Министерство образования и науки Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ»

МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ

«Исследование зависимости вероятности ошибки на блок от спектра графа Таннера для МППЧ-кодов»

Автор: Ковшаров Антон Павлович _____
Направление подготовки (специальность): 01.04.02 Прикладная математика и информатика
Квалификация: Магистр
Руководитель: Буздалов М.В., канд. техн. наук _____

К защите допустить
Зав. кафедрой Васильев В.Н., докт. техн. наук, проф. _____

«___» _____ 20___ г.

Санкт-Петербург, 2017 г.

Студент Ковшаров А.П. **Группа** М4239 **Кафедра** компьютерных технологий **Факультет** информационных технологий и программирования

Направленность (профиль), специализация Технологии проектирования и разработки программного обеспечения

Консультанты:

а) Кудряшов Б.Д., докт. техн. наук, профессор _____

б) Бочарова И.Е., канд. техн. наук, доцент _____

Квалификационная работа выполнена с оценкой _____

Дата защиты «15» июня 2017 г.

Секретарь ГЭК _____Принято: «___» _____ 20___ г.

Листов хранения _____

Демонстрационных материалов/Чертежей хранения _____

Министерство образования и науки Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ»

УТВЕРЖДАЮ
Зав. каф. компьютерных технологий
докт. техн. наук, проф.
_____ Васильев В.Н.
«___» _____ 20___ г.

ЗАДАНИЕ
НА МАГИСТЕРСКУЮ ДИССЕРТАЦИЮ

Студент Ковшаров А.П. **Группа** М4239 **Кафедра** компьютерных технологий **Факультет** информационных технологий и программирования **Руководитель** Буздалов Максим Викторович, канд. техн. наук, научный сотрудник Университета ИТМО

1 Наименование темы: Исследование зависимости вероятности ошибки на блок от спектра графа Таннера для МППЧ-кодов

Направление подготовки (специальность): 01.04.02 Прикладная математика и информатика

Направленность (профиль): Технологии проектирования и разработки программного обеспечения

Квалификация: Магистр

2 Срок сдачи студентом законченной работы: «31» мая 2017 г.

3 Техническое задание и исходные данные к работе.

В рамках работы требуется исследовать зависимость между спектром графа Таннера МППЧ-кода и его эффективностью при декодировании. Для проведения исследования необходимо разработать эффективный алгоритм вычисления спектра, позволяющий провести отбор кодов с хорошим спектром среди широкого спектра сгенерированных кодов. Также необходимо разработать итеративный декодер, позволяющий измерить эффективность кода посредством симуляции передачи кодовых слов через канал с шумом.

4 Содержание магистерской диссертации (перечень подлежащих разработке вопросов)

а) Обоснование важности установления зависимости между спектром и эффективностью для исследования и отбора МППЧ-кодов;
б) Разработка и реализация итеративного декодера для быстрой оценки эффективности кода посредством моделирования;
в) Разработка и реализация алгоритма подсчета спектра графа Таннера;
г) Описание плана исследования. Порядок отбора кодов для проведения тестирования;
д) Результаты исследования.

5 Перечень графического материала (с указанием обязательного материала)

Не предусмотрено

6 Исходные материалы и пособия

а) Б.Д.Кудряшов. Основы теории кодирования;
б) М.Холл. Комбинаторика;
в) D.J.C.MacKay. Encyclopedia of Sparse Graph Codes. http://www.inference.phy.cam.ac.uk/mackay/codes/data.html.

7 Календарный план

№№ пп.	Наименование этапов магистерской диссертации	Срок выполнения этапов работы	Отметка о выполнении, подпись руков.
1	Ознакомление с основами теории кодирования	12.2015	
2	Ознакомление с имеющимся набором программ для исследования и отбора МППЧ-кодов	05.2016	
3	Ознакомление с существующими итеративными декодерами	07.2016	
4	Разработка и реализация итеративного декодера заточенного под нужды исследования	09.2016	
5	Ознакомление с существующими подходами подсчета спектра кода	11.2016	
6	Разработка и реализация алгоритма подсчета спектра графа Таннера МППЧ-кода	12.2016	
7	Проведение исследования зависимости эффективности кода от спектра	03.2017	
8	Написание пояснительной записки	05.2017	

8 Дата выдачи задания: «01» сентября 2015 г.

Руководитель _____

Задание принял к исполнению _____ «01» сентября 2015 г.

Министерство образования и науки Российской Федерации

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ»

АННОТАЦИЯ
МАГИСТЕРСКОЙ ДИССЕРТАЦИИ

Студент: Ковшаров Антон Павлович

Наименование темы работы: Исследование зависимости вероятности ошибки на блок от спектра графа Таннера для МППЧ-кодов

Наименование организации, где выполнена работа: Университет ИТМО

ХАРАКТЕРИСТИКА МАГИСТЕРСКОЙ ДИССЕРТАЦИИ

1 Цель исследования: Установить существование критерия оценки эффективности МППЧ-кода, основанного на спектре графа Таннера.

2 Задачи, решаемые в работе:

а) разработка и реализация эффективного алгоритма вычисления спектра графа Таннера;

б) проведение исследования зависимости эффективности кода от спектра графа Таннера;

в) формулировка критерия оценки эффективности МППЧ-кода.

3 Число источников, использованных при составлении обзора: _____

4 Полное число источников, использованных в работе: 14

5 В том числе источников по годам

Отечественных			Иностранных		
Последние 5 лет	От 5 до 10 лет	Более 10 лет	Последние 5 лет	От 5 до 10 лет	Более 10 лет

6 Использование информационных ресурсов Internet: _____

7 Использование современных пакетов компьютерных программ и технологий: C++, CUDA C — для создания итеративного декодера. Python и zsh скрипты для автоматизация исследования. Java — алгоритм подсчета спектра. Python, matplotlib, pandas — обработка и визуализация результатов. LaTeX, Git.

8 Краткая характеристика полученных результатов: В результате была продемонстрирована зависимость между спектром графа Таннера и эффективностью кода. Разработан вычислительно эффективный алгоритм подсчета спектра графа Таннера. Результаты могут быть использованы для ускорения поиска эффективных МППЧ-кодов.

9 Гранты, полученные при выполнении работы: Грантов или других форм государственной поддержи и субсидирования в процессе работы не предусматривалось.

10 Наличие публикаций и выступлений на конференциях по теме работы:

1 *Ковшаров А.*, *Кудряшов Б.* Исследование зависимости вероятности ошибки на блок от спектра графа Таннера для МППЧ-кодов // Сборник тезисов докладов конгресса молодых ученых. Электронное издание. — 2017.

2 *Ковшаров А.*, *Анохина И.* LDPC-codes frame error rate and Tanner's graph spectrum correlation research // Сборник тезисов докладов конгресса молодых ученых. Электронное издание. — 2017.

Выпусник: Ковшаров А.П. _____

Руководитель: Буздалов М.В. _____

«__» _____ 20__ г.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Интуиция подсказывает, что большой обхват и малое число коротких циклов положительно сказываются на эффективности итеративного декодирования.

Существуют и другие критерии для поиска хороших МППЧ кодов, например, ACE. В любом случае, все сводится к анализу структуры циклов графа. Поиск кодов, обладающих хорошей структурой – вычислительно сложная задача. Наша задача - упростить вычисления.

Известные подходы ....

Новый подход имеет простое описание и низкую вычислительную сложность

ГЛАВА 1. ОБЩИЕ СВЕДЕНИЯ

1.1. Линейные коды

Определение 1. *Линейным двоичным* (n,k)*—кодом* называется любое k—мерное подпространство пространства всевозможных двоичных векторов длины n.

Определение 2. Отношение $R = k/n$ называется *скоростью линейного* (n,k) *кода*.

