученный полином $\Lambda(X)$ содержит в качестве локаторов ошибок X_2,X_5,X_7, что соответствует $e(X)$. Произведем вычисление полинома значений ошибок $\Omega(X)$. Для этого получим полином значений ошибок непосредственным вычислением

$$\Omega(X)=S(X)\Lambda(X). \qquad (4.3.20)$$

Для выполнения умножения полиномов представим $S(X)$ в виде, соответствующем $\Lambda(X)$

$$S(X)=\alpha^{12}+X+\alpha^{14}X^2+\alpha^{13}X^3+X^4+\alpha^{11}X^5.$$

Ограничивая число членов произведения 5-й степенью (по $\bmod X^{2t=6}$), получаем

$$\Omega(X)=\alpha^{12}+\left(\alpha^{11}+1\right)X+\overbrace{\left(\alpha^8+\alpha^{14}+\alpha^{14}\right)}^{=0}X^2+\overbrace{\left(\alpha^{11}+\alpha^{11}+\alpha^{13}+\alpha^{13}\right)}^{=0}X^3+$$
$$+\underbrace{\left(\alpha^{14}+\alpha^{10}+\alpha^{12}+1\right)}_{=0}X^4+\underbrace{\left(\alpha^{13}+\alpha^9+\alpha^{14}+\alpha^{11}\right)}_{=0}X^5+...=\alpha^{12}+\alpha^{12}X+\alpha^8X^2.$$

Полином связей $\Lambda^{(i)}(X)$	Длина регистра L_i	Регистры сдвига с обратными связями	Генерируемые значения $S(X)$

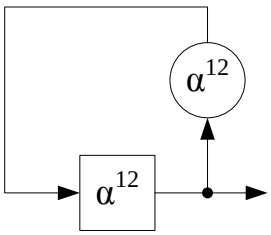
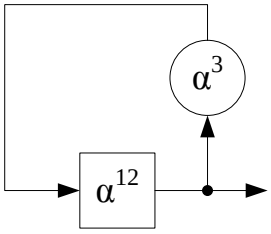
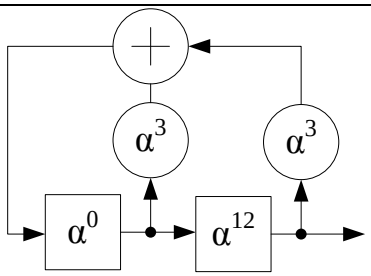
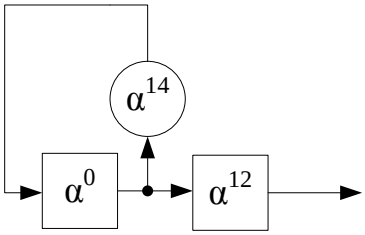
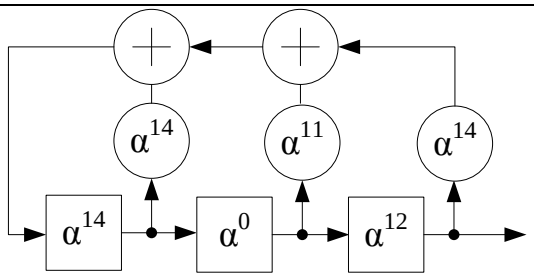
$1+\alpha^{12}X$	1		$S_1=\alpha^{12}$
$1+\alpha^{3}X$	1		$S_1=\alpha^{12}$ $S_2=\alpha^{0}$
$1+\alpha^{3}X+\alpha^{3}X^{2}$	2		$S_1=\alpha^{12}$ $S_2=\alpha^{0}$ $S_3=\alpha^{14}$
$1+\alpha^{14}X$	2		$S_1=\alpha^{12}$ $S_2=\alpha^{0}$ $S_3=\alpha^{14}$ $S_4=\alpha^{13}$
$1+\alpha^{14}X+\alpha^{11}X^{2}+\alpha^{14}X^{3}$	3		$S_1=\alpha^{12}$ $S_2=\alpha^{0}$ $S_3=\alpha^{14}$ $S_4=\alpha^{13}$ $S_5=\alpha^{0}$ $S_6=\alpha^{11}$

Рис. 20 Построение регистра минимальной длины, порождающего $S(X)$, с помощью алгоритма Берлекэмпа–Месси

Далее по алгоритму Форни находим значения ошибок. Для этого вычисляем

$$\Lambda'(X) = \alpha^{14} + \alpha^{14} X^2$$

и составляем выражение для Y_l

$$Y_l = \frac{\Omega\left(X_l^{-1}\right)}{\Lambda'\left(X_l^{-1}\right)},$$

где X_l локатор ошибки. Подставляя значения $X_2^{-1} = \alpha^{-2} = \alpha^{13}$, $X_5^{-1} = \alpha^{-5} = \alpha^{10}$ и $X_7^{-1} = \alpha^{-7} = \alpha^8$, получим $Y_2 = \alpha^{11}$, $Y_5 = \alpha^5$, $Y_7 = \alpha$. Таким образом, вычисленный полином ошибок $e(X) = \alpha X^7 + \alpha^5 X^5 + \alpha^{11} X^2$ соответствует заданному полиному ошибок.

Коды Рида-Соломона получили достаточно широкое применение как в системах хранения данных (CD/DVD), так и в системах передачи данных (DVB, 802.16d – фиксированный WiMAX). Например, (255, 239) код Рида-Соломона определенный над полем $GF(2^8)$ с размером символа 8 бит (1 байт), используется для кодирования данных в стандарте фиксированной связи WiMAX. Данный код позволяет исправлять до 8 символьных ошибок на длине кодового слова.

Глава 5

СВЕРТОЧНЫЕ КОДЫ

5.1. ПОНЯТИЕ СВЕРТОЧНРОГО КОДИРОВАНИЯ

Питер Элиас впервые предложил использовать сверточные коды в 1955 году как альтернативу блоковым кодам. В 1963 году Мессей предложил практическую пороговую схему декодирования сверточных кодов. Однако предложенный Мессей алгоритм декодирования не обеспечивал достаточную помехоустойчивость. В 1967 году Витерби предложил оптимальный алгоритм декодирования сверточных кодов по критерию максимума правдоподобия. В силу относительно невысокой вычислительной сложности алгоритм Витерби и сверточное кодирование получили широкое применение в практических задачах передачи информации. В настоящий момент сверточное кодирование является неотъемлемой частью современных стандартов радиосвязи.

Основным отличием сверточных от блоковых кодов является то, что n выходных кодовых символов зависят не только от текущего информационного блока длины k бит, но и от m предыдущих информационных блоков. Общая структура (n,k,m) сверточного кода показана на Рис. 21. Сверточное кодирование может быть реализовано с помощью линейной цепи имеющей k входов, n выходов и памяти размерности m. Как правило, n и k являются небольшими целыми числами, удовлетворяющие неравенству $n > k$. В то же время, для обеспечения необходимой помехоустойчивости размерность памяти m должна быть существенно больше параметров n и k.

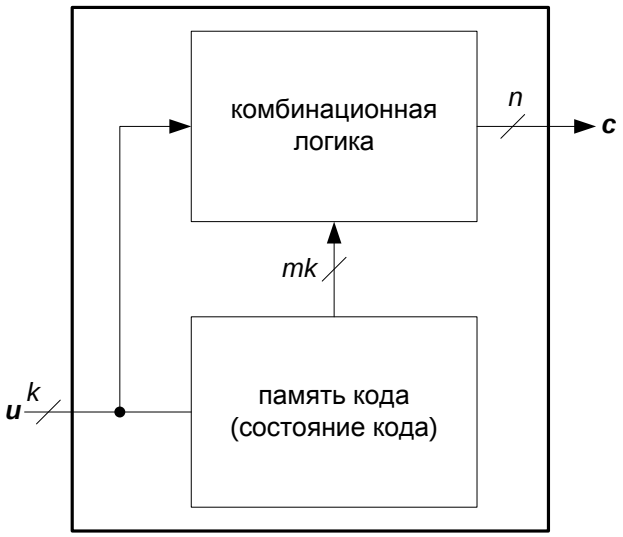


Рис. 21 Структура сверточного кода

Пример сверточного (2,1,2) кода приведен на Рис. 22

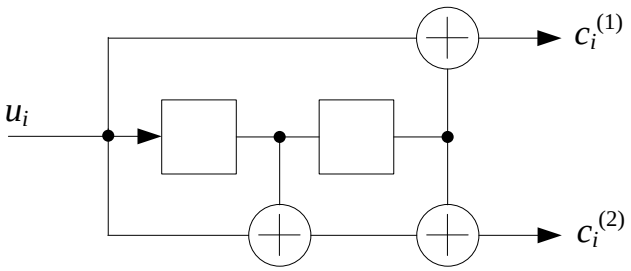


Рис. 22. Пример (2,1,2) сверточного кода

5.2. МЕТОДЫ ПРЕДСТАВЛЕНИЯ СВЕРТОЧНЫХ КОДОВ

Сверточные коды могут быть реализованы с помощью линейных цепей, использующих сумматоры и регистры памяти. В этом случае код может быть описан с помощью, так называемой, *диаграммы состояний*. Узлы в диаграмме представляют собой состояния кода, задаваемые содержимым регистров сдвига, а ребра описывают переход кода из одного состояния в другое. Пусть для (n,k,m) сверточного кода параметр K_i определяет

число элементов памяти для входа i, тогда $K = \sum_{i=1}^{k} K_i$ задает суммарную память сверточного кодера. Тогда полное состояние кода в момент времени l задается вектором

$$\left(u_{l-1}^{(1)} \quad u_{l-2}^{(1)} \quad \cdots \quad u_{l-K_1}^{(1)} \quad u_{l-1}^{(2)} \quad u_{l-2}^{(2)} \quad \cdots \quad u_{l-K_2}^{(1)} \quad \cdots \quad u_{l-1}^{(k)} \quad u_{l-2}^{(k)} \quad \cdots \quad u_{l-K_k}^{(k)}\right), \qquad (5.2.1)$$

а общее число возможных состояний кода будет равно 2^K. Памятью сверточного кода называется величина, $m = \max_{1 \le i \le k}(K_i)$, а длиной *кодового ограничения*[2]

$$\nu = n \cdot \left(\max_{1 \le i \le k} K_i + 1\right) \qquad (5.2.2)$$

Каждый информационный блок длины k бит приводит к переходу кода из одного стояния в другое. В этом случае общее число ребер выходящих из каждого состояния равно 2^k. Для удобства описания каждое ребро в диаграмме состояний помечается k входными информационными битами $\left(u_l^{(1)} \quad u_l^{(2)} \quad \cdots \quad u_l^{(k)}\right)$ и n выходными кодовыми битами $\left(c_l^{(1)} \quad c_l^{(2)} \quad \cdots \quad c_l^{(n)}\right)$. Состояния кода, как правило, обозначаются, как $S_0, S_1, ..., S_{2^K-1}$, где индекс i состояния S_i соответствует десятичному представлению содержимого регистров сдвига кода. Для каждой входной информационной или кодовой последовательностей, в диаграмме состояний может быть проложен некоторый путь, проходящий через ее узлы.

Пример диаграммы состояний для рассматриваемого (2,1,2) сверточного кода показан на Рис. 23.

[2] На практике используется несколько определений длины кодового ограничения. Другим способом определения длины кодового ограничения является суммарная память кода *K*.

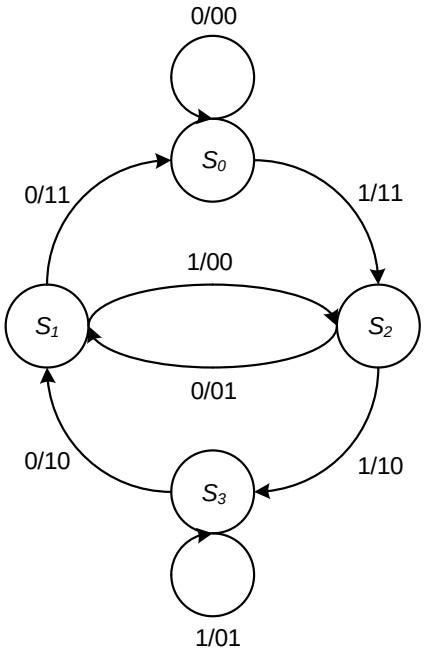


Рис. 23. Диаграмма состояния (2,1,2) сверточного кода

Помимо описания сверточного кода диаграмма состояний может быть также использована для анализа его структуры. В частности, с помощью диаграммы состояний можно определить число и вес всех ненулевых кодовых последовательностей (т.н. распределение веса кода). Для этого диаграмма состояний модифицируется следующим образом: удаляется переход из нулевого стояния в нулевое, а нулевое состояние разделяется на начальное и конечное состояния. Каждое ребро помечается X^i, где i число ненулевых бит на выходе кодера см. Рис. 24.

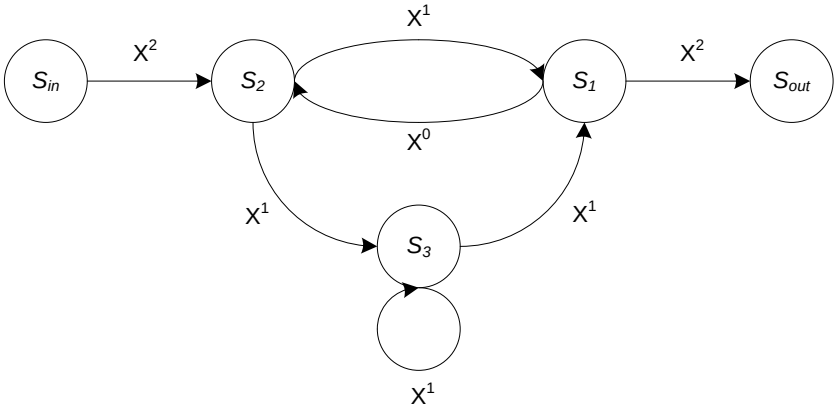


Рис. 24. Модифицированная диаграмма состояний (2,1,2) кода

Тогда путь, соединяющий начальное и конечные состояния, представляет собой ненулевую кодовую последовательность выходящую и входящую в нулевое состояние кода. При этом кодовые последовательности, проходящие через нулевое состояние более одного раза, могут быть рассмотрены как несколько более коротких последовательностей, начинающихся и заканчивающихся в нулевом состоянии.

Уравнения, определяющие переходы в модифицированной диаграмме состояний, определяются следующим образом

$$\begin{aligned} S_2 &= S_1 + X^2 S_{in} \\ S_1 &= XS_2 + XS_3 \\ S_3 &= XS_2 + XS_3 \\ S_{out} &= X^2 S_1 \end{aligned}, \qquad (5.2.3)$$

Распределение веса кода может быть получено из порождающей функции модифицированной диаграммы состояний следующим образом

$$T(X) = \frac{S_{out}}{S_{in}} = \sum_i A_i X^i = \frac{X^5}{1-2X} = X^5 + 2X^6 + 4X^7 + 8X^8 + \ldots. \qquad (5.2.4)$$

Множество весов ненулевых кодовых последовательностей определяется множеством степеней полинома $T(X)$, а число путей веса i коэффициентом A_i. Например, для рассматриваемого (2,1,2) сверточного минимальный вес ненулевой кодовой последовательности равен 5 и число таких путей равно 1.

Распределение веса (вес кодовых последовательностей их число) определяет помехоустойчивость сверточного кода. Так из распределения веса кода можно вычислить *свободное расстояние кода,* определяемое выражением

$$d_{free} = \min_{\mathbf{c}_i, \mathbf{c}_j, \mathbf{c}_i \neq \mathbf{c}_j} \left(d_H(\mathbf{c}_i, \mathbf{c}_j) \right). \qquad (5.2.5)$$

В силу линейности сверточных кодов свободное расстояние кода может быть также получено как минимальный вес ненулевой кодовой последовательности

$$d_{free} = \min_{\mathbf{c}_i, \mathbf{c}_i \neq \mathbf{0}} \left(w_H(\mathbf{c}_i) \right). \qquad (5.2.6)$$

Тогда для последовательностей большой длины свободное расстояние кода d_{free} равно минимальной степени слагаемого порождающей функции (5.2.4).

Другим способом представления сверточных кодов является *решетчатая диаграмма*. Решетчатая диаграмма может быть получена путем разворота диаграммы состояний кода во времени. Пример решетчатой диаграммы для рассматриваемого (2,1,2) сверточного кода

показан на Рис. 25. Начальное состояние кода в решетчатой диаграмме соответствует нулевому состоянию кода.

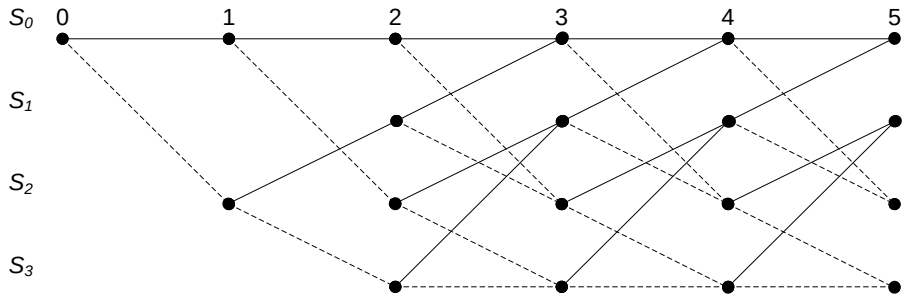


Рис. 25. Решетчатая диаграмма (2,1,2) сверточного кода

Сплошной линией в решетчатой диаграмме показаны ребра, соответствующие нулевому информационному биту, а пунктирной линией ребра соответствующие единичному информационному биту. Отметим, что число разрешенных состояний в решетке удваивается каждый момент времени и достигает максимального значения $2^K = 4$ в момент времени $l = 2$. Далее для $l > 2$ структура решетки начинает повторяться. Процедура кодирования может быть представлена как прокладывание пути в решетчатой диаграмме. Рис. 26 иллюстрирует процедуру кодирования информационной последовательности $\mathbf{u} = (1,0,1,1,1)$.

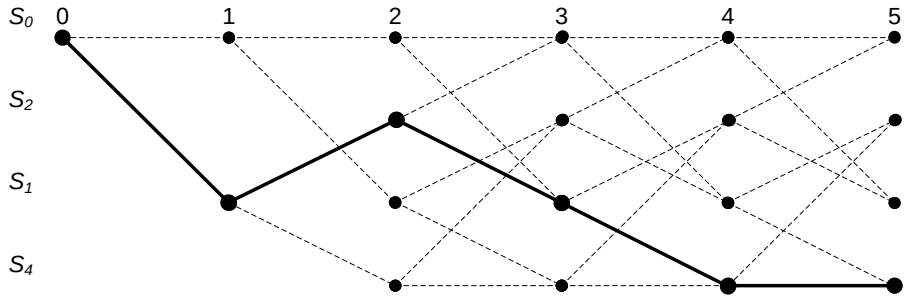


Рис. 26. Пример кодирования для последовательности $\mathbf{u} = (1,0,1,1,1)$

Сплошной линией показан путь в решетке соответствующей информационной последовательности, а пунктирной оставшиеся пути решетки.