Определение 3. *Порождающей матрицей линейного* (n,k)*—кода* называется матрица размера $k \times n$, строки которой его базисные вектора.

Пример 4.

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

задает код $000, 101, 100, 001$.

Определение 5. Двоичный вектор $\boldsymbol{h}$ длины n для которого все кодовые слова некоторого (n,k) кода $C = \{\boldsymbol{c}_i\}, i = 0, \ldots, 2^k - 1$ удовлетворяют тождеству

$$(\boldsymbol{c}_i, \boldsymbol{h}) = 0, i = 0, \ldots, 2^k - 1$$

называется *проверкой* по отношению к коду C.

Заметим, что предыдущее определение проверки эквивалентно

$$G \cdot \boldsymbol{h}^T = 0$$

так как для выполнения тождества для любого кодового слова достаточно выполнения тождества для базисных векторов.

Определение 6. Пространство проверок называется пространством, ортогональным линейному коду, или *проверочным пространством*.

Теорема 7. Размерность проверочного пространства линейного (n,k)—кода равна $r = n - k$.

Определение 8. *Проверочной матрицей линейного* (n,k)*—кода* называется матрица размера $r \times n$, строки которой составляют базис проверочного пространства

Для проверочной и порождающей матрицы выполнено следующее соотношение

$$G \cdot H^T = 0$$

Если же принятая последовательность $\boldsymbol{y}$ из-за шума в канале перестала быть кодовым словом то соответсвенно

$$\boldsymbol{s} = \boldsymbol{y} \cdot H^T \neq 0$$

и $\boldsymbol{s}$ называется *синдромом* вектора $\boldsymbol{y}$, неравенство нулевому вектору которого указывает на ошибки в принятой последовательности $\boldsymbol{y}$.

1.2. МППЧ-коды

Линейный код может быть задан проверочной матрицей H.

Галлагер [1] предложил идею выбора матрицы H разряженной (с малой плотностью) для уменьшения сложности кодирования и декодирования: в матрице должно быть мало единиц, строки и столбцы не должны иметь большое число общих элементов. Он также подкрепил свою идею анализом с использованием метода случайного декодирования.

Для интуитивного понимания почему малое число единиц приводит к более эффективному декодированию следует заметить, что в случае когда строки проверок мало между собой зависят, декодирование может производиться методом проб и ошибок, пытаясь подобрать последовательность символов, исправление которых будет уменьшать вес синдрома, с каждой следующей попыткой.

Используемый алгоритм декодирования, описанный далее, существенно опирается на факт, что влияние конкретного столбца на синдром не сильно зависит от остальных столбцов.

Определение 9. МППЧ-код называется (J,K) *регулярным* если его проверочная матрица H содержит ровно J единиц в каждом столбце и ровно K единиц в каждой строке. Иначе МППЧ-код называется *иррегулярным*.

1.3. Квазициклические МППЧ-коды

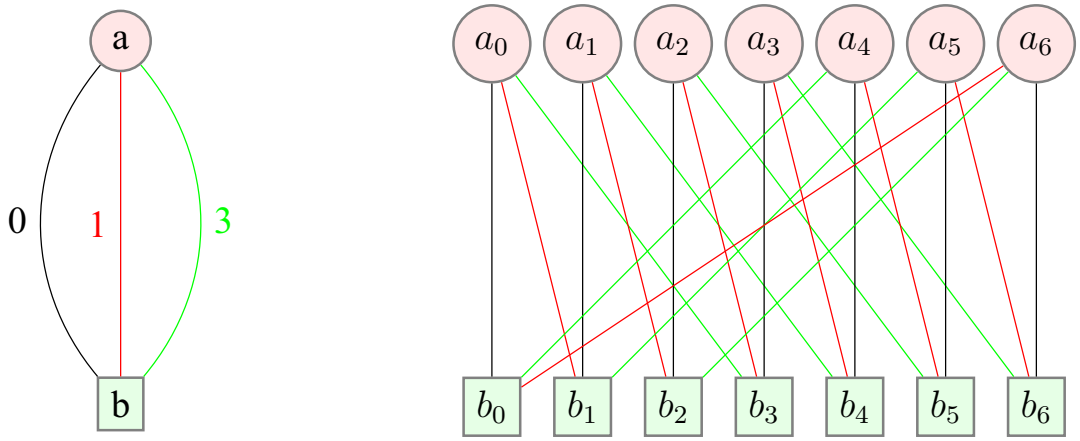
P_M^i — квадратная матрица порядка M, полученная из единичной сдвигом строк вправо i раз. Например:

$$P_3^0 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad P_3^1 = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix} \quad P_3^2 = \begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}$$

Матрицы P_M^i содержат одинаковые элементы на всех диагоналях параллельных главной — такие матрицы называются *циркулянтными матрицами* или *циркулянтами*.

МППЧ-код называется квазициклическим если его проверочная матрица может быть представлена в виде блочной матрицы из блоков циркулянтов единичной матрицы.

Для построения б



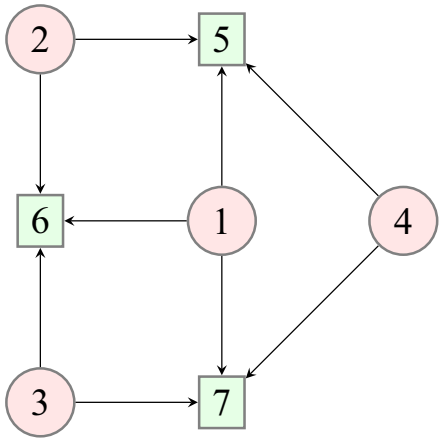
(a) Протограф (b) Расширенный граф

Рисунок 1 – Пример лифтинга

1.4. Граф Таннера

$$\begin{array}{cccc} 1 & 2 & 3 & 4 \\ \left(\begin{array}{cccc} 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \end{array}\right) & & & \begin{array}{c} 5 \\ 6 \\ 7 \end{array} \end{array}$$

(a) Проверочная матрица H



(b) Граф Таннера

Рисунок 2 – Пример графа Таннера

ГЛАВА 2. ДЕКОДЕР

Пример ссылок на литературные источники: [2–12].

Проведение экспериментального исследования зависимости эффективности итеративного декодирования от некоторого критерия подразумевает непосредственное моделирование передачи информации через канал с шумом посредством кодирования информации определенным МППЧ-кодом и последующим декодированием с помощью итеративного декодера.

Критерий, основанный на спектре циклов графа Таннера, побуждает к проведению исследования на иррегулярных МППЧ-кодах. Еще одним фактором не в пользу скорости декодирования является размер кода — который должен быть выбран в соотвествии с длинами кодов, используемыми на практике, которые достаточно велики. Кроме того, количество переданных кодовых слов при моделировании должны быть достаточно большим, чтобы оценка отношения ошибочно-переданных блоком к общему числу переданных блоков (FER - frame error rate) была достаточно точна.

Моделирование передачи достаточного количества кодовых слов занимает минуты на современных компьютерах, используя наивную реализацию декодера на CPU или реализацию из доступных библиотек (таких как например реализация из Communication System Toolbox в Matlab). С учетом того, что каждый код должен быть протестирован на десятках различных отношений сигнал-шум (SNR — signal noise ratio), подход с использованием наивной реализации при имеющихся вычислительных ресурсах займет недопустимо большое количество времени.

Альтернативным подходом являются реализации итеративных декодеров с использованием GPU. Многие из доступных реализаций заточены под МППЧ-коды из стандартов. Большая часть использует параллелизм на уровне декодирования одного кодового слова, что позволяет ускорить время между получением битовой последовательности и декодированного кодового слова [13], но не использует тот факт что при оценке вероятности ошибки на блок передается большое количество кодовых слов. В данном случае значительно более существенный выигрыш позволяет получить параллелизм на уровне набора кодовых слов, где вычислительные узлы разделены на группы, каждая из которых занимается декодированием различных кодовых слов [14].

К сожалению, итеративного GPU декодера, оптимизированного для получения оценки вероятности ошибки на блок, не было найдено в открытых источниках сети Internet. Что побудило к созданию простого итеративного декодера с использованием архитектуры CUDA, который позволил получить 60-70 кратное ускорение при моделировании передачи большого числа кодовых слов иррегулярного МППЧ-кода.

2.1. Количество кодовых слов при моделировании

В первую очередь следует провести анализ количества кодовых слов, которое следует передать, чтобы статистически достоверно оценить вероятность ошибки на блок.

Пусть P — вероятность ошибки на блок и передача производилась до достижения k ошибочно переданных блоков. Тогда суммарно передано $N \approx \frac{k}{P}$ кодовых слов. Оценим дисперсию среднего арифметического при N опытах. X_i — индикаторная случайная величина ошибки при передаче i-го кодового слова.

$$\begin{aligned}
\mathsf{D}\left(\frac{\sum_i X_i}{N}\right) &= \mathsf{E}\left(\frac{\sum_i X_i}{N}\right)^2 - \left(\mathsf{E}\left(\frac{\sum_i X_i}{N}\right)\right)^2 \\
&= \frac{\mathsf{E}(\sum_i X_i)^2}{N^2}) - \left(\frac{\sum_i \mathsf{E}(X_i)}{N}\right)^2 \\
&= \frac{\mathsf{E}(\sum_i X_i)^2}{N^2}) - \left(\frac{N \cdot P}{N}\right)^2 \\
&= \frac{\sum_{i \neq j} \mathsf{E}(X_i \cdot X_j) + \sum_i \mathsf{E}(X_i^2)}{N^2} - P^2 \\
&= \frac{N \cdot (N-1) \cdot P^2 + N \cdot P}{N^2} - P^2 \\
&= \frac{-N \cdot P^2 + N \cdot P}{N^2} \\
&= \frac{P \cdot (1-P)}{N} = \frac{P^2 \cdot (1-P)}{k}
\end{aligned}$$

Применяя Гауссовскую аппроксимацию биномиального распределения, можем говорить что пределы за $3 \cdot \sigma$ маловероятны (вероятность порядка 0.001). Поэтому погрешность находится в пределах (учитывая что в реальности $P \ll 1$):

$$\pm\delta = 3 \cdot P\sqrt{\frac{1-P}{k}} \approx P \cdot \frac{3}{\sqrt{k}}$$

Соответственно, например при $k = 50$ получаем относительную погрешность $\approx 42\%$, а при $k = 100 - \approx 30\%$. При увеличении k погрешность убывает обратно пропорционально корню. Для проведения дальнейших исследований остановимся на $k = 100$.

2.2. Построение порождающей матрицы по проверочной

Многие исследователи при моделировании передачи кодовых слов через канал останавливаются на многократной отправке нулевого кодового слова и его последующем декодировании. Однако, при таком подходе оценка эффективности МППЧ-кода оказывается существенно завышена относительно передачи различных кодовых слов при практическом использовании кода.

Для генерации большого числа различных кодовых слов необходимо иметь представление кода в виде порождающей матрицы. Имея представление (n,k) кода в виде проверочной матрицы H размера $r \times n$ и ранга $p = n - k$, порождающую матрицу кода G можно получить следующим образом:

а) С помощью двух проходов алгоритма Гаусса преобразуем матрицу H к матрице J того же размера, которая выглядит как:

$$J = \begin{pmatrix} I_p & 0 \\ 0 & 0 \end{pmatrix}$$

Действительно, с помощью первого прохода получаем матрицу с нулями ниже главной диагонали, записав линейное преобразование как матрицу P размера $r \times r$:

$$H_2 = P \cdot H = \begin{pmatrix} 1 & x_{1,2} & x_{1,3} & \dots & x_{1,p} & x_{1,p+1} & \dots & x_{1,n} \\ 0 & 1 & x_{2,3} & \dots & x_{2,p} & x_{2,p+1} & \dots & x_{2,n} \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & 1 & x_{p,p+1} & \dots & x_{p,n} \\ 0 & 0 & 0 & \dots & 0 & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & 0 & 0 & \dots & 0 \end{pmatrix}$$

Затем с помощью еще одного прохода H_2^T может быть приведен к J^T записав линейное преобразование в матрицу Q размера $n \times n$:

$$J^T = Q \cdot H_2^T$$

$$= Q \cdot \begin{pmatrix} 1 & 0 & \dots & 0 & 0 & \dots & 0 \\ x_{1,2} & 1 & \dots & 0 & 0 & \dots & 0 \\ x_{1,3} & x_{2,3} & \dots & 0 & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ x_{1,p} & x_{2,p} & \dots & 1 & 0 & \dots & 0 \\ x_{1,p+1} & x_{2,p+1} & \dots & x_{p,p+1} & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ x_{1,n} & x_{2,n} & \dots & x_{p,n} & 0 & \dots & 0 \end{pmatrix} = \begin{pmatrix} I_p & 0 & \dots & 0 \\ 0 & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & 0 \end{pmatrix}$$

Итого:

$$J = (Q \cdot H_2^T)^T = H_2 \cdot Q^T = P \cdot H \cdot Q^T$$

б) Запишем отношение между порождающей и проверочной матрицей:

$$G \cdot H^T = 0$$

или тоже самое:

$$H \cdot G^T = 0$$

Выражая H через J:

$$P^{-1} \cdot J \cdot (Q^T)^{-1} \cdot G^T = 0$$

где P^{-1} и $(Q^T)^{-1}$ обратные матрицы для P и Q^T соответственно. Обратные матрицы существуют так как проходы Гаусса сохраняли ранг матрицы, соответственно преобразование обратимо.

в) Обозначим $(Q^T)^{-1} \cdot G^T$ за Y (Y имеет размер $n \times k$), тогда:

$$J \cdot Y = 0$$

откуда следует, что Y имеет форму:

$$Y = \begin{pmatrix} 0 & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & 0 \\ y_{p+1,1} & y_{p+1,2} & \cdots & y_{p+1,k} \\ \dots & \dots & \dots & \dots \\ y_{n,1} & y_{n,2} & \cdots & y_{n,k} \end{pmatrix}$$

а

$$G^T = Q^T \cdot Y$$

Выбирая различные коэффициенты $y_{i,j}$ в G будут получатся различные наборы кодовых слов, не все из которых будут являться базисами. Для получения базисного набора векторов в G, необходимо и достаточно чтобы ранг G был равен k, так как $\det Q \neq 0$ соответственно ранг Y тоже должен быть равен k.

Одним из подходящих вариантов Y является следующий, ранг которого равен, очевидно, k (так как $n - k = p$):

$$Y = \begin{pmatrix} 0 \\ I_k \end{pmatrix}$$

Итого:

$$G = \begin{pmatrix} 0 & I_k \end{pmatrix} \cdot Q$$

2.3. Алгоритм декодирования

В качестве алгоритма декодирования был выбран один из множества разновидностей алгоритмов декодирования по принципу распространения доверия — LLR-SPA (Sum-Product Algorithm in Log-Likelihood domain — Алгоритм суммирования произведений в терминах логарифмов).

Алгоритм суммирования произведений входит в семейство алгоритмов передачи сообщений, которые позволяют организовать исходные данные в памяти таким образом, чтобы можно было организовать параллельную обработку.