5.3. МАТРИЧНОЕ ОПИСАНИЕ ПРОЦЕДУРЫ СВЕРТОЧНОГО КОДИРОВАНИЯ

Процедура кодирования сверточных кодов может быть также описана с помощью матричных операций. В силу линейности цепи кодирования, последовательность на каждом выходе j сверточного кода может быть представлена как свертка информационной последовательности $\mathbf{u}$ и импульсной характеристики линейной цепи $\mathbf{g}^{(j)}$ для $(n,1,m)$ кода. Импульсная характеристика $\mathbf{g}^{(j)}$ называется *порождающей последовательностью* кода. Заметим, что множество $\left\{\mathbf{g}^{(j)}\right\}$ полностью определяет сверточный код. Тогда для случая $(n,1.m)$ кода процедуру кодирования можно представить следующим образом

$$c_l^{(j)} = \sum_{i=0}^{m} u_{l-i} g_i^{(j)} \tag{5.3.1}$$

Для каждого момента времени выходы сверточного кода, как правило, объединяются в общий поток, т.е.

$$\left(c_0^{(1)} \quad c_0^{(2)} \quad \cdots \quad c_0^{(n)} \quad c_1^{(1)} \quad \cdots \quad c_1^{(n)} \quad \cdots\right), \tag{5.3.2}$$

тогда порождающая матрица сверточного кода будет иметь следующую структуру

$$\mathbf{G} = \begin{pmatrix} g_0^{(1)} \ldots g_0^{(n)} & \cdots & g_m^{(1)} \ldots g_m^{(n)} & & & \\ & g_0^{(1)} \ldots g_0^{(n)} & \cdots & g_m^{(1)} \ldots g_m^{(n)} & & \\ & & g_0^{(1)} \ldots g_0^{(n)} & \cdots & g_m^{(1)} \ldots g_m^{(n)} & \\ & & & \ddots & & \ddots \end{pmatrix}, \tag{5.3.3}$$

где пустые области матрицы $\mathbf{G}$ равны 0. Для кодовой последовательности длины L, матрица $\mathbf{G}$ имеет L строк и $n \cdot (L+m)$ столбцов. Процедура сверточного кодирования при этом записывается следующим образом

$$\mathbf{c} = \mathbf{uG} \tag{5.3.4}$$

Рассмотрим пример сверточного кодирования для (2,1,2) кода и информационной последовательности $\mathbf{u} = (1,0,1,1,1,1)$. Порождающие последовательности кода равны (1,0,1) и (1,1,1). Поражающая матрица этого кода равна

$$\mathbf{G}=\begin{pmatrix} 11 & 01 & 11 & & & & & \\ & 11 & 01 & 11 & & & & \\ & & 11 & 01 & 11 & & & \\ & & & 11 & 01 & 11 & & \\ & & & & 11 & 01 & 11 & \\ & & & & & 11 & 01 & 11 \end{pmatrix}.$$

Для информационной последовательности $\mathbf{u}=(1,0,1,1,1,1)$ с помощью выражения (5.3.4) получим кодовую последовательность $c=(11,01,00,10,01,01,10,11)$.

Обобщим порождающую матрицу для случая произвольного $(n,k.m)$ сверточного кода

$$\mathbf{G}=\begin{pmatrix} \mathbf{G}_0 & \cdots & \mathbf{G}_m & & & \\ & \mathbf{G}_0 & \cdots & \mathbf{G}_m & & \\ & & \mathbf{G}_0 & \cdots & \mathbf{G}_m & \\ & & & \ddots & & \ddots \end{pmatrix}, \qquad (5.3.5)$$

где матрица $\mathbf{G}_l$ определяется выражением

$$\mathbf{G}_l=\begin{pmatrix} g_{1,l}^{(1)} & g_{1,l}^{(2)} & \cdots & g_{1,l}^{(n)} \\ g_{2,l}^{(1)} & g_{2,l}^{(2)} & \cdots & g_{2,l}^{(n)} \\ \vdots & & & \vdots \\ g_{k,l}^{(1)} & g_{k,l}^{(2)} & \cdots & g_{k,l}^{(n)} \end{pmatrix}. \qquad (5.3.6)$$

При этом информационная последовательность на выходе кода имеет следующую структуру

$$\mathbf{u}=(\mathbf{u}_0 \quad \mathbf{u}_1 \quad \cdots)=\left(u_0^{(1)}u_0^{(2)}\cdots u_0^{(k)} \quad u_1^{(1)}u_1^{(2)}\cdots u_1^{(k)} \quad \cdots\right). \qquad (5.3.7)$$

5.4. ПОЛИНОМИАЛЬНОЕ ОПИСАНИЕ СВЕРТОЧНОГО КОДИРОВАНИЯ

В линейных системах каждую последовательность можно заменить некоторым полиномом, а операцию свертки на более удобную операцию умножения полиномов. Например, для (2,1,m) кода процедура кодирования запишется следующим образом

$$\begin{aligned} c^{(1)}(D) &= u(D)\cdot g_1(D) \\ c^{(2)}(D) &= u(D)\cdot g_2(D) \end{aligned}, \tag{5.4.1}$$

где $u(D)=u_0+u_1D+u_2D^2+\cdots$ информационный полином, $g^{(1)}(D)=g_0^{(1)}+g_1^{(1)}D+g_2^{(1)}D^2+\cdots+g_m^{(1)}D^m$ и $g^{(2)}(D)=g_0^{(2)}+g_1^{(2)}D+g_2^{(2)}D^2+\cdots+g_m^{(2)}D^m$ порождающие полиномы, $c^{(1)}(D)=c_0^{(1)}+c_1^{(1)}D+c_2^{(1)}D^2+\ldots$ и $c^{(2)}(D)=c_0^{(2)}+c_1^{(2)}D+c_2^{(2)}D^2+\ldots$ кодовые полиномы. Общая кодовая последовательность на выходе

$$c(D)=c^{(1)}\left(D^2\right)+D\cdot c^{(2)}\left(D^2\right). \tag{5.4.2}$$

Обобщим полиномиальное описание процедуры кодирования на случай $(n,k.m)$ сверточного кода. Пусть $U(D)=\left(u^{(1)}(D)\ \ u^{(2)}(D)\ \ \cdots\ \ u^{(k)}(D)\right)$ набор полиномов соответствующих k входным информационным последовательностям, $C(D)=\left(c^{(1)}(D)\ \ c^{(2)}(D)\ \ \cdots\ \ c^{(k)}(D)\right)$ набор полиномов соответствующих n выходным кодовым последовательностям, а $g_i^{(j)}(D)$ порождающий полином, соответствующий входу i и входу j сверточного кода. Тогда процедуру кодирования можно записать следующим образом

$$C(D)=U(D)G(D), \tag{5.4.3}$$

где

$$G(D)=\begin{pmatrix} g_1^{(1)}(D) & g_1^{(2)}(D) & \cdots & g_1^{(n)}(D) \\ g_2^{(1)}(D) & g_2^{(2)}(D) & \cdots & g_2^{(n)}(D) \\ \vdots & & & \vdots \\ g_k^{(1)}(D) & g_k^{(2)}(D) & \cdots & g_k^{(n)}(D) \end{pmatrix}. \tag{5.4.4}$$

Например, для (2,1,7) сверточного кода, представленного на Рис. 27, поражающий полином запишется следующим образом

$$G(D)=\left(1+D^2+D^3+D^5+D^6 \quad 1+D+D^2+D^3+D^6\right)$$

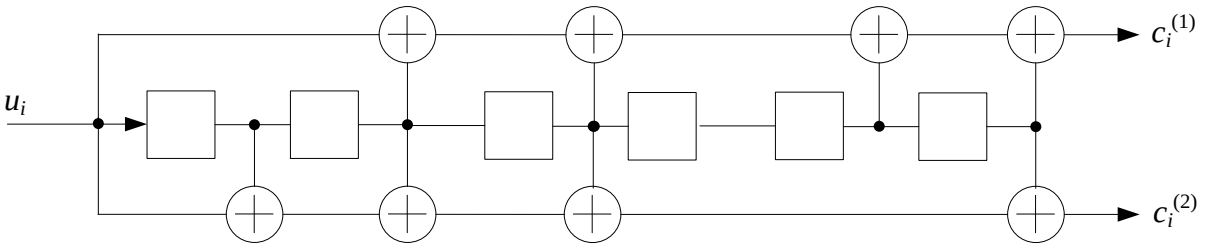


Рис. 27. Схема (2,1,7) сверточного кода

Код, представленный на Рис. 27 получил широкое применение в различных стандартах связи, таких как IEEE 802.11a и IEEE 802.16e.

5.5. СИСТЕМАТИЧЕСКИЕ СВЕРТОЧНЫЕ КОДЫ

Сверточный код называют систематическим, если входная информационная последовательность непосредственно поступает на один из выходов кодера без изменения. Систематические коды имеют некоторое преимущество при декодировании, т.к. информационная последовательность может быть получена непосредственно из принятой последовательности. Заметим, что несистематический сверточный код может быть приведен к систематическому виду следующим образом

$$c(D) = g^{(1)}\left(D^2\right) \cdot u\left(D^2\right) \cdot \left(1 + D \cdot \frac{g^{(2)}\left(D^2\right)}{g^{(1)}\left(D^2\right)}\right) = \tilde{u}\left(D^2\right) \cdot \left(1 + D \cdot \frac{g^{(2)}\left(D^2\right)}{g^{(1)}\left(D^2\right)}\right) \qquad (5.5.1)$$

Из выражения (5.5.1), легко видеть, что множество кодовых последовательностей кода с порождающими полиномами $\left\{g^{(1)}(D), g^{(2)}(D)\right\}$ и полиномами $\left\{1, \frac{g^{(2)}(D)}{g^{(1)}(D)}\right\}$ идентичны, т.е. сверточные коды $\left\{g^{(1)}(D), g^{(2)}(D)\right\}$ и $\left\{1, \frac{g^{(2)}(D)}{g^{(1)}(D)}\right\}$ эквивалентны. При этом изменяется отображение информационных последовательностей в кодовые, а код становится рекурсивным.

Пример рекурсивного систематического сверточного (2,1,2) кода показан на Рис. 28

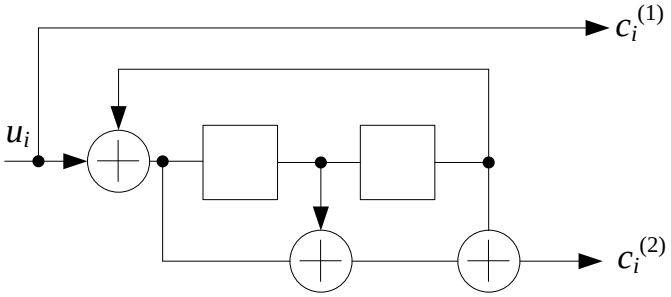


Рис. 28. Рекурсивный систематический (2,1,2) сверточный код

Сверточный код называется *катастрофическим*, если он отображает последовательность с бесконечным весом в кодовую последовательность конечного веса. Очевидно, что использовать в качестве помехоустойчивого кода такие схемы нельзя, т.к. конечное число ошибок в кодовом слове приведет к бесконечному числу ошибок в информационном слове. Оказывается, что необходимым и достаточным условием того, чтобы сверточный код $G(D)$ был некатастрофическим, является существование некоторого преобразования $G^{-1}(D)$, удовлетворяющего условию $G^{-1}(D)G(D)=1$.

5.6. ПРОЦЕДУРА ВЫКАЛЫВАНИЯ СВЕРТОЧНЫХ КОДОВ

Для практических приложений для контроля скорости передачи и помехоустойчивости очень важным является использование различных скоростей кода. При этом для упрощения процедуры кодирования и декодирования требуется сохранение структуры кода. Процедура выкалывания является эффективным способом достижения этих двух целей и заключается в исключении (выкалывании) определенных кодовых бит из передаваемой последовательности. Правило исключения кодовых бит задается с помощью, так называемого, шаблона выкалывания. Шаблон выкалывания представляет собой матрицу с числом строк равным числу выходов кода n и числом столбцов равным периоду выкалывания. Ноль в шаблоне выкалывания обозначает, что кодовый бит исключается из передачи. Пример процедуры выкалывания показан на Рис. 29 для шаблона выкалывания $\begin{bmatrix}1 & 0\\ 1 & 1\end{bmatrix}$.

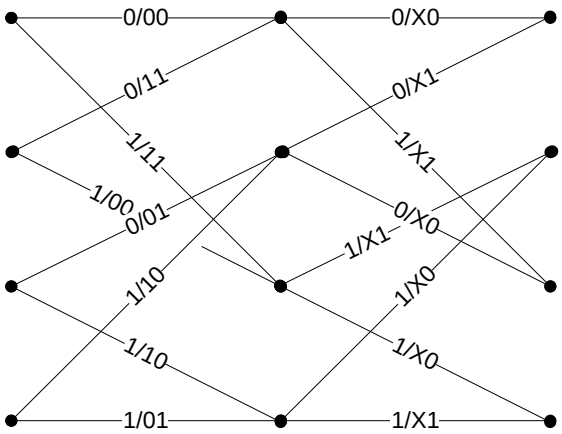


Рис. 29. Процедура выкалывания (2,1,2) сверточного кода

Исключенные из передачи биты, помечены на ребрах решетки символом X. При этом правило выкалывания кодовых бит повторяется каждые два сегмента решетки (входных информационных бит). Для рассматриваемого примера для каждых двух информационных бит на выходе кода получается три кодовых. В этом случае скорость кода равна 2/3. Заметим, что структура решетки кода (число состояний, переходы между состояний), полученная с помощью процедуры выкалывания, остается такой же, как и для исходного кода без выкалывания. Сохранение структуры кода позволяет существенно упростить процедуру декодирования особенно для высоких скоростей кода, т.к. решетка сверточного кода для $k > 1$, как правило, является более сложной.

В качестве примера рассмотрим структуру решетки для (3,2,2) кода с аналогичной скоростью кодирования и порождающей матрицей кода вида

$$G(D) = \begin{pmatrix} 1+D & 1+D & 1 \\ 0 & D & 1+D \end{pmatrix}.$$

Решетчатая диаграмма для такого кода показана на Рис. 30.

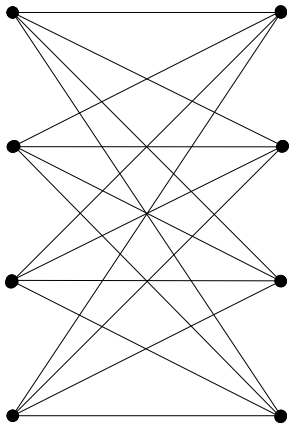


Рис. 30 Сегмент решетки (3,2,2) сверточного кода

Легко видеть, что структура кода (3,2,2) существенно сложнее структуры кода (2,1,2) с выкалыванием. Так, число ребер выходящих из каждого состояния для (3,2,2) кода в два раза больше чем для (2,1,2) кода с выкалыванием. Усложнение структуры, как правило, приводит к усложнению процедуры декодирования.

Другие примеры шаблонов выкалывания для сверточного кода

$$G(D) = \left(1 + D^2 + D^3 + D^5 + D^6 \quad 1 + D + D^2 + D^3 + D^6\right)$$

приведены в Таблице 8.

Таблица 8. Шаблоны выкалывания

Скорость кодирования	Шаблон выкалывания
2/3	$\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}$
3/4	$\begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}$
4/5	$\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix}$
5/6	$\begin{bmatrix} 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \end{bmatrix}$
7/8	$\begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 1 \end{bmatrix}$

Заметим, что процедура выкалывания сверточных кодов очень часто применяется в стандартах связи, поддерживающих сверточное кодирование для исправления ошибок. Так с помощью шаблонов выкалывания приведенных в Таблице 8 достигают скорости кода 1/2, 2/3 и 3/4 в стандартах связи IEEE 802.11a (Wi-Fi), IEEE 802.16e (WiMAX)

5.7. ЗАВЕРШЕНИЕ КОДИРОВАНИЯ ДЛЯ СВЕРТОЧНЫХ КОДОВ

В силу конечности информационной последовательности на практике применятся процедура завершения кодирования, возвращающая код в определенное состояние. Существуют два метода завершения кодирования сверточных кодов – метод с нулевым конечным состоянием и метод с циклической структурой. Для метода с нулевым конечным состоянием к концу информационной последовательности добавляются биты возвращающие код в нулевое состояние. Данный метод приводит к небольшому снижению скорости кода, связанной с передачей неинформационных добавочных бит. Решетка кода с процедурой завершения кодирования в нулевое состояние показана на Рис. 31.

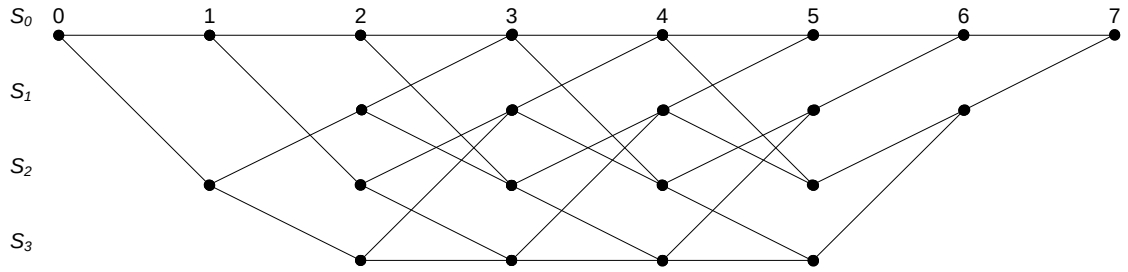


Рис. 31. Решетка кода с завершением кодирования в нулевое состояние

Другой метод с циклической структурой сверточного обеспечивает совпадение начального и конечного состояния кода в решетке при помощи специальной инициализации начального состояния в схеме кодирования. Поскольку инициализационные биты для такого метода зависят от информационной последовательности, начальное и конечное состояния кода заранее не известны на приемнике. Это приводит к некоторому усложнению процедуры декодирования сверточных кодов с циклической структурой, связанное с поиском начального и конечного состояний кода. Стоит отметить, что метод с циклической структурой кода не приводит к снижению скорости кода, что особенно важно для передачи информационных последовательностей малой длины.