Переход к логарифмам был необходим в первую очередь потому, что предполагалась реализация декодера на архитектуре CUDA, которая, за ис-

ключением видеокарт для научных вычислений Tesla, предоставляет только возможность вычисления в типах данных с плавающей точкой одинарной точности. Соответственно переход к логарифмам позволил получить приемлемую точность. Кроме того использование логарифмов позволяет использовать более простые операции, например, сложения вместо умножения.

Вкратце опишем алгоритм LLR-SPA.

2.3.1. LLR-SPA

Для простоты записи предположим, что мы рассматриваем (J,K)— регулярный МППЧ-код. Обозначим событие что все J проверочных соотношений выполнены за S, а выполнение каждой из проверок соответственно за S_i.

Основное предположение, которое, вообще говоря, не верно при наличии циклов в графе Таннера, говорит о том, что проверки независимы. То есть

$$\Pr(S) = \prod_i \Pr(S_i)$$

Соответственно соотношение о независимости проверок выполняется тем больше, чем ниже плотность единиц в проверочной матрице кода, а также зависит от многих других критериев, один из которых как раз рассмотрен в данной работе.

Пусть $\boldsymbol{x}$ передаваемое кодовое слово, соответственно x_i — i-ый символ кодового слова. Обозначим за $\boldsymbol{y}$ принятую последовательность, за p_i вероятность единицы на i-ом символе принятой последовательности, основываясь на мягком входе декодера, то есть $p_i = \Pr(x_i = 1|\boldsymbol{y})$.

Рассмотрим основное утверждение, которое позволит сформулировать LLR-SPA как алгоритм передачи сообщений.

Теорема 10. [3] Если результаты различных проверок, в которые входит данный символ — независимые события, а $p_{jh}, h = 1,\ldots,K-1, j = 1,\ldots,J$— вероятность единицы для h-го символа j-й проверки при заданной последовательности y, то

$$\frac{\Pr(x_i = 0|\boldsymbol{y},S)}{\Pr(x_i = 1|\boldsymbol{y},S)} = \frac{1-p_i}{p_i} \cdot \prod_{j=1}^{J} \frac{1 + \prod_{h=1,col(j,h)!=i}^{K}(1-2p_{jh})}{1 - \prod_{h=1,col(j,h)!=i}^{K}(1-2p_{jh})} \qquad (1)$$

где $col(j,h)$ — обозначает номер столбца h-ой единицы в j-ой проверке.

Лемма 11. [3] Пусть в последовательности из n двоичных символов вероятность единицы в позиции i равна p_i. Тогда вероятность четного числа единиц в последовательности равна

$$P_{even} = \frac{1 + \prod_{i=1}^{n}(q_i - p_i)}{2}, q_i = 1 - p_i \tag{2}$$

Доказательство. Рассмотрим бином:

$$\prod_{i=1}^{n}(q_i - p_i)$$

отрицательные слагаемые в котором соответствуют вероятностям последовательностей с нечетным числом единиц, а положительные с четным. Соответственно:

$$P_{even} - P_{odd} = \prod_{i=1}^{n}(q_i - p_i)$$

а

$$P_{even} + P_{odd} = 1$$

Таким образом складывая предыдущие выражения для получения P_{even} получаем:

$$P_{even} = \frac{1 + \prod_{i=1}^{n}(q_i - p_i)}{2}, q_i = 1 - p_i$$

□

Доказательство. По теореме Байеса:

$$\Pr(x_i = 0|\boldsymbol{y},S) = \frac{\Pr(S|\boldsymbol{y},x_i = 0) \cdot \Pr(x_i = 0|\boldsymbol{y})}{\Pr(S|\boldsymbol{y})} \tag{3}$$

$$\Pr(x_i = 1|\boldsymbol{y},S) = \frac{\Pr(S|\boldsymbol{y},x_i = 1) \cdot \Pr(x_i = 1|\boldsymbol{y})}{\Pr(S|\boldsymbol{y})} \tag{4}$$

После подстановки:

$$\frac{\Pr(x_i = 0|\boldsymbol{y},S)}{\Pr(x_i = 1|\boldsymbol{y},S)} = \frac{\Pr(S|\boldsymbol{y},x_i = 0) \cdot \Pr(x_i = 0|\boldsymbol{y})}{\Pr(S|\boldsymbol{y},x_i = 1) \cdot \Pr(x_i = 1|\boldsymbol{y})} \tag{5}$$

Так как $\Pr(x_i = 0|\boldsymbol{y}) = 1 - p_i$, а $\Pr(x_i = 1|\boldsymbol{y}) = p_i$:

$$\frac{\Pr(x_i = 0|\boldsymbol{y},S)}{\Pr(x_i = 1|\boldsymbol{y},S)} = \frac{1 - p_i}{p_i} \cdot \frac{\Pr(S|\boldsymbol{y},x_i = 0)}{\Pr(S|\boldsymbol{y},x_i = 1)} \tag{6}$$

В первом случае:

$$\Pr(S|\boldsymbol{y},x_i = 0) = \prod_{j=1}^{J} \Pr(S_j|\boldsymbol{y},x_i = 0) \tag{7}$$

Соответственно S_j выполняется, когда $\sum_{k!=i} x_k$ четна, то есть:

$$\Pr(S_j|\boldsymbol{y},x_i = 0) = \frac{1 + \prod_{h=1,col(j,h)!=i}^{K}(1 - p_{jh} - p_{jh})}{2}$$

Во втором случае:

$$\Pr(S|\boldsymbol{y},x_i = 1) = \prod_{j=1}^{J} \Pr(S_j|\boldsymbol{y},x_i = 1) \tag{8}$$

Здесь S_j выполняется, когда $\sum_{k!=i} x_k$ нечетна, то есть:

$$\Pr(S_j|\boldsymbol{y},x_i = 0) = \frac{1 - \prod_{h=1,col(j,h)!=i}^{K}(1 - p_{jh} - p_{jh})}{2}$$

Собирая все вместе получаем утверждение теоремы. □

Переформулируем формулы в терминах логарифмов. Для этого введем обозначение α для знака логарифма правдоподобия и β для абсолютной величины, со следующими индексами:

$$\ln \frac{1 - p_i}{p_i} = \alpha_i \beta_i$$

$$\ln \frac{1 - p_{jh}}{p_{jh}} = \alpha_{jh} \beta_{jh}$$

Введем функцию f:

$$f(\beta) = \ln \frac{e^{\beta} + 1}{e^{\beta} - 1}, \beta > 0$$

Заметим, что

$$f(\beta) = -\ln \frac{e^{\beta/2} - e^{-\beta/2}}{e^{\beta/2} + e^{-\beta/2}} = -\ln \tanh(\beta/2),$$

где

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} = \frac{e^{2x} - 1}{e^{2x} + 1}$$

Получаем:

$$1 - 2p = \frac{q - p}{q + p} = \frac{q/p - 1}{q/p + 1} = \tanh\left(\frac{\alpha\beta}{2}\right) = \alpha e^{-f(\beta)}$$

так как $\tanh(\alpha\beta) = \alpha \tanh(\beta)$.

Подставляя полученное выражение в основное тождество:

$$\alpha_i'\beta_i' = \alpha_i\beta_i + \sum_{j=1}^{J} \ln \frac{1 + \prod_{h=1,col(j,h)!=i}^{K}(1 - 2p_{jh})}{1 - \prod_{h=1,col(j,h)!=i}^{K}(1 - 2p_{jh})}$$

Из предыдущего замечания получаем

$$\prod_{h=1,col(j,h)!=i}^{K} (1 - 2p_{jh}) = \prod_{h=1,col(j,h)!=i}^{K} \alpha_{jh} \cdot e^{-\sum_{h=1,col(j,h)!=i}^{K} f(\beta_{jh})}$$

Обозначим $\prod_{h=1,col(j,h)!=i}^{K} \alpha_{jh}$ за A_{ij}, а $e^{-\sum_{h=1,col(j,h)!=i}^{K} f(\beta_{jh})}$ за T_{ij}. Тогда:

$$\alpha_i'\beta_i' = \alpha_i\beta_i + \sum_{j=1}^{J} \ln \frac{1 + A_{ij}T_{ij}}{1 - A_{ij}T_{ij}}$$

Заметим некоторые полезные свойства функции f:

$$f(\beta) = \ln \frac{e^\beta + 1}{e^\beta - 1} = \ln \frac{1 + e^{-\beta}}{1 - e^{-\beta}} = -\ln \frac{1 - e^{-\beta}}{1 + e^{-\beta}}, \beta > 0$$

Кроме того, если $\beta > 0$, тогда $f(\beta) > 0$.

Таким образом $0 < T_{ij} < 1$.

Если $A_{ij} = 1$, тогда:

$$\ln \frac{1 + A_{ij}T_{ij}}{1 - A_{ij}T_{ij}} = \ln \frac{1 + T_{ij}}{1 - T_{ij}} = f(-\ln T_{ij})$$

Иначе если $A_{ij} = -1$, тогда:

$$\ln \frac{1 + A_{ij}T_{ij}}{1 - A_{ij}T_{ij}} = \ln \frac{1 - T_{ij}}{1 + T_{ij}} = -f(-\ln T_{ij})$$

Или объединяя оба случая:

$$\ln \frac{1 + A_{ij}T_{ij}}{1 - A_{ij}T_{ij}} = A_{ij} \cdot f(-\ln T_{ij})$$

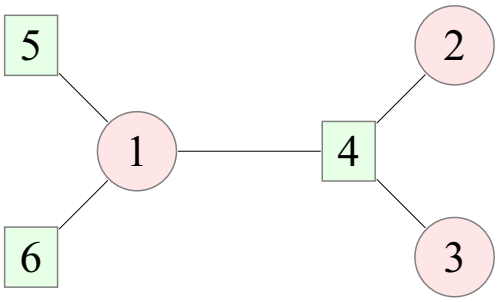
Подставляя назад получаем:

$$\alpha'_i\beta'_i = \alpha_i\beta_i + \sum_{j=1}^{J} A_{ij} \cdot f(-\ln T_{ij}) =$$

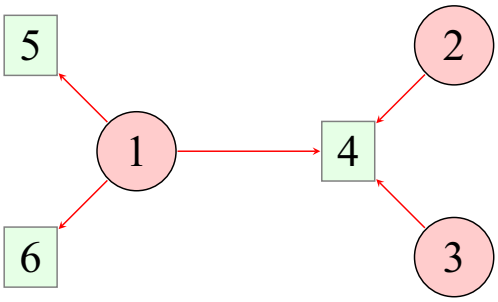
$$= \alpha_i\beta_i + \sum_{j=1}^{J} \prod_{h=1,col(j,h)!=i}^{K} \alpha_{jh} \cdot f(\sum_{h=1,col(j,h)!=i}^{K} f(\beta_{jh}))$$

2.3.2. LLR-SPA как алгоритм передачи сообщений

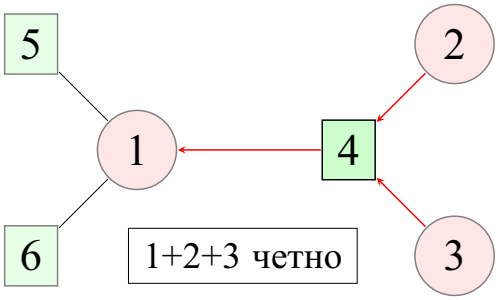
Рассмотрим некоторый граф Таннера (см. рис. 3а).



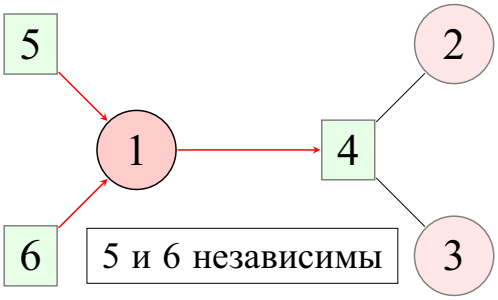
(a) граф Таннера



(b) Передача начальных приближений



(c) Передача от проверочных узлов к символьным



(d) Передача от символьных узлов к проверочным

Рисунок 3 – Пример передачи сообщений в графе Таннера

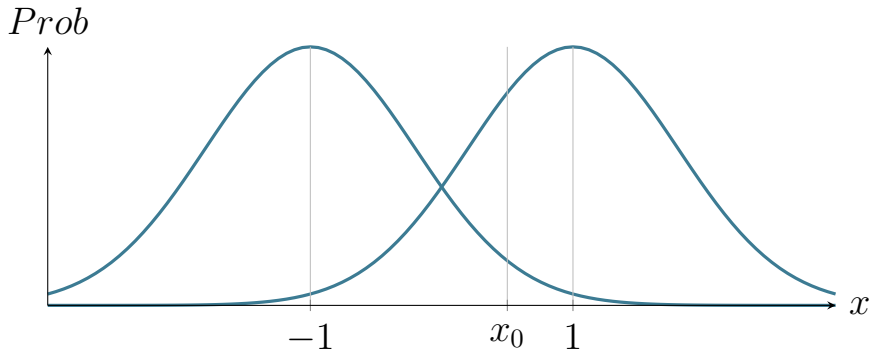


Рисунок 4 – Распределение вероятности в канале с шумом

1. На первом шаге алгоритма символьные узлы передают начальные приближения о полученных символах последовательности из канала $\alpha_i\beta_i$ проверочным узлам (см. рис. 3b).
   Для получения логарифма правдоподобия $\alpha_i\beta_i = \ln\frac{1-p}{p}$ по полученному значению x_0 из канала необходимо провести преобразование. По договоренности единичные биты передаются значением -1, а нулевые значением 0. Шум в канале размывает вероятность в колокол нормального распределения. Соответственно вероятность p переданной единицы составляет вероятность "принадлежности"левому колоколу на картинке, а $q = 1 - p$ соответственно правому.
   Рассмотрим соотношение плотностей нормального распределение в произвольной точке x_0 и получим формулу получения логарифма правдопадобия:

$$\ln\frac{1-p}{p} = \ln\frac{\frac{1}{\sigma\sqrt{2\pi}}e^{-\frac{(x_0-1)^2}{2\sigma^2}}}{\frac{1}{\sigma\sqrt{2\pi}}e^{-\frac{(x_0+1)^2}{2\sigma^2}}} = \ln\frac{e^{-\frac{(x_0-1)^2}{2\sigma^2}}}{e^{-\frac{(x_0+1)^2}{2\sigma^2}}} = \ln e^{-\frac{(x_0-1)^2}{2\sigma^2}+\frac{(x_0+1)^2}{2\sigma^2}} =$$

$$= \frac{-(x_0-1)^2+(x_0+1)^2}{2\sigma^2} = \frac{-x_0^2+2x_0-1+x_0^2+2x_0+1}{2\sigma^2} = \frac{2x_0}{\sigma^2}$$

2. На втором шаге каждый из проверочных узлов на основе полученной информации от символьных узлов и факта, что сумма должна быть четно. Таким образом проверочный узел j подсчитывает и отправляет новую оценку для символьного узла i основываясь на других узлах в проверке

(см. рис. 3c):

$$Z(j,i) = \prod_{h=1,col(j,h)!=i}^{K} \alpha_{jh} \cdot f(\sum_{h=1,col(j,h)!=i}^{K} f(\beta_{jh})) \tag{9}$$

3. На третьем шаге каждый из символьных узлов i комбинирует информацию полученную от проверочных узлов, считая что проверки независимы (что неверно в общем случае), и отсылает каждому смежному проверочному узлу j информацию от других проверочных узлов в уравнения которых этот символ также входит (см. рис. 3d):

$$\alpha_{ji}\beta_{ji} = L(j,i) = \alpha_i\beta_i + \sum_{h=1,row(h,i)!=j}^{J} Z(h,i) \tag{10}$$

4. На четвертом шаге исходя из всей имеющейся информации в символьных узлах строится текущее предполагаемая переданная последовательность:

$$\widetilde{y}_i = \alpha_i\beta_i + \sum_{h=1}^{J} Z(h,i) \tag{11}$$

$$x_i = \widetilde{y}_i < 0$$

Если вектор $\boldsymbol{x}$ жестких решений является кодовым словом, что может быть проверено с помощью умножения на проверочную матрицу H, алгоритм декодирования завершается успешно. В случае моделирования это может быть проверено быстрее — простым сравнением с переданным кодовым словом.