5.8. ДЕКОДИРОВАНИЕ СВЕРТОЧНЫХ КОДОВ

В общем случае процедура декодирования сверточных кодов состоит из двух этапов, показанных показана на Рис. 32. На первом этапе, на основании принятой последовательности ***r*** происходит выбор кодовой последовательности. На втором по выбранной кодовой происходит восстановление информационной последовательности.

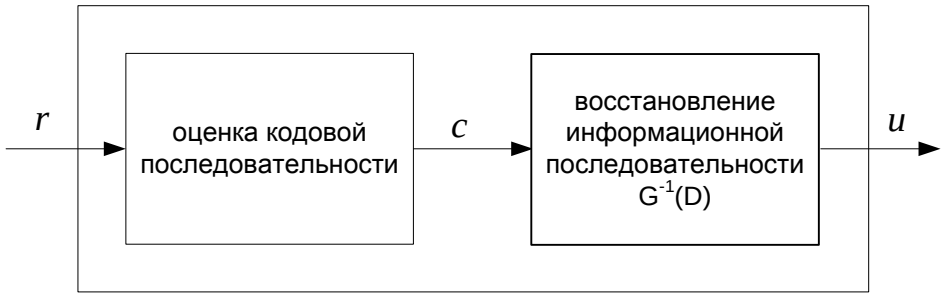


Рис. 32 Процедура декодирования сверточных кодов

Так для поиска кодовой последовательности может применяться алгоритм Витерби, осуществляющий оценку переданной последовательности по критерию максимума правдоподобия. Процедура декодирования включает поиск наиболее правдоподобной последовательности в решетке на основании принятой последовательности $\mathbf{r}$. В зависимости от используемого для поиска расстояния декодирование бывает с мягкими или жесткими решениями.

Предположим, что имеется информационная последовательность $\mathbf{u}=(\mathbf{u}_0,\mathbf{u}_1,\cdots,\mathbf{u}_{L-1})$, длины $k\cdot L$ информационных бит. Каждый информационный символ $\mathbf{u}_l$ состоит из k бит $\mathbf{u}_l=\left(u_l^{(0)},u_l^{(1)},\cdots,u_l^{(k-1)}\right)$. Предположим, что для кодирования используется (n,k,m) сверточный код. Тогда соответствующая кодовая последовательность $\mathbf{c}=(\mathbf{c}_0,\mathbf{c}_1,\cdots,\mathbf{c}_{L+m-1})$ состоит из $L+m$ символов и имеет длину $N=n\cdot(L+m)$ бит. Декодер, осуществляющий поиск кодовой последовательности по критерию максимума правдоподобия, должен найти путь в решетке кода обеспечивающий максимум вероятности

$$\hat{\mathbf{c}} = \arg\max_{\mathbf{c}} \{\mathrm{P}(\mathbf{r} \mid \mathbf{c})\}, \tag{5.8.1}$$

где $\mathbf{r} = (\mathbf{r}_0, \mathbf{r}_1, \cdots, \mathbf{r}_{L+m-1})$ принятая последовательность. Для канала без памяти данная вероятность записывается следующим выражением

$$\mathrm{P}(\mathbf{r} \mid \mathbf{c}) = \prod_{i=0}^{L+m-1} \mathrm{P}(\mathbf{r}_i \mid \mathbf{c}_i). \tag{5.8.2}$$

В силу монотонности логарифмической функции процедуру максимизации выражения (5.8.2) можно проводить в логарифмической области

$$\log\{\mathrm{P}(\mathbf{r} \mid \mathbf{c})\} = \sum_{i=0}^{L+m-1} \log\{\mathrm{P}(\mathbf{r}_i \mid \mathbf{c}_i)\}, \tag{5.8.3}$$

где $\mathrm{B}(\mathbf{r}_i \mid \mathbf{c}_i) = \log\{\mathrm{P}(\mathbf{r}_i \mid \mathbf{c}_i)\}$ называется *метрикой ребра*, определяемой суммой метрик бит.

$$\mathrm{B}(\mathbf{r}_i \mid \mathbf{c}_i) = \log\{\mathrm{P}(\mathbf{r}_i \mid \mathbf{c}_i)\} = \sum_{j=0}^{n-1} \log\left\{\mathrm{P}\left(r_i^{(j)} \mid c_i^{(j)}\right)\right\} \tag{5.8.4}$$

Тогда *метрика пути* $\mathbf{c}$ определяется как сумма метрик ребер

$$\mathrm{M}(\mathbf{r} \mid \mathbf{c}) = \sum_{i=0}^{L+m-1} \log\{\mathrm{P}(\mathbf{r}_i \mid \mathbf{c}_i)\} = \sum_{i=0}^{L+m-1} \sum_{j=0}^{n-1} \log\left\{\mathrm{P}\left(r_i^{(j)} \mid c_i^{(j)}\right)\right\}, \tag{5.8.5}$$

а *частичная метрика пути* $\mathbf{c}$ определяется как частичная сумма метрик ребер до момента времени j

$$\mathrm{M}_j(\mathbf{r} \mid \mathbf{c}) = \sum_{i=0}^{j} \log\{\mathrm{P}(\mathbf{r}_i \mid \mathbf{c}_i)\}. \tag{5.8.6}$$

5.9. АЛГОРИТМ ДЕКОДИРОВАНИЯ ВИТЕРБИ

Алгоритм Витерби осуществляет оценку кодовой последовательности по критерию максимума правдоподобия. Для этого на каждом этапе декодирования для каждого состояния кода алгоритм производит поиск кодовой последовательности имеющей максимальную частичную метрику пути. При этом путь, имеющий максимальную метрику, сохраняется.

Алгоритм Витерби можно формально разбить на следующие этапы:

Инициализация

Предполагается, что в момент времени $i = 0$, состояние кода известно. Тогда частичная метрика пути для каждого состояния инициализируется следующим образом

$$\mathrm{M}_0(s) = \begin{cases} 0, & s = 0 \\ -\infty, & s \neq 0 \end{cases}, \tag{5.9.1}$$

где s состояние кода. Переходим к моменту времени $i = 1$.

Суммирование

В момент времени i продолжаем все возможные пути, выходящие из всех состояний кода момента времени $i-1$. Вычисляем метрики продолженных путей путем суммирования частичной метрики пути состояния в предшествующий момент времени $\mathrm{M}_{i-1}(s)$ и метрики выходящего из этого состояния ребра.

Сравнение

Для каждого состояния s находим путь с наибольшей частичной метрикой. Этот путь называется *выжившим* для данного состояния в момент i.

Выбор

Определяем метрику частичного пути $\mathrm{M}_i(s)$ для каждого состояния равной метрике выжившего пути. Удаляем из решетчатой диаграммы все остальные пути. Увеличиваем i на единицу. Если $i < L + m$, то перейти к шагу *Суммирование*.

Восстановление

Двигаясь, по выжившему пути из состояния $s = 0$ в состояние соответствующее моменту $i = L + m - 1$ восстанавливаем исходное сообщение.

В силу принципа математической индукции, путь, найденный алгоритмом Витерби является максимально правдоподобным.

Вычислительная сложность алгоритма Витерби

Для двоичного сверточного кода, с полным числом регистров K, общее число состояний равно 2^K. Число ребер выходящих и входящих в каждое состояние равно 2^k. Тогда для каждого сегмента решетки имеется 2^{K+k} различных путей и 2^K выживших путей. Для каждого состояния на каждом этапе декодирования необходимо вычислить метрики всех продолженных 2^{K+k} путей и найти наилучшую метрику для каждого состояния, путем сравнения 2^k метрик. Легко видеть, что сложность вычислений декодирования экспоненциально возрастает с ростом числа сдвиговых регистров сверточного кода.

Алгоритм декодирования Витерби с жесткими решениями

Декодирование для двоичного симметричного канала называется декодированием с жесткими решениями, т.к. декодер получает на вход непосредственные оценки переданных бит без информации о надежности этих оценок. Для двоичного симметричного канала с вероятностью ошибки $p \le 1/2$ принятая последовательность является двоичной, тогда

$$\log\{\mathrm{P}(\mathbf{r}\,|\,\mathbf{c})\} = d(\mathbf{r},\mathbf{c})\cdot\log\left(\frac{p}{1-p}\right) + N\cdot\log(1-p), \qquad (5.9.2)$$

где $d(\mathbf{r},\mathbf{c})$ расстояние Хэмминга между принятой и кодовой последовательностями. Поскольку $\log(p/(1-p)) < 0$, а второе слагаемое выражения (5.9.2) не зависит от последовательности $\mathbf{c}$, максимально правдоподобная оценка обеспечивается при минимуме расстояния Хэмминга

$$d(\mathbf{r},\mathbf{c}) = \sum_{i=0}^{L+m-1} d(\mathbf{r}_i,\mathbf{c}_i) = \sum_{i=0}^{N-1}\sum_{j=0}^{n-1} d\left(r_i^{(j)}, c_i^{(j)}\right). \qquad (5.9.3)$$

Таким образом, метрика ребра и пути для двоично-симметричного канала вычисляется по расстоянию Хэмминга.

Пример декодирования алгоритма Витерби с жесткими решениями

Рассмотрим пример декодирования с помощью алгоритма Витерби. Пусть в кодовой последовательности (00, 11, 10, 10, 11, 00, 00) произошла ошибка во второй момент времени (на четвертом бите). Тогда принятая последовательность, поступающая на устройство декодирования, равна (00, 10, 10, 10, 11, 00, 00). Декодирование Витерби на каждом этапе показано на Рис. 33. С правой стороны показаны частичные метрики пути для каждого

состояния. На итерациях 1-2 алгоритм Витерби не производит выбор пути, т.к. число ребер входящих в каждое состояние равно 1. В силу завершения кодирования в нулевое состояние число разрешенных состояний в момент времени 6 и 7 равно 2 и 1 соответственно. На последней итерации декодер получает единственный путь, имеющий минимальное расстояние Хэмминга от принятой двоичной последовательности. Это расстояние равно 1, что соответствует числу ошибок произошедших на всей длине переданной последовательности.

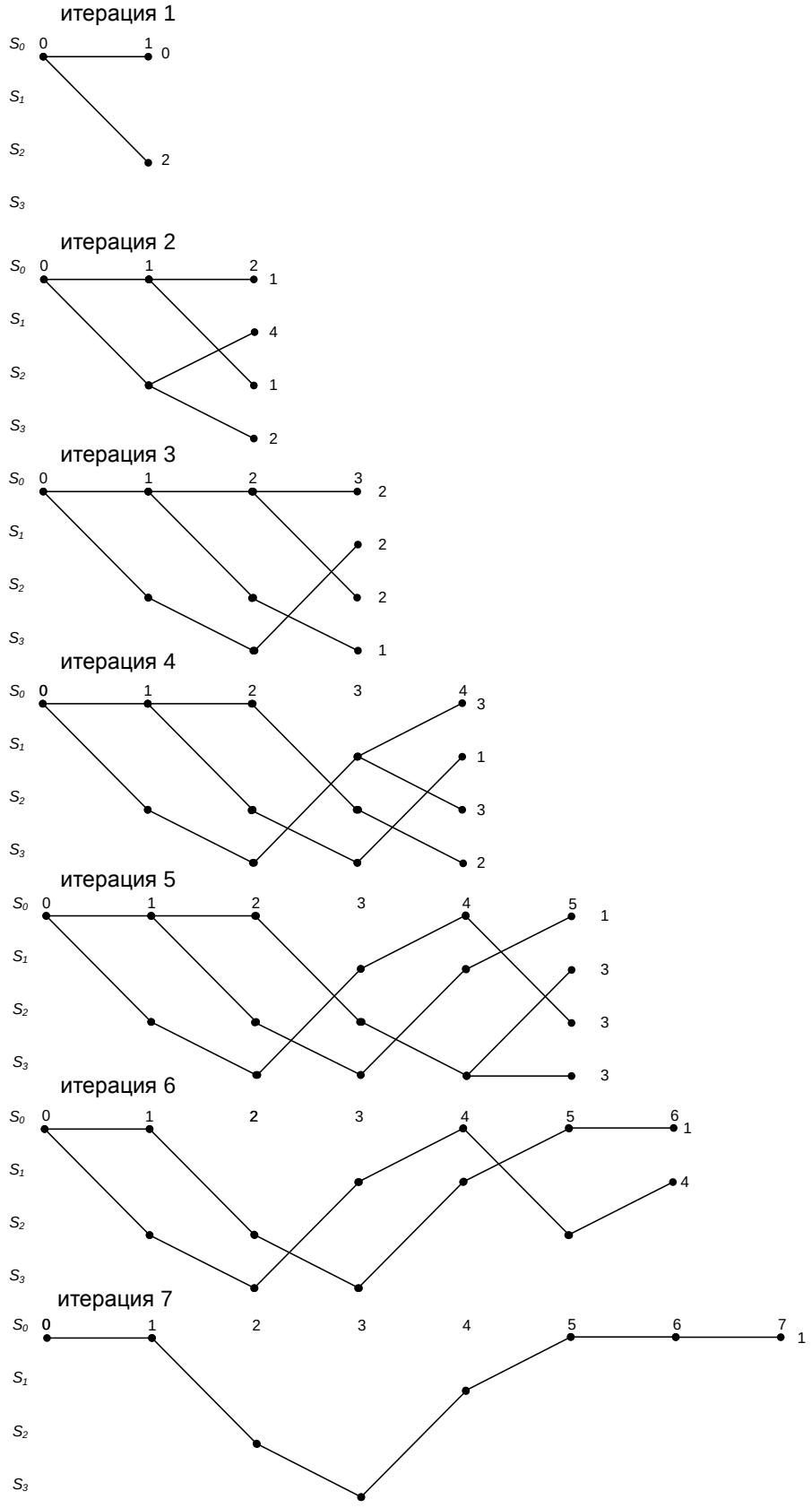


Рис. 33. Пример декодирования с помощью алгоритма Витерби

Алгоритм декодирования Витерби с мягкими решениями

Декодирование, где помимо оценки бит, передается информация о надежности, называется декодированием с мягкими решениями. Для примера декодирования с мягкими решениями рассмотрим двоичный канал с аддитивным белым гауссовским шумом **n**

$$\mathbf{r} = \mathbf{x} + \mathbf{n}, \tag{5.9.4}$$

где **r** последовательность, полученная на приемнике. Для случая двоичной модуляции запишем соответствие между кодовым битом $c_i^{(j)}$ и выходом двоичного фазового модулятора $x_i^{(j)}$

$$x_i^{(j)} = \begin{cases} +1, & c_i^{(j)} = 1 \\ -1, & c_i^{(j)} = 0 \end{cases}. \tag{5.9.5}$$

Несложно записать выражение для условной плотности вероятности $r_i^{(j)}$ при заданном символе $x_i^{(j)}$

$$p\left(r_i^{(j)} \mid x_i^{(j)}\right) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left\{-\frac{\left(r_i^{(j)} - x_i^{(j)}\right)^2}{2\sigma^2}\right\}. \tag{5.9.6}$$

Тогда метрика ребра определяется как квадрат расстояния между принятыми и кодовыми символами

$$B(\mathbf{r}_i, \mathbf{x}_i) = \sum_{j=0}^{n-1} \frac{\left(r_i^{(j)} - x_i^{(j)}\right)^2}{2\sigma^2}, \tag{5.9.7}$$

Алгоритм декодирования на основе Евклидова расстояния находит кодовую последовательность ближайшую к принятой последовательности. Не сложно показать, что произведение $r_i^{(j)} x_i^{(j)}$ можно также использовать в качестве метрики.

5.10. ИНТЕРЛИВИНГ

Интерливинг является одним из часто используемых способов повышения помехоустойчивости практических систем связи. Как правило, реальные каналы связи не

являются каналами без памяти, т.е. возникающие ошибки не являются случайными, а группируются в пакеты ошибок. Если число ошибок на длине кодового слова превышает исправляющую способность кода, то декодер не сможет правильно восстановить переданную последовательность. Процедура интерливинга позволяет решить проблему группировки ошибок путем перестановки кодовых символов на длине превышающей длину кодового слова. В результате на приемнике, после обратной перестановки (де-интерливинга), ошибки приходящие на схему декодирования становятся случайными.

Схемы интерливинга можно условно поделить на две группы: случайные и структурированные. Рассмотрим два примера структурированных перестановок. Первым способом перестановки является блоковый интерливинг. Для блокового интерливера кодовые биты перед модулятором записываются в матрицу размером $N_r \times N_c$ по столбцам, и считываются по строкам. Процедура блокового интерливинга показана на Рис. 34.

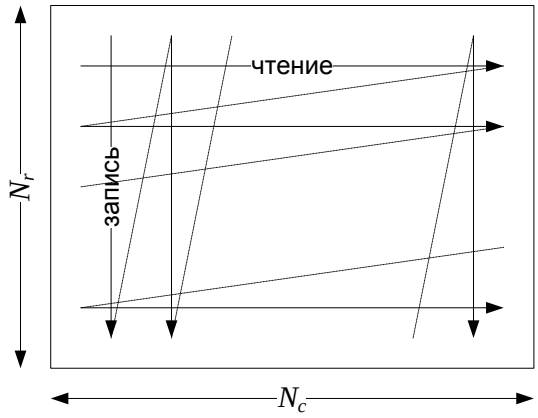


Рис. 34. Блоковый интерливер

Математически индексы перестановки для блокового интерливера могут быть получены с помощью выражения

$$i = N_c \cdot \mathrm{mod}(k, N_r) + \left\lfloor \frac{k}{N_r} \right\rfloor, \qquad (5.10.1)$$

где i индекс бита на выходе интерливинга, k индекс бита на входе интерливинга, $\lfloor \cdot \rfloor$ операция округления к меньшему целому числу. Процедура де-интерливинга на приемнике может быть осуществлена в обратном порядке – записью принятых бит по строкам и считыванием по столбцам.