После чего если не достигнуто максимальное число итераций, шаги 2-4 повторяются. Типичное используемое число итераций, которое также использовано в данном исследовании — 50.

Если после максимального числа итераций кодовое слово до сих пор не получено — фиксируется ошибка при передаче и алгоритм завершается.

2.4. Реализация алгоритма декодирования на GPU

При переносе алгоритма на GPU используется два уровня распараллеливания вычислений. Первый уровень на основе разделения по кодовым словам, что очень хорошо ложится на архитектуру блоков и потоков, используемую

в CUDA. Второй уровень разделения на уровне ребер — каждая сумма по смежным ребрам из выражений 9, 10, 11 при передаче сообщения по ребру считается отдельным потоком из пула потоков.

Таким образом на первом уровне, каждый из блоков занимается декодированием одного кодового слова. Потоки в блоке могут синхронизировать свое исполнение, что используется после окончания каждого шага алгоритма. Также потоки в блоке могут использовать общую память. Так как при декодировании различных кодовых слов экземпляры декодеров не взаимодействуют между собой, изолированность при работе различных блоков является большим плюсом, позволяя не тратить время на синхронизацию. Экспериментальным путем было решено использовать 100 блоков, таким образом параллельно декодируя 100 кодовых слов.

Для эффективного подсчета сумм из выражений 9, 10, 11 данные необходимо правильно расположить в памяти для попадания в кэш, так как работа с памятью один из самых узких участков в алгоритме.

Для этого продублируем информацию о ребрах, первый раз расположив ее таким образом, чтобы ребра, соответствующие символам из одной проверки, находились последовательно, для попадания в кэш при подсчете выражения 9. Значения $\alpha_{jh}\beta_{jh}$ с одинаковым j также должны быть расположены последовательно. Этого можно добиться построчным обходом матрицы H, последовательной записью соответствующих ребер в участок памяти и индексов границ ребер из соответствующих проверок в памяти.

Второй раз информацию о ребрах необходимо расположить так, чтобы ребра соответствующий одному символьному узлу лежали последовательно, для быстрого вычисления выражений 10 и 11. Кроме того, значения $Z(h,i)$ с одинаковым i также должны быть расположены последовательно. Этого можно добиться обходом матрицы H по столбцам с последовательной записью соответствующих ребер и индексов где начинаются и заканчиваются ребра, соответствующие тому же символьному узлу в записанном блоке памяти.

Таким образом для вычисления $Z(j,i)$,$L(j,i)$,$\widetilde{y_i}$ из выражений 9, 10 и 11 берется отдельный поток из пула потоков, который с помощью прохода по соответствующей области памяти со смежными ребрами выполняет необходимые операции. Экспериментальным путем размер пула потоков установлен равным 512. Производительность оптимизировалась под видеокарту на кото-

рой производились эксперименты — GeForce GTX 780 3Gb. После выполнения каждого шага производилась синхронизация потоков в блоке. После подсчета выражения 9 производилось округление слишком больших значений $Z(j,i)$ по абсолютному значение до значения ± 13.07. Такая константа выбрана из тех соображений, что любое значение $\leqslant 10$ при используемых размерах кодов не дает достаточной точности, а значение $\geqslant 16$ приводит к переполнению при проведении вычислений в типе данных с плавающей точкой одинарной точности.

На экспериментах из исследования реализованный алгоритм позволил получить почти 60-кратное ускорения по сравнению с версиями на CPU. Исходный код декодера может быть найден в сети интернет https://github.com/antonkov/CUDA-LDPC-BER-Simulator

2.5. Подсчет эффективности кода

Стандартный способ оценки эффективности кода — построения графика вероятности ошибки на блок от отношения сигнал-шум.

При фиксированной величине отношения сигнал-шум для оценки вероятности ошибки на блок необходимо отсылать кодовые слова до достижения некоторого порогового значения количества ошибок на блок — в нашем случае $k = 100$, как было обсуждено ранее. В силу того, что разработанный декодер обрабатывает несколько полученных кодовых последовательностей одновременно, декодирование будет производиться блоками по $D = 100$ кодовых слов (наиболее оптимальное количество кодовых слов в блоке для одновременного декодирования установлено экспериментальным путем). Проверка достижения порогового количества ошибок будет происходить после обработки каждого блока.

Таким образом при фиксированном значении отношения сигнал-шум подсчет эффективности производится следующим образом:

а) Вычислить порождающую матрицу G по проверочной матрице H.

б) Сгенерировать достаточное количество кодовых слов для подсчета эффективности. Было решено генерировать $D = 100$ различных кодовых слов, то есть количество равное числу слов в блоке. Таким образом набор кодовых слов в блоках будет одинаковым, но переданные последовательности будут разными за счет различных шумовых данных. Каждое случайно сгенерированное кодовое слово является линейной комбина-

цией базисных кодовых слов, где каждое базисное кодовое слово взято с вероятностью $\frac{1}{2}$. Генератор случайных чисел инициализируется одинаковым случайным значением, соответственно подсчет эффективности проводится на одинаковой последовательности кодовых слов для фиксированного кода на разных отношениях сигнал-шум.

в) Сгенерировать шум и прибавить шумовые данные к кодовым словам. Исходя из определения отношения сигнал-шум:

$$SNR(dB) = 10 \log_{10} \left(\frac{A^2_{signal}}{A^2_{noise}} \right)$$

где A_{signal},A_{noise} — среднеквадратичное значение амплитуды сигнала и шума соответственно.

Принимая амплитуды сигнала за ± 1 как у Галлагера считаем, что $A_{signal} = 1$. Соответственно:

$$A^2_{noise} = 10^{-\frac{SNR}{10}}$$

Таким образом для моделирования шума с заданным отношением SNR необходимо сгенерировать данные с нормальным распределением со средним равным 0 и дисперсией $\sigma^2 = A^2_{noise}$.

г) Производить декодирование по блокам с последующим измерением числа ошибок до достижения порогового значения количества ошибок.

В данном исследовании нас будет интересовать интервал отношения сигнал-шум от 1 дБ до 3.5 дБ, так как большинство случайных кодов достигают вероятности ошибки на блок 10^{-3} для исследуемых размеров ранее 3.5 дБ. Кроме того заметим, что нет необходимости измерять вероятность ошибки для кода при большем отношении сигнал-шум, если при текущем отношении код настолько хорош, что ошибка оказывается менее некоторой пороговой, например 10^{-3}. Более того это еще и очень долго, так как вероятность ошибки при увеличении отношения сигнал-шум может уменьшаться экспоненциально.

Шаг между отношениями сигнал шум выбран равным 0.1, опять же из соображений того, что вероятность может убывать очень быстро и при большом шаге будет настолько мала, что достижение достаточного числа ошибок

займет существенное время. Использование более малого шага нецелесообразно, так как вероятность уменьшается не так существенно.