Вторым способом перестановки является сверточный интерливинг. Структура такого интерливинга показана на Рис. 35

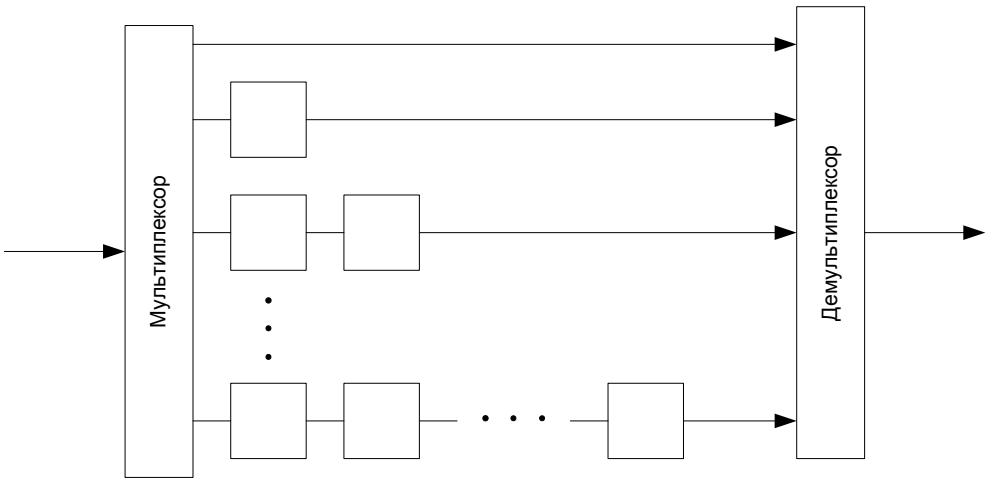


Рис. 35. Сверточный интерливер

Кодовая последовательность поступает на вход мультиплексора, разделяющего входную последовательность на B параллельных потока, которые, в свою очередь, поступают на линии задержки. При этом каждая линия задержки i содержит $i-1$ регистров сдвига. Общее число элементов памяти сверточного интерливера равно $\frac{B\cdot(B-1)}{2}$. Процедура сверточного де-интерливинга может быть выполнена с помощью аналогичной схемы, имеющей $B-i-1$ регистра памяти для линии i.

Использование битового интерливинга и алгоритма декодирования с мягкими решениями не позволяет применять стандартные подходы для вычисления метрик. В качестве метрики декодирования в таких схемах необходимо использовать метрику на бит. В качестве такой метрики, применяется логарифм отношения вероятностей, задаваемый следующим равенством

$$\mathrm{LLR}(c_i)=\log\left\{\frac{P(c_i=1|r_i)}{P(c_i=0|r_i)}\right\}=\log\left\{\frac{P(r_i|c_i=1)}{P(r_i|c_i=0)}\right\}. \qquad (5.10.2)$$

Знак выражения (5.10.2) определяет «жесткое» решение о принятом бите, а модуль надежность такого решения. Можно показать, что оптимальный путь найденный с помощью метрики (5.10.2) совпадает с путем, найденным с помощью метрики (5.8.5). Для доказательства предположим, что это не так. Пусть $\mathbf{c}'$ и $\mathbf{c}''$ является путями,

обеспечивающими максимум суммы метрик (5.8.5) $M^{(1)}(\cdot)$ и (5.10.2) $M^{(2)}(\cdot)$ соответственно. Заметим, что разница метрик для бита 1 и 0 одинакова для способа (5.9.7) и (5.10.2) с точностью до постоянного множителя, которым можно пренебречь, тогда

$$M^{(1)}(\mathbf{c}'')-M^{(1)}(\mathbf{c}')=M^{(2)}(\mathbf{c}'')-M^{(2)}(\mathbf{c}'). \qquad (5.10.3)$$

Поскольку путь $\mathbf{c}'$ является оптимальным для метрики (5.8.5), то $M^{(1)}(\mathbf{c}'')-M^{(1)}(\mathbf{c}')<0$. Используя равенство (5.10.3) следует $M^{(2)}(\mathbf{c}'')-M^{(2)}(\mathbf{c}')<0$, что противоречит предположению, что $\mathbf{c}''$ является оптимальным для метрики $M^{(2)}(\cdot)$.

Вернемся к вычислению выражения (5.10.2). Используя правило Байеса, логарифм отношения вероятностей (5.10.2) может быть записан в следующем виде

$$\mathrm{LLR}(c_i)=\log\left\{\frac{\sum\limits_{x_i\in X^{(1)}}P(r_i\mid x_i)}{\sum\limits_{x_i\in X^{(0)}}P(r_i\mid x_i)}\right\}, \qquad (5.10.4)$$

где суммирование в числителе и знаменателе ведется по всем сигнальным точкам созвездия соответствующим биту $c_i=1$ (множество $X^{(1)}$) или $c_i=0$ (множество $X^{(0)}$) соответственно. Пример разделения точек 16-QAM сигнального созвездия на два множества $X^{(1)}$ и $X^{(0)}$ для первого бита, приведен на Рис. 36.

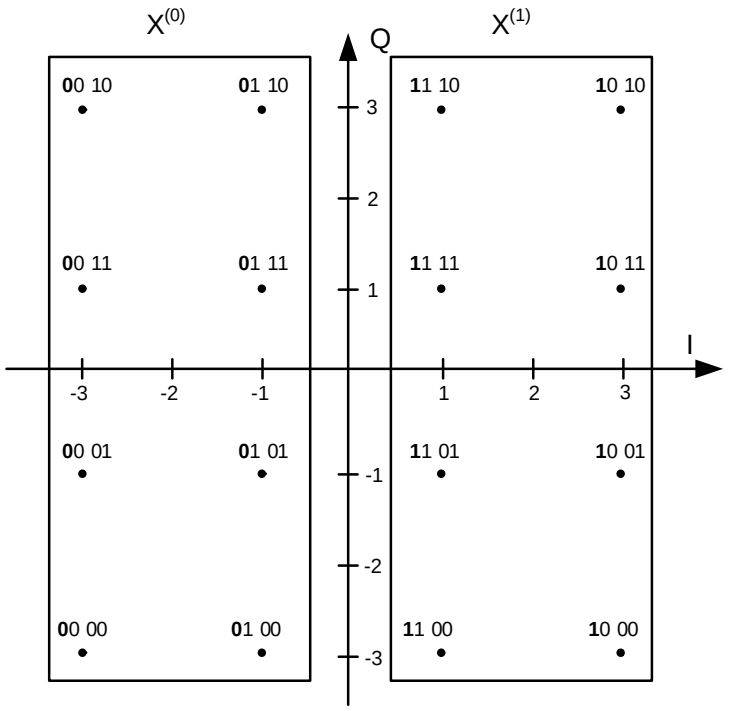


Рис. 36. Разделение точек 16-QAM сигнального созвездия на множества $X^{(1)}$ и $X^{(0)}$

Используя аппроксимацию $\log\left\{\sum_i a_i\right\} \approx \max(\log(a_i))$, получим

$$\mathrm{LLR}(c_i) = \max_{X^{(1)}}\{\log P(r_i \mid x_i)\} - \max_{X^{(0)}}\{\log P(r_i \mid x_i)\}. \qquad (5.10.5)$$

Для канала с аддитивным белым гауссовским шумом

$$\mathrm{LLR}(c_i) = \frac{1}{2\sigma^2}\left\{\min_{X^{(0)}}|r_i - x_i|^2 - \min_{X^{(1)}}|r_i - x_i|^2\right\}. \qquad (5.10.6)$$

Выражение (5.10.6) задает общий подход к расчету метрик. Дальнейшие упрощения возможны при рассмотрении конкретных примеров сигнальных созвездий. Так для созвездия 16-QAM, логарифм отношения правдоподобия может быть вычислен с помощью

$$\mathrm{LLR}(c_i) = \begin{cases} \mathrm{Re}(r_i), & |\mathrm{Re}(r_i)| \le 2 \\ 2\cdot(\mathrm{Re}(r_i)-1), & \mathrm{Re}(r_i) > 2 \\ 2\cdot(\mathrm{Re}(r_i)+1), & \mathrm{Re}(r_i) < -2 \end{cases}, \qquad (5.10.7)$$

для первого бита и с помощью

$$\mathrm{LLR}(c_i) = -2|\mathrm{Re}(r_i)| + 1, \qquad (5.10.8)$$

для второго. Заменяя операцию взятия действительной части $\mathrm{Re}(\cdot)$ на операцию взятия мнимой части $\mathrm{Im}(\cdot)$, аналогичным способом могут быть вычислены метрики для третьего и

четвертого бита. График отображения принятого символа в метрику бита $\mathrm{LLR}(c_i)$ для 16-QAM модуляции показана на Рис. 37.

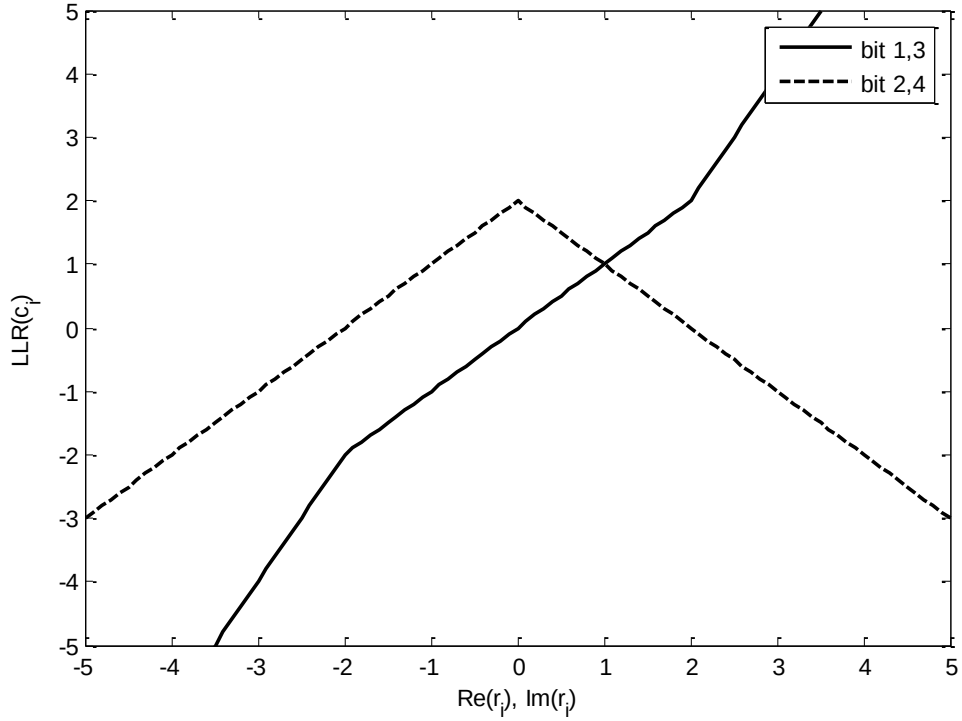


Рис. 37. Функция отображения $\mathrm{LLR}(c_i)$ для 16-QAM модуляции

Глава 6

КОДЫ С МАЛОЙ ПЛОТНОСТЬЮ ПРОВЕРКИ НА ЧЕТНОСТЬ

6.1. ОПРЕДЕЛЕНИЕ КОДОВ С МАЛОЙ ПЛОТНОСТЬЮ ПРОВЕРКИ НА ЧТЕНОСТЬ

Коды с малой плотностью проверки на четность являются подклассом линейных блоковых кодов, рассмотренных в Главе 2. Напомним, что (n,k) линейный блоковый код можно рассматривать как отображение (с помощью порождающей матрицы $\mathbf{G}$) k-мерного подпространства в пространство размерности n. При этом процедура кодирования может быть представлена в матричном виде

$$\mathbf{c} = \mathbf{u} \cdot \mathbf{G}. \tag{6.1.1}$$

Строки порождающей матрицы $\mathbf{G}$ задают базис кодового подпространства размерности k, а базисные вектора ортогональные кодовому подпространству задают проверочную матрицу $\mathbf{H}$ размерности $(n-k,n)$. В силу ортогональности подпространств, для любого кодового слова $\mathbf{c}$ выполняется равенство

$$\mathbf{c} \cdot \mathbf{H}^T = \mathbf{0}, \tag{6.1.2}$$

Отличительной особенностью кодов с малой плотностью проверки на четность является разреженность проверочной матрицы $\mathbf{H}$, т.е. число ненулевых элементов $O(n)$, что является незначительным по отношению к общему числу элементов матрицы $\mathbf{H}$. Данное ограничение позволяет существенно упростить процедуру декодирования кодов, которую мы рассмотрим ниже.

Несмотря на то, что эти коды были изобретены в 1963 году Робертом Галлагером, их практическое применение началось относительно недавно.

Далее мы ограничимся рассмотрением двоичных кодов с малой плотностью проверки на четность, заданных над полем $GF(2)$. Введем некоторые определения. Код с малой плотностью проверки на четность является кодом регулярности t, если вес (число единиц) каждого столбца проверочной матрицы $\mathbf{H}$ является постоянной величиной. В этом случае среднее число единиц в каждой строке проверочной матрицы $\mathbf{H}$ равно $nt/(n-k)$. Если вес

каждой строки проверочной матрицы также является постоянной величиной равной $s = nt/(n-k)$, то код называется (s,t) регулярным.

Пример кода с малой плотностью проверки на плотность регулярности $t = 2$ приведен ниже

$$\mathbf{H} = \begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix}$$

Напомним, что для любого кодового слова $\mathbf{c}$ должно выполняться равенство

$$\begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ \vdots \\ c_8 \\ c_9 \end{bmatrix} = 0.$$

Выражение (6.1.2) определяет набор уравнений проверок на четность, которые для рассматриваемого примера можно записать следующим образом

$$\begin{aligned} p_0 &= c_0 + c_1 + c_2 + c_3 \\ p_1 &= c_0 + c_4 + c_5 + c_6 \\ p_2 &= c_1 + c_4 + c_7 + c_8 \\ p_3 &= c_2 + c_5 + c_7 + c_9 \\ p_4 &= c_3 + c_6 + c_8 + c_9 \end{aligned}$$

Коды с малой плотностью проверки на четность описываются с помощью двухстороннего (двудольного) графа, где нижние (битовые) узлы соответствуют кодовым битам c_i, а верхние (проверочные) узлы соответствуют проверочным уравнениям p_j. Битовый узел соединяется с проверочным узлом ребром, если кодовый бит присутствует в уравнении проверки на четность (6.1.2). Пример двустороннего графа для рассматриваемого кода приведен на Рис. 38.

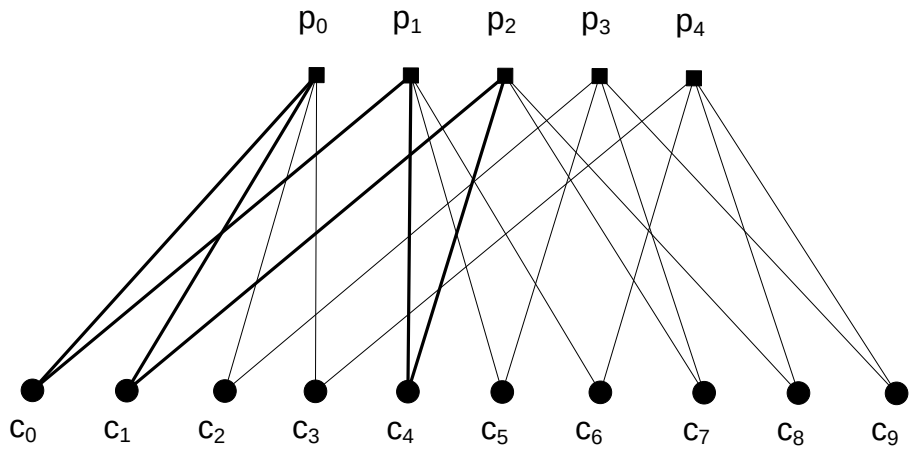


Рис. 38. Двусторонний граф кода с малой плотностью проверки на четность

Циклом длины ν в двустороннем графе будем называть замкнутый путь, состоящий из ν ребер. На Рис. 38 жирной линией выделен цикл длины 6. Цикл минимальной длины называется *обхватом кода*.

6.2. ПРИМЕРЫ ПОСТРОЕНИЯ КОДОВ С МАЛОЙ ПЛОТНОСТЬЮ ПРОВЕРКИ НА ЧТЕНОСТЬ

Коды Галлагера

Коды Галлагера является регулярными кодами с малой плотностью проверки на четность. Проверочная матрица таких кодов может быть представлена в виде

$$\mathbf{H} = \begin{bmatrix} \mathbf{H}_1 \\ \mathbf{H}_2 \\ \vdots \\ \mathbf{H}_t \end{bmatrix}, \qquad (6.2.1)$$

где подматрицы $\mathbf{H}_d$ имеют следующую структуру. Для любого целого μ и s больше единицы, подматрица $\mathbf{H}_d$ имеет размерность $\mu \times \mu \cdot s$, а вес каждой строки и столбца равен s и 1 соответственно. Первая подматрица $\mathbf{H}_1$ имеет специальный вид, такой, что строка i для $i = 1,...,\mu$ содержит все s единиц в столбцах с индексами от $(i-1)\cdot s+1$ до $i \cdot s$. Остальные подматрицы $\mathbf{H}_d$ где $d = 2,...,t$ могут быть получены путем перестановки столбцов матрицы $\mathbf{H}_1$. Очевидно, что код Галлагера является регулярным, а матрица $\mathbf{H}$

имеет размерность $\mu \cdot t \times \mu \cdot s$. Отсутствие циклов длины 4 не гарантируется. Однако их можно избежать путем соответствующего выбора матрицы $\mathbf{H}$.

Коды МакКея

Построение кодов МакКея основывается на случайном поиске матриц. Ниже приведены основные способы построения кода, предложенные МакКеем:

1. Матрица $\mathbf{H}$ создается путем набора векторов весом s
2. Матрица $\mathbf{H}$ создается из набора векторов весом s, гарантирующего вес t для каждой строки матрицы. При этом любая пара векторов имеет расстояние Хэмминга не более чем 1.
3. Из множества матриц $\mathbf{H}$, удовлетворяющих построению 2, исключаются матрицы, имеющие короткие циклы в двустороннем графе.
4. Матрица $\mathbf{H}$ удовлетворяет построению 3. При этом $\mathbf{H}$ можно представить как $\mathbf{H} = [\mathbf{H}_1 \quad \mathbf{H}_2]$, где матрица $\mathbf{H}_2$ обратима.

6.3. МЕТОДЫ ДЕКОДИРОВАНИЯ КОДОВ С МАЛОЙ ПЛОТНОСТЬЮ ПРОВЕРКИ НА ЧТЕНОСТЬ

Пусть вектор $\mathbf{r} = (r_0, r_1, \ldots, r_{n-1})$ длины n задает принятую последовательность, а вектор $\mathbf{c} = (c_0, c_1, \ldots, c_{n-1})$ кодовую. Для простоты рассмотрения будем читать что -1 и +1 соответствуют 0 и 1 соответственно. Тогда кодовая последовательность c_k может принимать значения -1 или +1 при передаче в канале. Задача декодирования ставится как оценка элементов кодовой последовательности c_k на основе наблюдения $\mathbf{r}$. Декодирование кодовых бит c_k будем осуществлять путем вычисления логарифма отношения апостериорной вероятности на основе наблюдения $\mathbf{r}$

$$L(c_k \mid \mathbf{r}) = \log\left(\frac{P(c_k = 0 \mid \mathbf{r})}{P(c_k = 1 \mid \mathbf{r})}\right). \quad (6.3.1)$$

Если логарифм отношения апостериорной вероятности $L(c_k \mid \mathbf{r}) > 0$, то $\hat{c}_k = 0$. В противном случае $\hat{c}_k = 1$. Напомним, что все n кодовых бит при декодировании должны удовлетворять равенствам проверки на четность. Данное структурное ограничение кода должно также учитывается при декодировании. Тогда выражение для апостериорной вероятности модифицируется следующим образом

$$P(c_k \mid S_k, \mathbf{r}), \quad (6.3.2)$$

где событие S_k соответствует выполнению всех уравнений проверки на четность для бита c_k. Используя правило Байеса, апостериорная вероятность $P(c_k \mid S_k, \mathbf{r})$ может быть преобразована к следующему виду

$$P(c_k \mid S_k, \mathbf{r}) = K \cdot P(r_k \mid c_k) \cdot P(S_k \mid c_k, \mathbf{r}). \quad (6.3.3)$$

Первый множитель K в равенстве (6.3.3) является константой и в дальнейшем может быть опущен из рассмотрения, т.к. является величиной независящей от c_k. Второй множитель $p(r_k \mid c_k)$ выражения (6.3.3) может быть вычислен из наблюдения r_k с использованием модели канала. Например, для двоичного канала с аддитивным белым гауссовским шумом вероятность $p(r_k \mid c_k)$ определяется как

$$p(r_k \mid c_k) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left\{-\frac{(r_k - c_k)^2}{2\sigma^2}\right\}. \quad (6.3.4)$$

Третий множитель в равенстве (6.3.3) задает вероятность выполнения всех проверок на четность с участием кодового бита c_k. Событие S_k может быть представлено как объединение событий $\{S_{1k}, S_{2k}, ..., S_{Lk}\}$, где S_{mk} является событием, соответствующим выполнению проверочного выражения для проверочного узла m, соединенного с битовом узлом k. Пусть $M(k)$ является набором проверочных узлов соответствующих биту c_k, а $N(m)$ множеством битовых узлов соответствующих проверочному узлу m. Если в двустороннем графе отсутствуют замкнутые циклы, то третий множитель может быть представлен в виде произведения

$$P(S_k \mid c_k = b, \mathbf{r}) = p(S_{1k}, S_{2k}, ..., S_{Lk} \mid c_k = b, \mathbf{r}) = \prod_{m \in M(k)} P(S_{mk} \mid c_k, \mathbf{r}), \qquad (6.3.5)$$

где $p(S_{mk} \mid c_k, \mathbf{r})$ вероятность, что проверка соответствующая проверочному узлу m, соединенная с битом c_k, выполняется. Стоит отметить, что условие независимости кодовых битов выполняется, если в двустороннем графе отсутствуют замкнутые циклы. Если $c_k = 0$, то $p(S_{mk} \mid c_k, \mathbf{r})$ равняется вероятности того, что все битовые узлы, соединяющие проверочный узел m за исключением узла соответствующего биту c_k, имеют четное число единиц. Аналогично, если $c_k = 1$, то для выполнения условия проверки на четность в проверочном узле m, число единиц в оставшихся битовых узлах нечетно.

Далее предположим, что два бита удовлетворяют уравнению проверки на четность (т.е. $c_1 + c_2 = 0$), а вероятности равны $P(c_1 = 0) = p_1 = 1 - q_1$, $P(c_2 = 0) = p_2 = 1 - q_2$. Тогда вероятность выполнения проверки на четность для двух бит

$$P(c_1 + c_2 = 0) = (1 - p_1) \cdot (1 - p_2) + p_1 \cdot p_2 = 2 \cdot p_1 \cdot p_2 + 1 - p_1 - p_2. \qquad (6.3.6)$$

Преобразуем выражение (6.3.6) в следующий вид

$$2 \cdot P(c_1 + c_2 = 0) - 1 = (1 - 2 \cdot p_1) \cdot (1 - 2 \cdot p_2) = (q_1 - p_1) \cdot (q_2 - p_2). \qquad (6.3.7)$$

Обобщим выражение (6.3.7) на случай L кодовых бит, участвующих в уравнении проверки на четность $z_L = c_1 + c_2 + ... + c_L$

$$P(z_L = 0) = \frac{1}{2}\left(1 + \prod_{i=1}^{L}(q_i - p_i)\right), \qquad (6.3.8)$$

где $p_i = 1 - q_i = P(c_i = 0)$. Аналогичным образом можно показать, что

$$P(z_L = 1) = \frac{1}{2}\left(1 - \prod_{i=1}^{L}(q_i - p_i)\right). \qquad (6.3.9)$$

Тогда вероятность $P(S_{mk} \mid c_k, \mathbf{r})$ при условии $c_k = 0$ равна

$$P(S_{mk} \mid c_k = 0, \mathbf{r}) = \frac{1}{2}\left(1 + \prod_{n' \in N(m) \setminus k}\left(q_{mn'}(0) - q_{mn'}(1)\right)\right), \qquad (6.3.10)$$

где $q_{mn'}(0)$ вероятность, что кодовый бит $c_{n'}$ равен 0, $N(m) \setminus k$ множество кодовых узлов за исключением узла k. Заметим, что равенство (6.3.10) включает в себя произведение по всем ребрам, соединяющими проверочный узел m кодовыми, за исключением узла k. Таким

образом, при вычислении $P(S_{mk} \mid c_k, \mathbf{r})$ исключается информация о бите c_k, а полученная информация является *внешней*, вычисленной с использованием структуры кода.

Объедения, полученные выше выражения

$$P(c_k = 0 \mid S_k, r) = K \cdot P(r_k \mid c_k = 0) \cdot \prod_{m \in M(k)} \frac{1}{2}\left(1 + \prod_{n' \in N(m) \setminus k} \left(q_{mn'}(0) - q_{mn'}(1)\right)\right). \quad (6.3.11)$$

$$P(c_k = 1 \mid S_k, r) = K \cdot P(r_k \mid c_k = 1) \cdot \prod_{m \in M(k)} \frac{1}{2}\left(1 - \prod_{n' \in N(m) \setminus k} \left(q_{mn'}(0) - q_{mn'}(1)\right)\right). \quad (6.3.12)$$

Равенства (6.3.11) и (6.3.12) можно рассматривать как последовательное вычисление сообщений от проверочных узлов к битовым узлам и наоборот. Например, для $c_k = 1$ вычисление выражения (6.3.12) можно разбить на два этапа

$$\underbrace{\overbrace{P(r_k \mid c_k = 1)}^{\text{априорная вероятность на основе наблюдений } r_k} \cdot \prod_{m \in M(k)} \overbrace{\frac{1}{2}\left(1 - \prod_{n' \in N(m) \setminus k} (q_{mn'}(0) - q_{mn'}(1))\right)}^{\text{проверочный узел}}}_{\text{битовый узел}} \quad (6.3.13)$$

Первый этап соответствует вычислению сообщений от проверочных узлов к битовым узлам. Вводя обозначение $\delta q_{mn'} = q_{mn'}(0) - q_{mn'}(1)$, выражения, соответствующие первому этапу, задаются следующим образом

$$r_{mk}(0) = \frac{1}{2}\left(1 + \prod_{n' \in N(m) \setminus k} \delta q_{mn'}\right) \quad (6.3.14)$$

$$r_{mk}(1) = \frac{1}{2}\left(1 - \prod_{n' \in N(m) \setminus k} \delta q_{mn'}\right). \quad (6.3.15)$$

Графически вычисление $r_{mk}(\cdot)$ проиллюстрировано на Рис. 39.

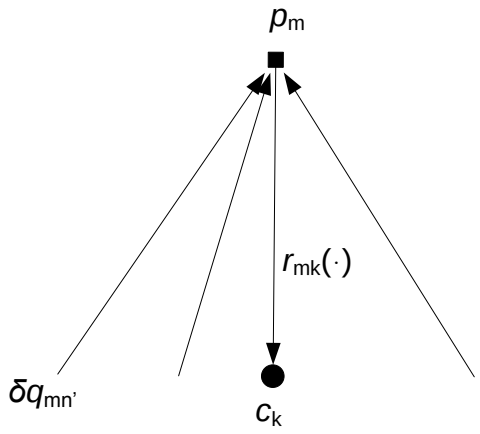


Рис. 39. Вычисление $r_{mk}(\cdot)$

Для вычисления сообщения $r_{mk}(\cdot)$ находится набор ребер $N(m)\setminus k$ двухстороннего графа, входящих в узел p_m, при этом исключается ребро, соединяющее битовый узел c_k с проверочным узлом p_m. Далее согласно выражениям (6.3.14) и (6.13.15) производится вычисление сообщения $r_{mk}(\cdot)$ от проверочного узла p_m к битовому узлу c_k.

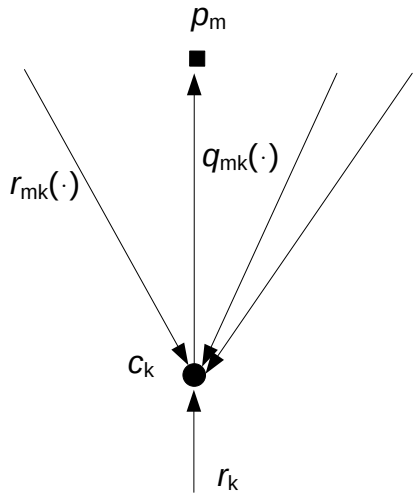


Рис. 40 Вычисление $q_{mk}(\cdot)$

Аналогичным способом выполняется второй этап вычисления выражений (6.3.11) и (6.3.12), соответствующий вычислению сообщений от битовых узлов к проверочным (см. Рис. 40). При этом учитывается априорная информация $P(r_k \mid c_k)$ кодового бита c_k, полученная на основе принятого символа r_k.

Псевдокод алгоритма декодирования кодов с малой плотностью проверки на четность приведен ниже:

Инициализация

Переменная $q_{mn}(\cdot)$ инициализируются следующим образом

$$q_{mk}(\cdot) = f_k(\cdot) = \begin{cases} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(r_k+1)^2}{2\sigma^2}\right), & c_k = 0 \\ \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(r_k-1)^2}{2\sigma^2}\right), & c_k = 1 \end{cases} \quad (6.3.16)$$

Вычисление сообщений проверочных узлов

Для каждого ребра исходящего из проверочного узла с индексом $m \in M(k)$ вычисляется сообщение

$$\delta r_{mk} = \prod_{i' \in N(m) \setminus k} \delta q_{mi'}, \quad (6.3.17)$$

где $r_{mk}(0) = \frac{1}{2}\left(1 + \delta r_{mk}\right)$, $r_{mk}(1) = \frac{1}{2}\left(1 - \delta r_{mk}\right)$.

Вычисление сообщений битовых узлов

Для каждого ребра, исходящего из битового узла с индексом $k = 0,...,n-1$, вычисляется *внешнее* сообщение, исключающее информацию проверочного узла m

$$q_{mk}(0) = \alpha_{mk} f_k(0) \prod_{m' \in M(k) \setminus m} r_{m'k}(0) \quad (6.3.18)$$

$$q_{mk}(1) = \beta_{mk} f_k(1) \prod_{m' \in M(k) \setminus m} r_{m'k}(1) \quad (6.3.19)$$

Постоянные нормировки α_{mk} и β_{mk} выбираются таким образом, чтобы выполнялось условие нормировки вероятности $q_{mk}(0) + q_{mk}(1) = 1$.

Вычисление выражений проверки на четность

Для каждого элемента кодовой последовательности с индексом $k = 0,...,n-1$ вычисляется *полная* вероятность

$$q_k(0) = \alpha_k f_k(0) \prod_{m' \in M(k)} r_{m'k}(0) \tag{6.3.20}$$

$$q_k(1) = \beta_k f_k(1) \prod_{m' \in M(k)} r_{m'k}(1). \tag{6.3.21}$$

Постоянные нормировки α_k и β_k выбираются таким образом, чтобы выполнялось условие нормировки вероятности $q_k(0) + q_k(1) = 1$. На основе вероятностей кодовых узлов вычисляются решения

$$c_k = \begin{cases} 0, & q_k(0) > 1/2 \\ 1, & q_k(1) \le 1/2 \end{cases} \tag{6.3.22}$$

Если проверка на четность $\mathbf{c} \cdot \mathbf{H}^T = \mathbf{0}$ не выполняется, и число итераций не превысило максимального значения, процедура декодирования повторяется.

Заметим, что при вычислении сообщений для ребер исходящих из битовых и проверочных узлов исключается информация соответствующего ребра. Таким образом, на каждой итерации вычисляется только *внешняя информация*, которая передается далее по графу. При этом для вычисления выражений проверки на четность учитывается *полная* информация. Можно показать, что если в графе отсутствуют замкнутые циклы, то приведенный выше алгоритм осуществляет оценку, обеспечивающую максимум апостериорной вероятности для каждого кодового бита. Однако для двухстороннего графа с обхватом γ информация возвращается к началу цикла на итерации γ и алгоритм в общем случае не является оптимальным. Не смотря на это, результаты моделирования практических схем показали, что эффективность алгоритма декодирования кодов остается достаточно высокой при использовании кодов с обхватом 6 или более.

Основным недостатком полученного итеративного алгоритма декодирования является вычислительная неустойчивость, связанная с наличием большого числа умножений. На практике предпочтительным является реализация алгоритма декодирования на основе операций сложения. Для этого декодирование необходимо проводить в области логарифма вероятности. Тогда вычислительно-емкие операции умножения и деления заменяются относительно простыми операциями сложения и деления. Для упрощения процедуры декодирования будем использовать следующее выражение для произведения множителей

$$\prod_i a_i = \left(\prod_i sign(a_i) \right) \cdot \exp\left(\sum_i \log\left(|a_i| \right) \right), \tag{6.3.23}$$

Введем функцию

$$\tanh\left(\frac{x}{2} \right) = \frac{\exp(x)-1}{\exp(x)+1}, \tag{6.3.24}$$

и получим равенство для логарифма отношения апостериорной вероятности

$$L(c_k \mid \mathbf{r}) = \log\left(\frac{p(r_k \mid c_k = 0)}{p(r_k \mid c_k = 1)} \right) + \log \prod_{m \in M(k)} \frac{1 + \prod_{n' \in N(m) \setminus k} \left(q_{mn'}(0) - q_{mn'}(1) \right)}{1 - \prod_{n' \in N(m) \setminus k} \left(q_{mn'}(0) - q_{mn'}(1) \right)}. \tag{6.3.25}$$

Используя определения $\delta q_{mn'} = q_{mn'}(0) - q_{mn'}(1)$ и $\mathrm{LLR}\left(q_{mn'} \right) = \log\left(\frac{q_{mn'}(0)}{q_{mn'}(1)} \right)$, простая подстановка дает $\delta q_{mn'} = \tanh\left(\frac{\mathrm{LLR}\left(q_{mn'} \right)}{2} \right)$. Тогда равенство (6.3.25) можно представить в следующем виде

$$L(c_k \mid \mathbf{r}) = 2\frac{r_k}{\sigma^2} + \sum_{m \in M(k)} \log \frac{1 + s_{mk} \exp(A_{mk})}{1 - s_{mk} \exp(A_{mk})}, \tag{6.3.26}$$

где коэффициенты s_{mk} и A_{mk} определяются как

$$s_{mk} = \prod_{n' \in N(m) \setminus k} sign(\delta q_{mn'}) = \prod_{n' \in N(m) \setminus k} sign\left(\mathrm{LLR}\left(q_{mn'} \right) \right), \tag{6.3.27}$$

$$A_{mk} = \sum_{n' \in N(m) \setminus k} \log\left(\left| \tanh\left(\frac{\mathrm{LLR}\left(q_{mn'} \right)}{2} \right) \right| \right). \tag{6.3.28}$$

Легко видеть, что коэффициент s_{mk} содержит информацию о знаке сообщения, т.е. жестком решении, а коэффициент A_{mk} об абсолютном значении сообщения, т.е. надежности этого решения.

Равенство (6.3.26) можно упростить

$$L(c_k \mid \mathbf{r}) = 2\frac{r_k}{\sigma^2} - \sum_{m \in M(k)} s_{mk} \log\left(-\tanh\left(\frac{A_{mk}}{2} \right) \right). \tag{6.3.29}$$

Вводя обозначение $\psi(x) = \log\left(\left| \tanh\left(\frac{x}{2} \right) \right| \right)$, получим

$$L(c_k \mid \mathbf{r}) = 2\frac{r_k}{\sigma^2} - \sum_{m \in M(k)} s_{mk}\psi(A_{mk}), \tag{6.3.30}$$

где $A_{mk} = \sum_{n'=N(m)\setminus k} \log\left(\left|\tanh\left(\frac{\mathrm{LLR}(q_{mn'})}{2}\right)\right|\right) = \sum_{n'=N(m)\setminus k}\psi\left(\mathrm{LLR}(q_{mn'})\right)$. График функции $\psi(x)$ представлен на Рис. 41.

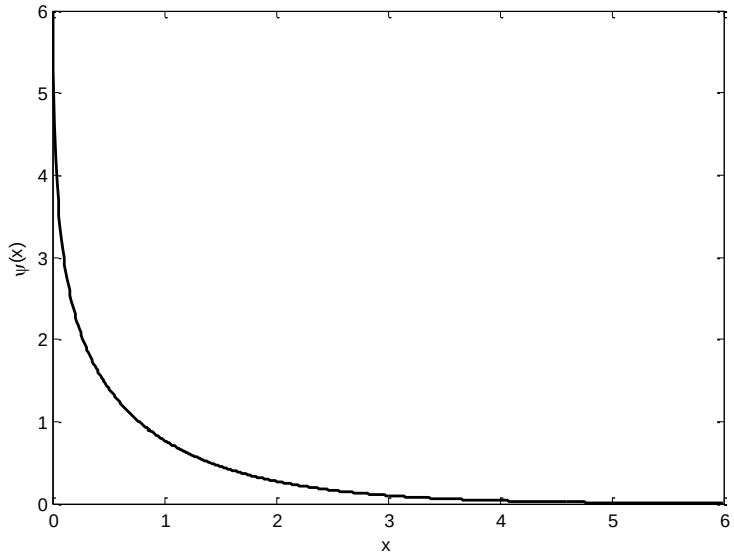


Рис. 41. График функции $\psi(x)$ для $x > 0$

Отметим, что функция $\psi(x)$ является сама себе обратной, т.е. $\psi^{-1}(x) = \psi(x)$ для $x > 0$.