Иногда представление эффективности кода в виде графика зависимости от отношения сигнал-шум неудобно, так как в таком представлении различные коды могут быть несравнимы. Значение отношения сигнал-шум S_0 при котором вероятность ошибки на блок впервые становится менее некоторого заданного значения, например 10^{-3}, дает гораздо более компактное представление об эффективности кода и может быть использовано для сравнения эффективности различных кодов.

Подсчет величины S_0 можно организовать на основе построения графика эффективности кода от отношения сигнал-шум до достижения необходимой вероятности, однако подсчет может быть организован гораздо эффективнее.

Заметим, что вероятность ошибки монотонно убывает при росте отношения сигнал-шум, соответственно можно использовать бинарный поиск по отношению сигнал-шум для нахождения S_0. Однако при данном подходе подсчет вероятности при значениях больше S_0 может занимать существенное время, так как мы знаем что вероятность ошибки может убывать экспоненциально. Для экономии времени при обработке слишком больших отношении сигнал-шум ограничим количество кодовых слов для отправки.

Обозначим текущее отношение сигнал-шум за S, вероятность ошибки при S_0 за FER_{target} (где FER — Frame Error Rate), полученное количество ошибок на блок за k, пороговое значение количества ошибок на блок за k_0, количество переданных слов за N. Тогда нет необходимости пересылать более N кодовых слов:

$$N = \frac{2 \cdot k_0}{FER_{target}}$$

Оценим вероятность ошибочного определения положения S относительно S_0.

Действительно, при условии что $S > S_0$ ограничение на количество переданных кодовых слов при недостижении заданного порога количества ошибок только увеличивает вероятность правильного определения предиката нахождения S относительно S_0.

При условии $S < S_0$ и передаче N кодовых слов матожидание k не менее $2 \cdot k_0$. Вспомним, что при $k_0 = 100$ среднеквадратичное отклонение составляет $\sigma = 0.1$ и соответственно вероятность, что $k > 2 \cdot k_0 - 3 \cdot \sigma \cdot 2 \cdot k_0 = 1.4 \cdot k_0$ более

0.999, а вероятность, что $k > 2 \cdot k_0 - 5 \cdot \sigma \cdot 2 \cdot k_0 = k_0$ более 0.999999997. Таким образом вероятность, протестировав N слов, получить менее k_0 ошибок при отношении сигнал-шум $S < S_0$ составляет менее $3 \cdot 10^{-9}$, что достаточно мало.

В итоге для нахождения S_0 с помощью бинарного поиска достаточно порядка 10 итераций при начальных границах от 1 дБ до 5 дБ.

ГЛАВА 3. АЛГОРИТМ ПОДСЧЕТА СПЕКТРА ГРАФА ТАННЕРА

Эта задача порождена проблемой анализа и оптимизации МППЧ кодов. Пусть B – (двоичная) базовая матрица кода. Для нее определен двудольный граф Таннера $T = \{V, E\}$ с множеством вершин $V = V_s \cup V_c$, где V_s и V_c – множества символьных и проверочных вершин, соответственно. Единицам матрицы B соответствуют ребра графа T.

Пример 12. Рассмотрим базовую матрицу B

$$B(D) = \begin{pmatrix} 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \end{pmatrix}, \tag{12}$$

для задающей квазициклический МППЧ код полиномиальной проверочной матрицы

$$H(D) = \begin{pmatrix} D^{w_{11}} & D^{w_{12}} & 0 & D^{w_{14}} \\ D^{w_{21}} & D^{w_{22}} & D^{w_{23}} & 0 \\ D^{w_{31}} & 0 & D^{w_{33}} & D^{w_{34}} \end{pmatrix} \tag{13}$$

Для удобства переназначим веса переходов

$$H(D) = \begin{pmatrix} D^{w_1} & D^{w_4} & 0 & D^{w_8} \\ D^{w_2} & D^{w_5} & D^{w_6} & 0 \\ D^{w_3} & 0 & D^{w_7} & D^{w_9} \end{pmatrix} \tag{14}$$

Соответствующая матрица инцидентности графа Таннера

$$T(D) = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ D^{w_1} & 0 & 0 & D^{w_4} & 0 & 0 & 0 & D^{w_8} & 0 \\ 0 & D^{w_2} & 0 & 0 & D^{w_5} & D^{w_6} & 0 & 0 & 0 \\ 0 & 0 & D^{w_3} & 0 & 0 & 0 & D^{w_7} & 0 & D^{w_9} \end{pmatrix}$$

Сам граф показан на рис. 6. Кружками и квадратами показаны символьные и проверочные узлы.

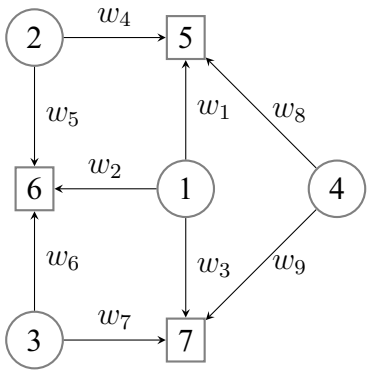


Рисунок 5 – Граф Таннера для кода из примера 12

Задача состоит в подсчете числа циклов заданной длины в расширенном графе с заданной степенью расширения M. Длина цикла равна числу переходов в пути, начинающемся и заканчивающемся в одном и том же узле и таком, что сумма весов переходов по модулю M равна нулю. Веса суммируются с учетом знаков, зависящих от направления перехода (см. рис. 6).

Такой тип циклов часто называют замкнутым обходом, однако при подсчете рассматриваемых объектов необходимо учесть следующие дополнительные ограничения.

— Запрещено двигаться обратно по последнему пройденному ребру. Например, путь 6→1→6 веса 0 запрещен.

— Циклические сдвиги путей и инверсии путей должны учитываться как один цикл. Например, путь $p = 6 \to 1 \to 7 \to 3 \to 6$ образует цикл при условии $w_2 - w_6 + w_7 - w_3 = 0$. При этом пути $6 \to 3 \to 7 \to 1 \to 6$ и $1 \to 7 \to 3 \to 6 \to 1$ тоже циклы, но они уже не вносят вклад в число циклов длины 4, если цикл p учтен.

Хотелось бы применить стандартные методы, используемые при анализе систем на основе конечных автоматов. Данный граф не является конечным автоматом, поскольку перемещение из состояния в состояние зависит от предыдущего состояния. Например, на рис. 6 после состояния 4 возможно только 5, если предыдущим было 7 и, наоборот, только 7, если предыдущим было 5.

Чтобы свести задачу к анализу конечных автоматов, введем новое множество состояний $U = \{e,\xi\}$, где e задает ребро исходного графа, а ξ – направление перехода. Сокращенно будем записывать пары в виде $+e$ и $-e$.

Из графа на рис. 6 получится граф с 18 состояниями $\{\pm 1, \pm 2, ..., \pm 9\}$. Заметим, однако, что после отрицательного ребра следуют только положитель-

ные и после положительного отрицательные. Это позволит записать матрицу переходов компактно в виде двух матриц, матрицы положительных и матрицы отрицательных переходов. Например, следующими состояниями (ребрами графа Таннера) после положительного перехода $+1$ возможны отрицательные $-4, -8$. После отрицательного перехода -1 возможны положительные 2,3.