Псевдокод алгоритма декодирования кодов с малой плотностью проверки на четность в области логарифма вероятности можно записать следующим образом:

Инициализация

Переменная $\mathrm{LLR}(q_{mk})$ инициализируются следующим образом

$$\mathrm{LLR}(q_{mk}) = \log\left(\frac{q_{mk}(0)}{q_{mk}(1)}\right) = \frac{2r_k}{\sigma^2} \tag{6.3.31}$$

Вычисление сообщений проверочных узлов

Для каждого ребра исходящего из проверочного узла с индексом $m \in M(k)$ вычисляется сообщение $R_{mk} = s_{mk}\psi(A_{mk})$, где $A_{mk} = \sum_{n' \in N(m)\setminus k}\psi\left(\mathrm{LLR}(q_{mn'})\right)$.

Вычисление сообщений битовых узлов

Для каждого ребра исходящего из битового узла с индексом $k = 0,...,n-1$ вычисляется сообщение, $\mathrm{LLR}\left(q_{mk}\right) = \mathrm{LLR}\left(q_k\right) - R_{mk}$, где $\mathrm{LLR}\left(q_k\right) = \sum_{m \in M(k)} R_{mk}$.

Вычисление выражений проверки на четность

Для каждого элемента кодовой последовательности с индексом $k = 0,...,n-1$ с помощью $\mathrm{LLR}\left(q_k\right)$ вычисляется жесткое решение

$$c_k = \begin{cases} 0, \mathrm{LLR}\left(q_k\right) > 0 \\ 1, \mathrm{LLR}\left(q_k\right) \le 0 \end{cases}. \qquad (6.3.32)$$

Если проверка на четность $\mathbf{c} \cdot \mathbf{H}^T = \mathbf{0}$ не выполняется, и число итераций не превысило максимального значения, процедура декодирования повторяется.

Заметим, что алгоритм на основе логарифма вероятности является вычислительно устойчивым по сравнению с алгоритмом декодирования в области вероятности. Стоит также отметить, что для декодирования принятой последовательности требуется информация о мощности аддитивного шума σ^2.

Упрощение алгоритма декодирования кодов с малой плотностью проверки на четность возможно при использовании различных аппроксимаций функции $\psi(\cdot)$. Например, используя простейшую аппроксимацию

$$\psi\left(\sum_i \psi(x_i)\right) \approx \min_i x_i \qquad (6.3.33)$$

можно получить следующее упрощение выражения (6.3.30)

$$L(c_k \mid \mathbf{r}) = 2\frac{r_k}{\sigma^2} - \sum_{m \in M(k)} s_{mk} \min_{n' \in N(m) \setminus k} \mathrm{LLR}\left(q_{mn'}\right). \qquad (6.3.34)$$

В этом случае сообщение, исходящее из проверочного соответствует входящему ребру с минимальным значением $\mathrm{LLR}\left(q_{mn'}\right)$. Поскольку вычисление функции $\min(\cdot)$ можно проводить с точностью до постоянного множителя, постоянный множитель мощности шума σ^2 можно опустить при инициализации $\mathrm{LLR}\left(q_{mk}\right)$, что дает дополнительное сокращение

вычислительной сложности алгоритма декодирования кодов с малой плотностью проверки на четность.

6.4. МЕТОДЫ ЭФФЕКТИВНОГО КОДИРОВАНИЯ КОДОВ С МАЛОЙ ПЛОТНОСТЬЮ ПРОВЕРКИ НА ЧТЕНОСТЬ

В этом разделе рассматривается вычислительно эффективный алгоритм кодирования кодов с малой плотностью проверки на четность. Эффективность достигается за счет использования разреженности проверочной матрицы кода. Пусть проверочная матрица кода размерности $m \times n$ задается **H**. По определению линейного блокового кода, любая кодовая последовательность $\mathbf{c}$ удовлетворяет следующему равенству

$$\mathbf{H}\mathbf{c}^T = \mathbf{0}. \qquad (6.4.1)$$

На первом этапе с помощью алгоритма последовательного исключения система уравнений приводится к равносильной системе с проверочной матрицей треугольного вида, показанного на Рис. 42.

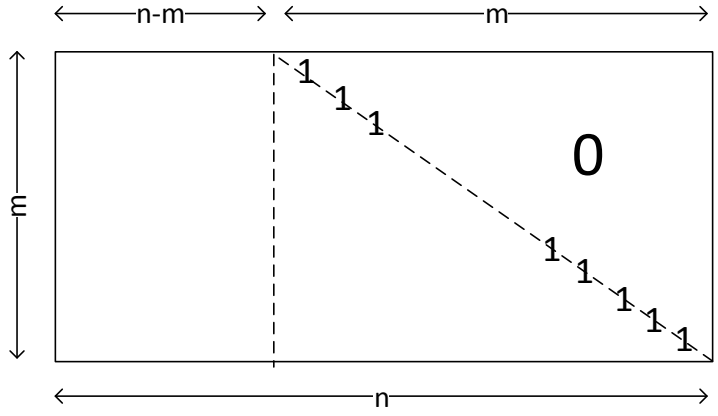


Рис. 42 Эквивалентная проверочная матрица кода треугольного вида

Пусть кодовая последовательность задается объединением систематического $\mathbf{u} = (u_0, u_1, ..., u_{k-1})$ и проверочного $\mathbf{p} = (p_0, p_1, ..., p_{n-k-1})$ векторов – $\mathbf{c} = [\mathbf{u}, \mathbf{p}]$. Тогда

проверочная последовательность **p** длины $m = n - k$ бит может быть получена из уравнения (6.4.2) последовательно с помощью метода обратной подстановки

$$p_l = \sum_{j=0}^{k-1} H_{l,j} u_j + \sum_{j=0}^{l-1} H_{l,j+k} p_j \qquad (6.4.2)$$

Сложность процедуры кодирования, главным образом определяется преобразованием проверочной матрицы к треугольному виду, что требует порядка $O(n^3)$ операций и обратной подстановкой для получения проверочных бит, что требует порядка $O(n^2)$ операций. Поскольку после преобразования проверочной матрицы к треугольному виду, эквивалентная матрица в общем случае не является разреженной, фактическое число требуемых операций для кодирования может быть значительным.

Рассмотрим другой подход кодирования, использующий разреженность проверочной матрицы кода. Предположим, что проверочную матрицу с помощью элементарных операций перестановки столбцов и строк можно привести к следующему виду

$$\mathbf{H} = \begin{bmatrix} \mathbf{A} & \mathbf{B} & \mathbf{T} \\ \mathbf{C} & \mathbf{D} & \mathbf{E} \end{bmatrix}, \qquad (6.4.3)$$

где размеры матриц **A**, **B**, **T**, **C**, **D** и **E** равны $m-g \times n-m$, $m-g \times g$, $m-g \times m-g$, $g \times n-m$, $g \times g$ и $m-g \times g$ соответственно (см. Рис. 43).

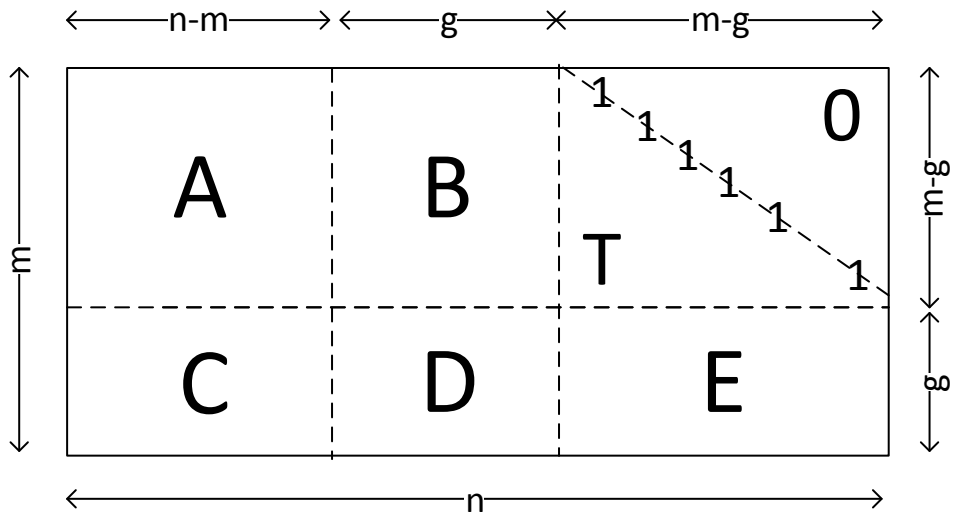


Рис. 43 Эквивалентная проверочная матрица кода приблизительно треугольного вида

Умножая матрицу **H** слева на матрицу

$$\begin{bmatrix} \mathbf{I} & \mathbf{0} \\ -\mathbf{E}\mathbf{T}^{-1} & \mathbf{I} \end{bmatrix}, \tag{6.4.4}$$

можно получить

$$\begin{bmatrix} \mathbf{A} & \mathbf{B} & \mathbf{T} \\ -\mathbf{E}\mathbf{T}^{-1}\mathbf{A}+\mathbf{C} & -\mathbf{E}\mathbf{T}^{-1}\mathbf{B}+\mathbf{D} & \mathbf{0} \end{bmatrix}. \tag{6.4.5}$$

Пусть $\mathbf{c}=[\mathbf{u},\mathbf{p}_1,\mathbf{p}_2]$ кодовая последовательность, состоящая из систематической части $\mathbf{u}$ и двух проверочных частей $\mathbf{p}_1$ и $\mathbf{p}_2$ длиной g и $m-g$ соответственно. Тогда из равенств (6.4.1) и (6.4.5) можно получить систему из двух уравнений

$$\begin{aligned} &\mathbf{A}\mathbf{u}^T+\mathbf{B}\mathbf{p}_1^T+\mathbf{T}\mathbf{p}_2^T=\mathbf{0} \\ &\left(-\mathbf{E}\mathbf{T}^{-1}\mathbf{A}+\mathbf{C}\right)\mathbf{u}^T+\left(-\mathbf{E}\mathbf{T}^{-1}\mathbf{B}+\mathbf{D}\right)\mathbf{p}_1^T=\mathbf{0} \end{aligned}. \tag{6.4.6}$$

Пусть $\mathbf{\Phi}=-\mathbf{E}\mathbf{T}^{-1}\mathbf{B}+\mathbf{D}$ является невырожденной, тогда из выражения (6.4.6) можно получить первую проверочную последовательность

$$\mathbf{p}_1^T=\mathbf{\Phi}^{-1}\left(-\mathbf{E}\mathbf{T}^{-1}\mathbf{A}+\mathbf{C}\right)\mathbf{u}^T. \tag{6.4.7}$$

Поскольку матрица $\mathbf{\Phi}^{-1}\left(-\mathbf{E}\mathbf{T}^{-1}\mathbf{A}+\mathbf{C}\right)$ может быть вычислена заранее, значение вектора $\mathbf{p}_1^T$ может быть получено с помощью $O(g\times k)$ операций.

Сложность вычисления $\mathbf{p}_1^T$ можно сократить с помощью поэтапного вычисления вектора $\mathbf{p}_1^T$. На первом этапе данного метода вычисляется произведение $\mathbf{A}\mathbf{u}^T$, требующее $O(n)$ операций в силу разрежённости матрицы $\mathbf{A}$. На втором этапе, полученный вектор умножают на матрицу $\mathbf{T}^{-1}$. Поскольку $\mathbf{T}$ треугольная матрица и $\mathbf{T}^{-1}\mathbf{A}\mathbf{u}^T=\mathbf{y}^T$ эквивалентно решению системы уравнений $\mathbf{A}\mathbf{u}^T=\mathbf{T}\mathbf{y}^T$, вектор $\mathbf{y}^T$ может быть получен с помощью обратной подстановки с помощью $O(n)$ операций. Произведение $\mathbf{C}\mathbf{u}^T$ может быть также получено с помощью $O(n)$ операций в силу разрежённости матрицы $\mathbf{C}$. На последнем этапе вычисляется произведение $\mathbf{\Phi}^{-1}\left(-\mathbf{E}\mathbf{T}^{-1}\mathbf{A}\mathbf{u}^T+\mathbf{C}\mathbf{u}^T\right)$, что требует $O(g^2)$ операций из-за

общего вида матрицы $\mathbf{\Phi}^{-1}$. Таким образом. Общее число операций требуемых для вычисления $\mathbf{p}_1^T$ равно $O\left(n+g^2\right)$. Аналогичным образом можно получить значения второго проверочного вектора $\mathbf{p}_2$ с помощью выражения

$$\mathbf{p}_2^T = -\mathbf{T}^{-1}\left(\mathbf{A}\mathbf{u}^T + \mathbf{B}\mathbf{p}_1^T\right)^{-1}, \tag{6.4.8}$$

требующее порядка $O(n)$ операций.

Глава 7

ТУРБО КОДЫ

7.1. АЛГОРИТМ ДЕКОДИРОВАНИЯ ПО КРИТЕРИЮ МАКСИМУМА АПОСТЕРИОРНОЙ ВЕРОЯТНОСТИ

Изучение турбо кодов мы начнем с рассмотрения алгоритма декодирования сверточных кодов по критерию максимума апостериорной вероятности (МАВ). Этот алгоритм был предложен в 1974 году. В литературе алгоритм МАВ также получил названия BCJR (по первым буквам фамилий авторов Bahl, Cocke, Jelinek и Raviv) или «прямой-обратный» алгоритм. Стоит отметить, что в силу высокой вычислительной сложности, алгоритм МАВ долгое время не использовался в практических приложениях. Однако, ситуация существенно изменилась после изобретения в 1993 году группой математиков К. Берроу, А. Главьё и П. Ситимашимой нового класса кодов – турбо кодов. Для декодирования предложенного ими кода был использован модифицированный BCJR алгоритм. После этого различные модификации МАВ алгоритма стали активно внедряться в различные практические приложения декодирования турбо кодов.

Пусть $\mathbf{x} = \mathbf{x}_0, \mathbf{x}_2, \ldots, \mathbf{x}_{N-1}$ кодовая последовательность, состоящая из N символов длины n бит, полученная с помощью сверточного кодирования. Символ $\mathbf{x}_k$ соответствует кодовой последовательности, полученной сверточным кодом в момент времени k. Для простоты рассмотрения будем считать что -1 и +1 соответствуют 0 и 1 соответственно. Тогда информационная последовательность u_k может принимать значения -1 или +1 и имеет априорную вероятность $P(u_k)$. Предположим, что кодовая последовательность $\mathbf{x}$ была передана в канале с аддитивным белым гауссовским шумом. Тогда принятая последовательность $\mathbf{r} = \mathbf{r}_0, \mathbf{r}_2, \ldots, \mathbf{r}_{N-1}$ может быть представленная в следующем виде

$$\mathbf{r} = \mathbf{x} + \mathbf{n}, \tag{7.1.1}$$

где $\mathbf{n}$ реализация гауссовского шума. Для декодирования по критерию МАВ принятая последовательность $\mathbf{r}$ используется для вычисления логарифма отношения вероятностей

$$L(u_k \mid \mathbf{r}) = \log \frac{P(u_k = +1 \mid \mathbf{r})}{P(u_k = -1 \mid \mathbf{r})}, \tag{7.1.2}$$

где числитель и знаменатель в логарифме выражения (7.1.2) представляет собой апостериорную вероятность при условии принятой последовательности $\mathbf{r}$. Информация, содержащаяся в $L(u_k \mid \mathbf{r})$, может быть использована как для принятия решения о преданном информационного бите, так и предана на следующий блок декодирования в качестве априорной информации.

Для удобства рассмотрим решетку сверточного кода со скоростью $R = 1/2$, $n = 2$ и двумя регистрами сдвига. Число состояний кода M в этом случае равно 4. Пример сегмента решетки кода представлен на Рис. 44

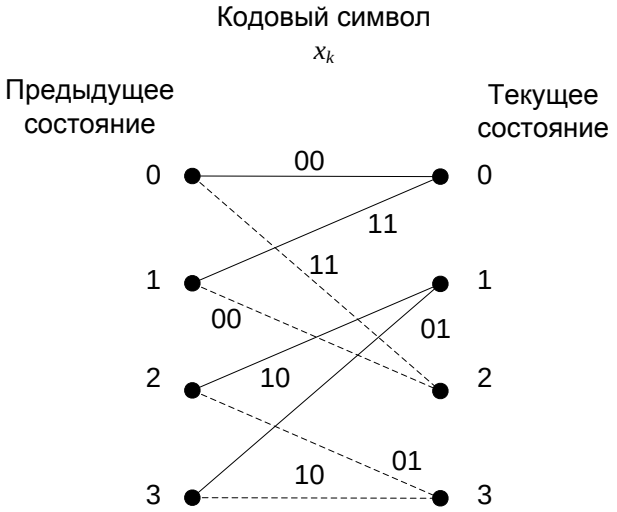


Рис. 44. Сегмент решетки сверточного кода

Напомним, что узлы обозначают возможные состояния кода, сплошные линии показывают ребра решетки соответствующие информационному биту -1, а пунктирные, ребра соответствующие информационному биту +1. Рассмотрим сегмент решетки в момент времени k. Состояние в текущий момент времени k обозначим как $S_k = s$, а в предыдущий $k-1$ как $S_{k-1} = s'$. Поскольку при заданном состоянии информационный бит однозначно определяет переход кода из одного состояния решетки в другое выражение (7.1.2) может быть записано в следующем виде

$$L(u_k \mid \mathbf{r}) = \log \frac{P(u_k = +1 \mid \mathbf{r})}{P(u_k = -1 \mid \mathbf{r})} = \log \frac{\sum\limits_{R_1} P(s', s \mid \mathbf{r})}{\sum\limits_{R_0} P(s', s \mid \mathbf{r})}, \tag{7.1.3}$$

где R_1 и R_0 задают множество переходов (ребер в решетке) с начальным состоянием s' и конечным s, соответствующее информационным битам $u_k = +1$ и $u_k = -1$. Выражение (7.1.3) можно переписать в следующем виде

$$L(u_k \mid \mathbf{r}) = \log \frac{\sum\limits_{R_1} P(s', s \mid \mathbf{r}) P(\mathbf{r})}{\sum\limits_{R_0} P(s', s \mid \mathbf{r}) P(\mathbf{r})} = \log \frac{\sum\limits_{R_1} P(s', s, \mathbf{r})}{\sum\limits_{R_0} P(s', s, \mathbf{r})} \tag{7.1.4}$$

Для дальнейшего упрощения выражения (7.1.4) принятую последовательность $\mathbf{r}$ можно условно разделить на три сегмента

$$\mathbf{r} = \underbrace{\mathbf{r}_0, \mathbf{r}_2, \ldots, \mathbf{r}_{k-1}}_{\mathbf{r}_0^{k-1}}, \mathbf{r}_k, \underbrace{\mathbf{r}_{k+1}, \ldots, \mathbf{r}_{N-1}}_{\mathbf{r}_{k+1}^{N-1}} = \mathbf{r}_0^{k-1} \mathbf{r}_k \mathbf{r}_{k+1}^{N-1}. \tag{7.1.5}$$

Тогда используя правило Байеса можно получить

$$P(s', s, \mathbf{r}) = P\left(s', s, \mathbf{r}_0^{k-1} \mathbf{r}_k \mathbf{r}_{k+1}^{N-1}\right) = P\left(\mathbf{r}_{k+1}^{N-1} \mid s', s, \mathbf{r}_0^{k-1} \mathbf{r}_k\right) P\left(s', s, \mathbf{r}_0^{k-1} \mathbf{r}_k\right) \tag{7.1.6}$$

Поскольку состояние решетки $S_k = s$ в момент времени k задано, то последовательность $\mathbf{r}_{k+1}^{N-1}$ не зависит от состояния $S_{k-1} = s'$ и последовательностей $\mathbf{r}_0^{k-1} \mathbf{r}_k$. В этом случае выражение (7.1.6) можно записать

$$P(s', s, \mathbf{r}) = P\left(\mathbf{r}_{k+1}^{N-1} \mid s\right) P\left(s', s, \mathbf{r}_0^{k-1} \mathbf{r}_k\right). \tag{7.1.7}$$

Упрощая второй множитель выражения (7.1.7) получим

$$P(s', s, \mathbf{r}) = P\left(\mathbf{r}_{k+1}^{N-1} \mid s\right) P\left(s, \mathbf{r}_k \mid s', \mathbf{r}_0^{k-1}\right) P\left(s', \mathbf{r}_0^{k-1}\right) = P\left(\mathbf{r}_{k+1}^{N-1} \mid s\right) P(s, \mathbf{r}_k \mid s') P\left(s', \mathbf{r}_0^{k-1}\right) \tag{7.1.8}$$

Вероятность $P\left(\mathbf{r}_{k+1}^{N-1} \mid s\right) = \beta_k(s)$ определяет условную вероятность последовательности $\mathbf{r}_{k+1}^{N}$ при заданном состоянии $S_k = s$. $P(s, \mathbf{r}_k \mid s') = \gamma_k(s', s)$ задает вероятность последовательности $\mathbf{r}_k$ и состояния $S_k = s$ при заданном состоянии $S_{k-1} = s'$. Вероятность $P\left(s', \mathbf{r}_0^{k-1}\right) = \alpha_{k-1}(s')$ задает совместную вероятность принятой последовательности $\mathbf{r}_0^{k-1}$ и состояния $S_{k-1} = s'$. Подставляя полученные выражения в (7.1.3)

$$L(u_k \mid \mathbf{r}) = \log \frac{\sum\limits_{R_1} \alpha_{k-1}(s')\gamma_k(s',s)\beta_k(s)}{\sum\limits_{R_0} \alpha_{k-1}(s')\gamma_k(s',s)\beta_k(s)}. \qquad (7.1.9)$$

Рассмотрим вычисление множителя $P(s, r_k \mid s') = \gamma_k(s',s)$. Используя правило Байеса

$$\gamma_k(s',s) = P(s, \mathbf{r}_k \mid s') = P(\mathbf{r}_k \mid s', s)P(s \mid s') \qquad (7.1.10)$$

Рассмотрим второй множитель $P(s \mid s')$. Поскольку состояние s' задано, то вероятность перерода в одно из двух возможных состояний будет полностью определяться вероятностью соответствующего информационного бита, т.е. $P(u_k)$. Например, если информационные биты равновероятны $P(u_k = \pm 1) = 1/2$, то вероятность перехода из текущего состояния s' в одно из двух возможных состояний s будет также равновероятным. Аналогично вероятность $P(\mathbf{r}_k \mid s', s)$ будет равна вероятности $P(\mathbf{r}_k \mid u_k) = P(\mathbf{r}_k \mid \mathbf{x}_k)$. Возвращаясь к выражению $P(u_k)$ и вводя $L(u_k) = \log \dfrac{P(u_k = +1)}{P(u_k = -1)}$ легко получить

$$P(u_k = \pm 1) = c_{1k} \exp(u_k L(u_k)/2), \qquad (7.1.11)$$

где коэффициент $c_{1k} = \dfrac{\exp(L(u_k)/2)}{1 + \exp(L(u_k))}$ не зависит от u_k и равен 1/2 при $P(u_k) = 1/2$.

Вероятность $P(\mathbf{r}_k \mid \mathbf{x}_k)$, что n элементов $r_k^{(1)}, r_k^{(2)}, ..., r_k^{(n)}$ получено при заданной кодовой последовательности $x_k^{(1)}, x_k^{(2)}, ..., x_k^{(n)}$, равно произведению вероятностей $P\left(r_k^{(l)} \mid x_k^{(l)}\right)$, т.е.

$$P(\mathbf{r}_k \mid \mathbf{x}_k) = \prod_{l=1}^{n} P\left(r_k^{(l)} \mid x_k^{(l)}\right) \qquad (7.1.12)$$

Для рассматриваемого примера сверточного кода

$$P(\mathbf{r}_k \mid \mathbf{x}_k) = P\left(r_k^{(1)} \mid x_k^{(1)}\right) P\left(r_k^{(2)} \mid x_k^{(2)}\right) \qquad (7.1.13)$$

Отметим, что вероятности $P\left(r_k^{(l)} \mid x_k^{(l)}\right)$ зависят от модели канала и используемой модуляции. Для двоичной фазовой модуляции (BPSK)

$$P\left(r_k^{(l)} \mid x_k^{(l)}\right) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{\left(r_k^{(l)} - x_k^{(l)}\right)^2}{2\sigma^2}\right) \qquad (7.1.14)$$

Подставляя полученное выражение (7.1.14) в (7.1.12) получим

$$P(\mathbf{r}_k \mid \mathbf{x}_k) = \exp\left(-\sum_{l=1}^{n} \frac{r_k^{(l)} x_k^{(l)}}{\sigma^2}\right) \cdot \underbrace{\left\{\left(\frac{1}{\sqrt{2\pi\sigma^2}}\right)^n \exp\left(-\sum_{l=1}^{n} \frac{r_k^{(l)2}}{2\sigma^2}\right) \exp\left(-\sum_{l=1}^{n} \frac{x_k^{(l)2}}{2\sigma^2}\right)\right\}}_{c_{2k}} = c_{2k} \exp\left(\sum_{l=1}^{n} \frac{r_k^{(l)} x_k^{(l)}}{\sigma^2}\right) \quad (7.1.15)$$

Множитель c_{2k} не зависит от информационной последовательности и является одинаковой величиной для числителя и знаменателя выражения (7.1.9). Напротив, значение второго множителя выражения (7.1.15) зависит от кодовой последовательности. Вводя $L_c = \frac{2}{\sigma^2}$ и $r_k \cdot x_k = \sum_{l=1}^{n} r_k^{(l)} x_k^{(l)}$, получим

$$\gamma_k(s', s) = C_k \exp\left(u_k \frac{L(u_k)}{2}\right) \exp\left(\frac{L_c}{2}(r_k \cdot x_k)\right), \quad (7.1.16)$$

где множитель $c_k = c_{1k} \cdot c_{2k}$ является одинаковым для числителя и знаменателя выражения (7.1.9).

Рассмотрим вычисление множителя $\alpha_{k-1}(s') = P\left(s', \mathbf{r}_1^{k-1}\right)$. В момент времени k

$$\alpha_k(s) = P\left(s, \mathbf{r}_k \mathbf{r}_0^{k-1}\right) = \sum_{s'} P\left(s, s', \mathbf{r}_k \mathbf{r}_0^{k-1}\right) \quad (7.1.17)$$

Используя правило Байеса

$$\sum_{s'} P\left(s, s', \mathbf{r}_k \mathbf{r}_0^{k-1}\right) = \sum_{s'} P\left(s, \mathbf{r}_k \mid s', r_0^{k-1}\right) P\left(s', \mathbf{r}_0^{k-1}\right) = \sum_{s'} P\left(s, \mathbf{r}_k \mid s'\right) P\left(s', \mathbf{r}_0^{k-1}\right). \quad (7.1.18)$$

Легко видеть, что

$$\alpha_k(s) = \sum_{s'} \alpha_{k-1}(s') \gamma_k(s', s). \quad (7.1.19)$$

Таким образом, если нам известно $\alpha_{k-1}(s')$, значение $\alpha_k(s)$ может быть получено путем суммирования произведений $\alpha_{k-1}(s')$ и $\gamma_k(s', s)$ соответствующих ребрам, входящим в состояние s в момент времени k. На Рис. 45 приведен пример вычисления $\alpha_k(s)$ для второго состояния.

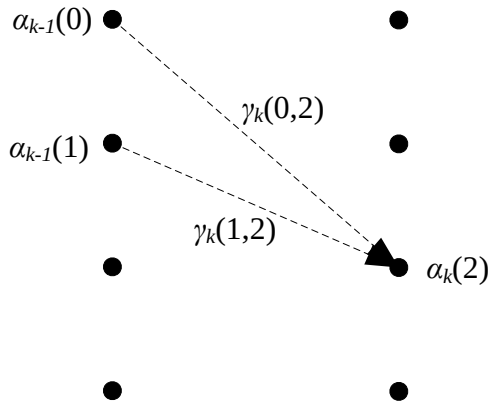


Рис. 45. Пример вычисления $\alpha_k(s)$

Таким образом, значения $\alpha_k(s)$ могут быть вычислены рекурсивно (прямая рекурсия) при заданном начальном состоянии $\alpha_0(s)$. Например, для решеток с фиксированным начальным (нулевым) состоянием

$$\alpha_{-1}(s)=\begin{cases}1, s=0\\0, s\neq 0\end{cases}. \tag{7.1.20}$$

Рекурсивное вычисление $\beta_k(s)=P\left(\mathbf{r}_{k+1}^{N}\mid s\right)$ может быть получено аналогичным способом. Для этого рассмотрим момент времени $k-1$

$$\beta_{k-1}(s')=P\left(\mathbf{r}_{k}^{N-1}\mid s'\right)=\sum_{s}P\left(s,\mathbf{r}_{k+1}^{N-1}\mathbf{r}_k\mid s'\right). \tag{7.1.21}$$

Применяя правило Байеса, получим

$$\beta_{k-1}(s')=\sum_{s}P\left(\mathbf{r}_{k+1}^{N-1}\mid s',s,\mathbf{r}_k\right)P\left(s,\mathbf{r}_k\mid s'\right)=\sum_{s}P\left(\mathbf{r}_{k+1}^{N-1}\mid s\right)P\left(s,\mathbf{r}_k\mid s'\right)=\sum_{s}\beta_k(s)\gamma_k(s',s). \tag{7.1.22}$$

При завершении кодирования конечное состояние кода считается известным. Тогда начальные значения для рекурсивного вычисления $\beta_k(s)$ (обратная) определяются следующим образом

$$\beta_{N-1}(s)=\begin{cases}1, s=0\\0, s\neq 0\end{cases}. \tag{7.1.23}$$

Пример вычисления $\beta_k(s)$ показан на Рис. 46.

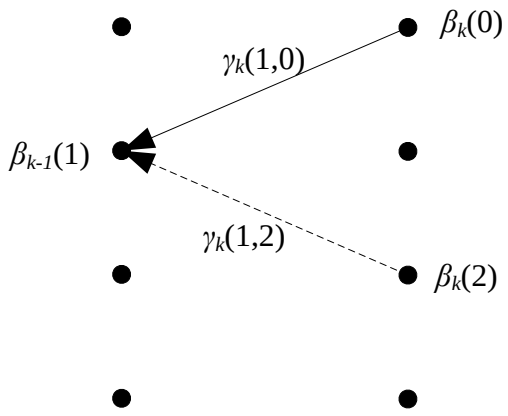


Рис. 46. Пример вычисления $\beta_k(s)$

После вычисления $\gamma_k(s',s), \alpha_{k-1}(s')$ и $\beta_k(s)$ для каждого сегмента решетки кода можно вычислить значение $P(s',s,\mathbf{r})$ для каждого момента времени и пары состояний s' и s.

$$P(s',s,\mathbf{r}) = \alpha_{k-1}(s')\gamma_k(s',s)\beta_k(s) \tag{7.1.24}$$

Подставляя вычисленные значения $P(s',s,\mathbf{y})$ в выражение (7.1.9) получим искомые значения $L(u_k \mid \mathbf{r})$. Для рассматриваемого примера кода $L(u_k \mid \mathbf{r})$ равно

$$L(u_k \mid \mathbf{r}) = \log\frac{P(0,2,\mathbf{r}) + P(1,2,\mathbf{r}) + P(2,3,\mathbf{r}) + P(3,3,\mathbf{r})}{P(0,0,\mathbf{r}) + P(1,0,\mathbf{r}) + P(2,1,\mathbf{r}) + P(3,1,\mathbf{r})} \tag{7.1.25}$$

7.2. АЛГОРИТМ ДЕКОДИРОВАНИЯ ТУРБО КОДОВ

Далее рассмотрим систематический сверточный код, в котором кодовый бит $x_k^{(1)} = u_k$ равен информационному биту. Чуть позже мы покажем, что $L(u_k \mid \mathbf{r})$ может быть представлена как сумма трех слагаемых

$$L(u_k \mid \mathbf{r}) = L(u_k) + L_c r_k^{(1)} + L_e(u_k) \tag{7.2.1}$$

Первые два слагаемых относятся к информационному символу u_k. Напротив, третий множитель $L_e(u_k)$ зависит только от проверочных бит. Слагаемое $L_e(u_k)$ принято называть *внешней информацией*. Эта информация является оценкой априорной информации,

поскольку при заданных входных значениях $L(u_k)$ и $L_c r_{k1}$, декодер МАВ получает $L(u_k \mid \mathbf{r})$. Тогда дополнительная информация, получаемая из процедуры декодирования, может быть получена

$$L_e(u_k) = L(u_k \mid \mathbf{r}) - L(u_k) - L_c r_k^{(1)} \tag{7.2.2}$$

Полученная оценка априорной информации может быть использована как входная информация на следующем устройстве декодирования, на выходе которого мы ожидаем получить более точную оценку $L(u_k \mid \mathbf{r})$. Эта идея используется при декодировании турбо кодов, которые мы рассмотрим далее. Общая структура турбо кода с параллельным соединением кодов показана на Рис. 47.

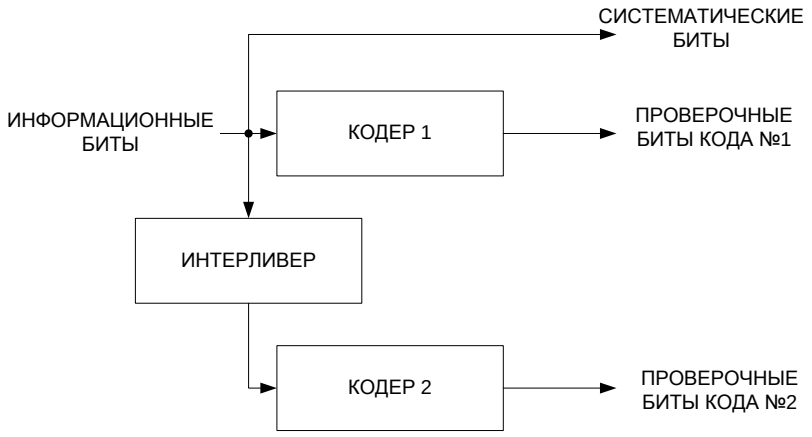


Рис. 47. Структура турбо кода с параллельным соединением кодов

Соответствующая структура турбо декодера показана на Рис. 48.

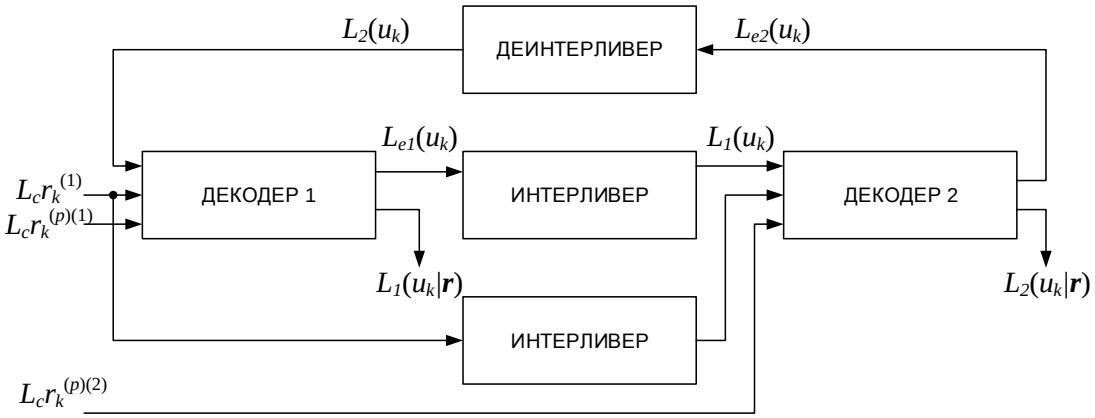


Рис. 48 Структура турбо декодера

Рассмотрим принцип работы итеративного турбо декодирования:

1. На первой итерации происходит инициализация параметров. Поскольку приемник не имеет априорной информации о переданных информационных битах $L(u_k)=0$.
2. На основе принятой кодовой последовательности (систематической $L_c r_{kl}$ и проверочных $L_c r_k^{(p)(1)}$ частей) и априорной информации $L(u_k)$ декодер МАВ первого кода вычисляет внешнюю информацию $L_{e1}(u_k \mid \mathbf{r})$
3. После соответствующей перестановки в блоке интерливинга внешняя информация $L_{e1}(u_k)$ поступает на вход декодера второго кода в качестве априорной информации $L_1(u_k)$ о систематических битах. На основании априорной информации и принятой кодовой последовательности (систематической $L_c r_k^{(l)}$ и проверочных $L_c r_k^{(p)(2)}$ частей) декодер МАВ второго кода вычисляет внешнюю информацию $L_{e2}(u_k \mid \mathbf{r})$, которая подается на декодер первого кода.
4. После определенного числа итераций с выхода декодера первого или второго кода значения $L_1(u_k \mid \mathbf{r})$ или $L_2(u_k \mid \mathbf{r})$ подаются на блок оценки переданной информационной последовательности.
5. Если $L(u_k \mid \mathbf{r})>0$, то принимается решение $u_k=+1$. Если $L(u_k \mid \mathbf{r})\le 0$, то принимается решение $u_k=-1$

Вернемся к выражению (7.2.2) и покажем его справедливость. Напомним, что рассматриваемый сверточный код является систематическим и $x_{1k} = u_k$. Тогда выражение (7.1.16) можно записать в следующем виде

$$\gamma_k(s', s) = C_k \exp\left(u_k \left[\frac{L(u_k)}{2} + \frac{L_c r_k^{(1)}}{2} \right] \right) \exp\left(\frac{L_c}{2} \sum_{l=2}^{n} x_k^{(l)} r_k^{(l)} \right) \tag{7.2.3}$$

Введем дополнительное обозначение второго множителя выражения (7.2.3)

$$\delta_k(s', s) = \exp\left(\frac{L_c}{2} \sum_{l=2}^{n} x_k^{(l)} r_k^{(l)} \right). \tag{7.2.4}$$

Подставляя (7.2.4) в выражение (7.2.3) а затем в (7.1.9) получим

$$L(u_k \mid \mathbf{r}) = \log \frac{\sum_{R_1} C_k \exp\left(u_k \left[\frac{L(u_k)}{2} + \frac{L_c r_k^{(1)}}{2} \right] \right) \alpha_{k-1}(s') \delta_k(s', s) \beta_k(s)}{\sum_{R_0} C_k \exp\left(u_k \left[\frac{L(u_k)}{2} + \frac{L_c r_k^{(1)}}{2} \right] \right) \alpha_{k-1}(s') \delta_k(s', s) \beta_k(s)} \tag{7.2.5}$$

Поскольку множества R_0 и R_1 соответствуют информационным битам $u_k = -1$ и $u_k = +1$, прямая подстановка u_k в (7.2.5) дает

$$L(u_k \mid \mathbf{r}) = \log \left\{ \exp(L(u_k)) \cdot \exp\left(L_c r_k^{(1)}\right) \cdot \frac{\sum_{R_1} \alpha_{k-1}(s') \delta_k(s', s) \beta_k(s)}{\sum_{R_0} \alpha_{k-1}(s') \delta_k(s', s) \beta_k(s)} \right\} \tag{7.2.6}$$

Легко видеть, что выражение (7.2.6) может быть представлено как сумма трех слагаемых

$$L(u_k \mid \mathbf{r}) = L(u_k) + L_c r_k^{(1)} + L_e(u_k), \tag{7.2.7}$$

где

$$L_e(u_k) = \log \left\{ \frac{\sum_{R_1} \alpha_{k-1}(s') \delta_k(s', s) \beta_k(s)}{\sum_{R_0} \alpha_{k-1}(s') \delta_k(s', s) \beta_k(s)} \right\} \tag{7.2.8}$$

Заметим, что для вычисления $L_e(u_k)$ можно использовать как выражение (7.2.8) так и выражение (7.2.2).

7.3. МОДИФИКАЦИИ АЛГОРИТМОВ ДЕКОДИРОВАНИЯ ТУРБО КОДОВ

Как уже отмечалось ранее, основным недостатком алгоритма декодирования заключается в относительно высокой вычислительной сложности. Далее рассмотрим модификации алгоритма MAB, позволяющие существенно упростить процедуру декодирования с приемлемой потерей помехоустойчивости. Для этого введем следующие переменные

$$\Gamma_k(s',s)=\log(\gamma_k(s',s))=\log(C_k)+u_k\frac{L(u_k)}{2}+\frac{L_c}{2}(r_k\cdot x_k), \tag{7.3.1}$$

$$\mathrm{A}_k(s)=\log(\alpha_k(s))=\max_{s'}{}^*(\mathrm{A}_{k-1}(s')+\Gamma_k(s',s)), \tag{7.3.2}$$

$$\mathrm{B}_{k-1}(s')=\log(\beta_{k-1}(s'))=\max_{s}{}^*(\mathrm{B}_k(s)+\Gamma_k(s',s)), \tag{7.3.3}$$

где

$$\max^*(x+y)=\begin{cases}\max(x,y)+\log(1+\exp(-|x-y|)) & \log\text{ MAB алгоритм}\\ \max(x,y) & \max\text{ - }\log\text{ MAB алгоритм}\end{cases} \tag{7.3.4}$$

Тогда выражение для вычисления апостериорной вероятности информационного бита можно упростить как

$$L(u_k\mid\mathbf{r})=\max_{R_1}{}^*(\mathrm{A}_{k-1}(s')+\Gamma_k(s',s)+\mathrm{B}_k(s))-\max_{R_0}{}^*(\mathrm{A}_{k-1}(s')+\Gamma_k(s',s)+\mathrm{B}_k(s)) \tag{7.3.5}$$

Вычисление $\max^*(\cdot)$ от трех аргументов в выражении 7.3.5 может быть выполнено рекурсивно с использованием следующего равенства

$$\max^*(x_1+x_2+x_3)=\max^*(\max^*(x_1+x_2)+x_3). \tag{7.3.6}$$

7.4. ОСНОВНЫЕ ПРИНЦИПЫ ПОСТРОЕНИЯ ТУРБО КОДОВ

После рассмотрения процедуры декодирования обсудим структуру турбо кодов, показанную на Рис. 47. Напомним, что в классической теореме Шеннона утверждается, что с помощью случайного кода можно обеспечить передачу данных со сколь угодно низкой вероятностью ошибки. При этом безошибочная передача обеспечивается при больших длинах кодового слова. Как мы уже видели ранее, проблема использования случайного кода с большой длиной кодового слова состоит в сложности его декодирования. Случайность или отсутствие структуры кода приводит к существенным вычислительным затратам при декодировании. Идея построения случайного кода с приемлемой сложностью декодирования реализуется в турбо кодах. Как видно из Рис. 47, турбо код состоит из двух параллельно соединенных компонентных кодов, разделенных внутренним интерливером. Задача интерливера является наиболее важной для обеспечения случайности кода и состоит в перестановке информационной последовательности для «декорреляции» проверочных бит, полученных компонентными кодами. Заметим, что для получения случайного кода, применение псевдослучайного интерливера является предпочтительным. Стоит также отметить, что высокая помехоустойчивость всего кода достигается даже при использовании относительно простых компонентных кодов (например, сверточных кодов с малой длиной кодового ограничения). Не смотря на простоту каждого компонентного кода, структура всего турбо кода является достаточно сложной для использования методов прямого декодирования. Использование итеративного метода декодирования, рассмотренного выше, позволяет получить достаточно высокую эффективность кодирования. При этом систематичность компонентных кодов требуется для применения таких алгоритмов декодирования. Стоит отметить, что для уменьшения вероятности появления ненулевой кодовой последовательности минимального веса в качестве компонентных кодов, как правило, используют рекурсивные систематические сверточные коды. Использование рекурсивных сверточных кодов, разделенных псевдослучайным интерливером, позволяет существенно уменьшить вероятность одновременного появления кодовых последовательностей минимального веса на выходе компонентных кодов.

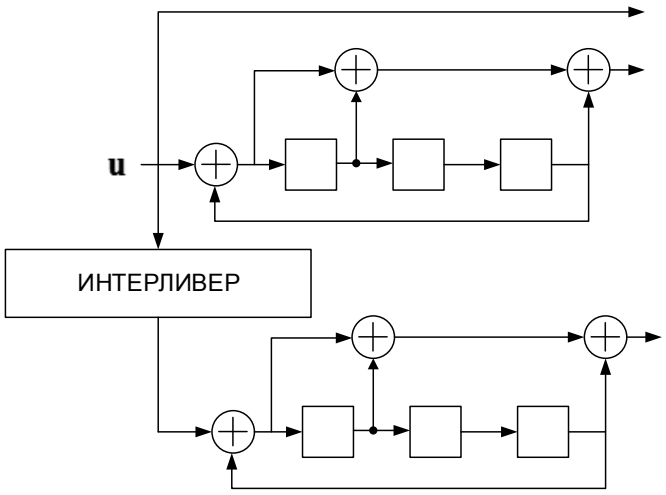


Рис. 49. Пример турбо кода стандарта 3GPP Release 8

Пример турбо кода используемого в 3GPP Release 8 стандарте сотовой связи показан на Рис. 49. В коде используются два систематических рекурсивных сверточных кода со скоростью кода 1/2. В качестве интерливинга используется квадратичный интерливер с индексами перестановки задаваемыми выражением

$$j = \left(f_1 \cdot i + f_2 \cdot i^2\right) \bmod N, \tag{7.4.1}$$

где N длина информационного блока, а f_1 и f_2 параметры интерливера.

7.5. МЕТОД ПАРАЛЛЕЛЬНОГО ДЕКОДИРОВАНИЯ ТУРБО КОДОВ

Последовательное вычисление внешней информации для систематических бит компонентного кода с помощью алгоритма MAB, как правило, приводит к существенной задержке при итеративном декодировании турбо кодов. Поэтому для высокоскоростных систем связи одной из наиболее важных задач является возможность реализации параллельного декодирования принятой последовательности.

Для турбо кодов параллельное декодирование реализуется с помощью разбиения решетки кода на сегменты длины W и параллельного вычисления внешней информации для систематических бит $L_e(u_k)$ в каждом сегменте решетки (см. Рис. 50).

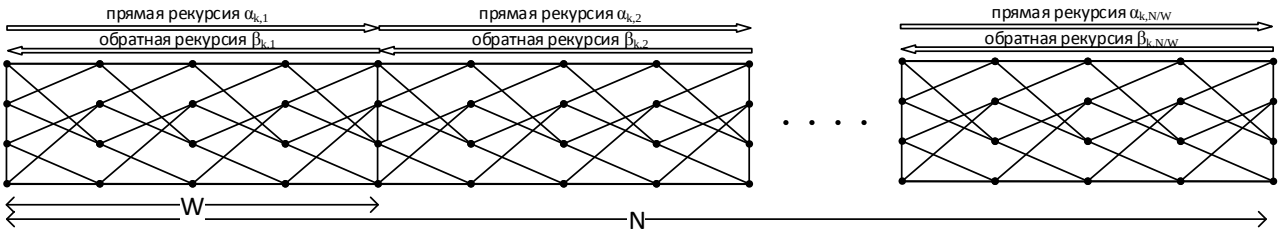


Рис. 50. Разбиение решетки кода на сегменты для параллельного декодирования принятой последовательности

Эффективность такого подхода параллельного вычисления внешней информации определяется, главным образом, возможностью параллельного доступа в память для записи вычисленных значений внешней информации каждого сегмента решетки. Для обеспечения такой записи в архитектуре декодера используется несколько блоков памяти, а интерливинг строится, таким образом, чтобы избежать одновременного доступа к одному блоку памяти более чем с одного сегмента решетки при записи вычисленных значений $L_e(u_k)$.

Пример процедуры интерливинга, не имеющего конфликта параллельного доступа к памяти, показан на Рис. 51 для четырех сегментов решетки длины $W = 4$. Из рисунка видно, что для фиксированного индекса бита i внутри каждого сегмента решётки запись производится в память с адресом $f(i+tW)$ (где $f(\cdot)$ функция интерливера). При этом индекс блока памяти, определяемый значением $\lfloor f(i+tW)/W \rfloor$, для каждого сегмента решетки *t* всегда оказывается разным.

В общем случае условие отсутствия конфликта одновременного доступа к памяти для интерливера $f(\cdot)$ можно записать следующим образом

$$\left\lfloor \frac{f(i+vW)}{W} \right\rfloor \neq \left\lfloor \frac{f(i+uW)}{W} \right\rfloor, 0 \le v,u \le N-1, \qquad (7.5.1)$$

где $i \in 0,...,W-1$ индекс систематического бита внутри сегмента решетки.

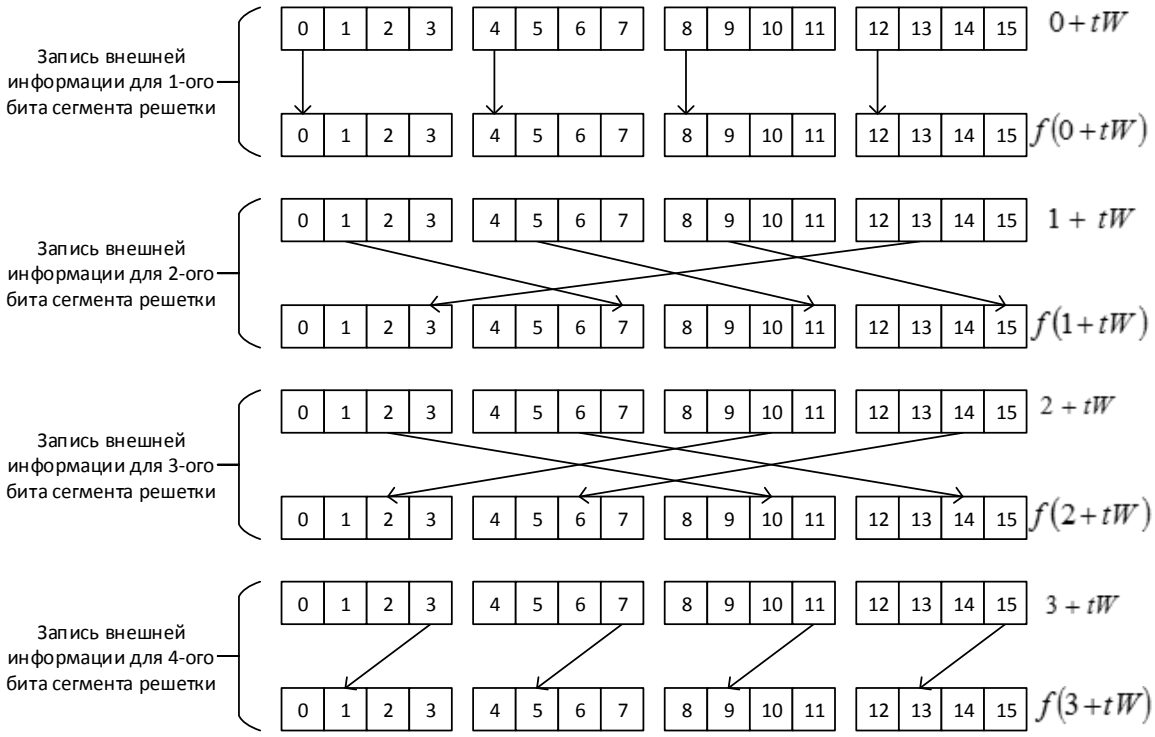


Рис. 51. Пример процедуры интерливинга для параллельной архитектуры кода

Определение: Интерливер обеспечивающий отсутствие конфликта одновременного доступа к памяти для всех возможных значений длин сегментов решетки W (являющегося делителем N) называется интерливером с *максимальным* отсутствием конфликта одновременного доступа к памяти.

Можно показать, что квадратичный интерливер $j = \left(f_1 \cdot i + f_2 \cdot i^2\right) \bmod N$ удовлетворяет условию максимально отсутствия конфликта одновременного доступа к памяти.

Доказательство:

Пусть

$$Q_v = \left\lfloor \frac{f(i+vW)}{W} \right\rfloor, \quad Q_u = \left\lfloor \frac{f(i+uW)}{W} \right\rfloor, \tag{7.5.2}$$

тогда

$$\begin{aligned} f(i+vW) &= Q_v W + \mathrm{mod}(f(i+vW), W) \\ f(i+uW) &= Q_u W + \mathrm{mod}(f(i+uW), W) \end{aligned}. \qquad (7.5.3)$$

Необходимо показать, что для любого $i \in 0,...,W-1$, $Q_v \neq Q_u$ для любого $u \neq v$. Действительно, предположим что $Q_v = Q_u$. Тогда используя выражение (7.5.3) получим

$$Q_v - Q_u = \frac{f(i+vW) - f(i+uW) + \mathrm{mod}(f(i+vW), W) - \mathrm{mod}(f(i+uW), W)}{W} = 0. \quad (7.5.4)$$

Поскольку для квадратичного интерливера выполняется равенство $\mathrm{mod}(f(i+vW), W) = \mathrm{mod}(f(i+uW), W)$, то $f(i+vW) = f(i+uW)$. Данное равенство противоречит уникальности функции интерливера $f(\cdot)$. Отсюда следует, что $Q_v \neq Q_u$ для любого $u \neq v$ и справедливо для всех индексов $i \in 0,...,W-1$.

СПИСОК ЛИТЕРАТУРЫ

1. Прокис Дж. Цифровая связь. Пер. с англ. / Под ред. Д. Д. Кловского. – М.: Радио и связь, 2000. – 800 с.
2. Феер К. Беспроводная цифровая связь. Методы модуляции и расширения спектра. Пер. с англ. – М.: Радио и связь, 2000. – 520 с.
3. Блейхут Р. Теория и практика кодов, контролирующих ошибки. – М.: Мир, 1986. – 576 с.
4. Кларк Дж. мл., Кейн Дж. Кодирование с исправлением ошибок в системах цифровой связи: Пер. с англ. – М.: Радио и связь, 1987. – 392 с.
5. Витерби А. Д., Омура Дж. К. Принципы цифровой связи и кодирования. – М.: Радио и связь, 1982. – 536 с.
6. T. K. Moon, Error Correction Coding: Mathematical Methods and Algorithms, Wiley 2005. – p.756.
7. Мак-Вильямс Ф. Дж., Слоэн Н.Дж.А. Теория кодов, исправляющих ошибки. – М.: Связь, 1979. – 687 с.
8. Sklar, Bernard. Digital Communications: Fundamentals and Applications. –Englewood Cliffs, N.J.: Prentice-Hall, 1988. – p. 1079.
9. Clark, George C. Jr. and J. Bibb Cain. Error-Correction Coding for Digital Communications. – New York: Plenum Press, 1981. – p. 436.
10. Lin, Shu and Daniel J. Costello, Jr. Error Control Coding: Fundamentals and Applications. – Englewood Cliffs, N.J.: Prentice-Hall, 1983. – p. 603.
11. Peterson, W. Wesley and E. J. Weldon, Jr. Error-correcting Codes, 2nd ed. –Cambridge, Mass.: MIT Press, 1972. – p. 572.
12. van Lint, J. H. Introduction to Coding Theory. – New York: Springer-Verlag, 1982. – p. 241.
13. Proakis, John G. Digital Communications, 3rd ed. – New York: McGraw-Hill, 1995. – p. 1024.
14. Blahut, Richard E. Theory and Practice of Error Control Codes. Reading, Mass.: Addison-Wesley, 1983, – p.105.
15. Lang, Serge. Algebra. Third Edition. Reading, Mass.: Addison-Wesley, 1993. – p. 914.

ОГЛАВЛЕНИЕ