В нашем примере две матрицы переходов имеют вид

$$A_- = \begin{pmatrix} 0 & 0 & 0 & D^{-w_4} & 0 & 0 & 0 & D^{-w_8} & 0 \\ 0 & 0 & 0 & 0 & D^{-w_5} & D^{-w_6} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & D^{-w_7} & 0 & D^{-w_9} \\ D^{-w_1} & 0 & 0 & 0 & 0 & 0 & 0 & D^{-w_8} & 0 \\ 0 & D^{-w_2} & 0 & 0 & 0 & D^{-w_6} & 0 & 0 & 0 \\ 0 & D^{-w_2} & 0 & 0 & D^{-w_5} & 0 & 0 & 0 & 0 \\ 0 & 0 & D^{-w_3} & 0 & 0 & 0 & 0 & 0 & D^{-w_9} \\ D^{-w_1} & 0 & 0 & D^{-w_4} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & D^{-w_3} & 0 & 0 & 0 & D^{-w_7} & 0 & 0 \end{pmatrix}$$

$$A_+ = \begin{pmatrix} 0 & D^{w_2} & D^{w_3} & 0 & 0 & 0 & 0 & 0 & 0 \\ D^{w_1} & 0 & D^{w_3} & 0 & 0 & 0 & 0 & 0 & 0 \\ D^{w_1} & D^{w_2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & D^{w_5} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & D^{w_4} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & D^{w_7} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & D^{w_6} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & D^{w_9} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & D^{w_8} & 0 \end{pmatrix}$$

Заметим, что все циклы имеют четную длину и состоят из пар переходов (по стрелке, против стрелки). Можно построить матрицу переходов за два

шага, как произведение матриц A_+, A_-

$$A = A_+A_- = \begin{pmatrix} 0 & 0 & 0 & 0 & \omega_{45} & 0 & 0 & 0 & \omega_{89} \\ 0 & 0 & 0 & \omega_{54} & 0 & 0 & \omega_{67} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \omega_{76} & 0 & \omega_{98} & 0 \\ 0 & \omega_{12} & \omega_{13} & 0 & 0 & 0 & 0 & 0 & \omega_{89} \\ \omega_{21} & 0 & \omega_{23} & 0 & 0 & 0 & \omega_{67} & 0 & 0 \\ \omega_{21} & 0 & \omega_{23} & \omega_{54} & 0 & 0 & 0 & 0 & 0 \\ \omega_{31} & \omega_{32} & 0 & 0 & 0 & 0 & 0 & \omega_{98} & 0 \\ 0 & \omega_{12} & \omega_{13} & 0 & \omega_{45} & 0 & 0 & 0 & 0 \\ \omega_{31} & \omega_{32} & 0 & 0 & 0 & \omega_{76} & 0 & 0 & 0 \end{pmatrix}$$

где $\omega_{ij} = D^{-w_i+w_j}$.

В общем случае матрица переходов за 2 шага будет иметь размер, равный числу единиц в базовой матрице B.

Для подсчета производящей функции числа путей длины $2L$, начинающихся с ребра 1 в положительном направлении нужно начальный вектор $a_0 = (1,0,...,0)$ умножить на A^L. Для нашего примера при

$$\boldsymbol{a}_2 = \boldsymbol{a}_0 A = \begin{pmatrix} 0 & 0 & 0 & 0 & \omega_{45} & 0 & 0 & 0 & \omega_{89} \end{pmatrix} \tag{15}$$

$$\boldsymbol{a}_4 = \boldsymbol{a}_2 A = \begin{pmatrix} \omega_{4521,8931} & \omega_{8932} & \omega_{4523} & 0 & 0 & \omega_{8976} & \omega_{4567} & 0 & 0 \end{pmatrix} \tag{16}$$

где первая компонента является сокращенной записью полинома

$$\omega_{4521,8931} = D^{-w_4+w_5-w_2+w_1} + D^{-w_8+w_9-w_3+w_1}$$

Кроме того необходимо отметить:

- Соблюдение условия неповторения ребра на стыке цикла также соблюдается.
- Хотя мы пишем $+, -$ в выражениях типа $D^{-w_4+w_5-w_2+w_1}$, w_i не коммутируют между собой так как неупорядоченный набор ребер не задает однозначно цикл (13), таким образом до подстановки конкретных значений w_i не могут быть сложены.

Пример 13. Обозначим пути $c_1 = 1 \to -4 \to 5 \to -2$, $c_2 = 1 \to -4 \to 6 \to -3$, $c_3 = 2 \to -5 \to 6 \to -3$. Обратные к этим путям по

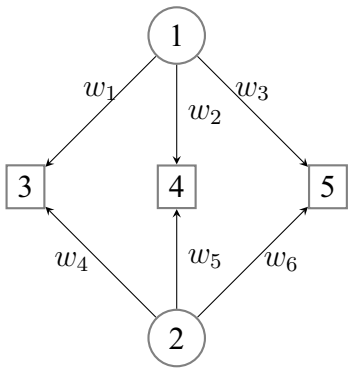


Рисунок 6 – Граф Таннера для кода из примеров 13, 14

направлению соответственно c_{-1}, c_{-2}, c_{-3}. Тогда набор ребер как объединением $\{c_1,c_{-2},c_{-3}\}$ может обозначать два пути – $c_1c_{-3}c_{-2}$ и $c_1c_{-2}c_{-3}$, которые не могут быть получены друг из друга с помощью циклического сдвига и инверсии.

Из (20) видим, что существуют 2 потенциальных цикла длины 4. Цикл образуется в том случае, когда сумма в показателе степени равна нулю. Например, если $-w_4 + w_5 - w_2 + w_1 = 0$, а $-w_8 + w_9 - w_3 + w_1 \neq 0$ то

$$a_{41}(D) = 1 + D^{-w_8+w_9-w_3+w_1},$$

и $a_{41}(0) = 1$. Пусть $a^j_{2L,i}$ – обозначает коэффициент при D^j в i-ой компоненте a_{2L}.

(здесь и далее считаем, что $D^0 = 1$, несмотря на то что D может быть 0. Таким образом свободный член полинома равен количеству циклов веса 0 без учета эквивалентных циклов) В общем случае все циклы длины $2L$, проходящие через ребро +1 учтены в $a_{2L,1}(0)$, однако некоторые из них могут быть учтены многократно (14). Кроме того необходимо аналогично учесть циклы, проходящие через ребро +2, +3 и так далее. Такое суммарное рассмотрение неизбежно будет учитывать циклы повторно, однако количество повторений не обязательно равно длине цикла и зависит от его состава.

Пример 14. Пути $1 \to -4 \to 5 \to -2 \to 1 \to -4 \to 6 \to -3$ и $1 \to -4 \to 6 \to -3 \to 1 \to -4 \to 5 \to -2$, начинающиеся с 1 эквивалентны, так как являются циклическим сдвигом друг друга, но будут учтены дважды.

Дальнейшие рассуждения можно проводить двумя способами:

— Рассматривать w_i в качестве символьных переменных. Можно выписать все возможные комбинации весов, приводящие к потенциальным циклам

длины $2L$ и затем вычислять спектры циклов, подставляя разные наборы разметок в полученные уравнения. Число циклов равно числу нулей в уравнениях.

— При заданных (фиксированных) весах ребер подсчитать спектр длин циклов в заданном диапазоне.

3.1. Веса как символьные переменные

В случае рассмотрения w_i как символьных переменных необходимо избавиться от эквивалентных относительно сдвига и инверсии комбинаций. Для этого каждый путь приводится к минимальному лексикографическому виду после чего убираются дубликаты.

Заметим, что цикл в обратном направлении к рассматриваемому циклу не может быть эквивалентен относительно сдвига исходному. Обозначим $\bar{p}$ путь в обратном направлении к пути $p = e_1,e_2,...,e_n$. Предположим p и $\bar{p}$ эквивалентны относительно сдвига – тогда найдется индекс i, такой что $\bar{e}_i, \bar{e}_{i-1}, ..., \bar{e}_1 = e_1, e_2, ..., e_i$. Если i нечетно, тогда $\bar{e}_{(i+1)/2} = e_{(i+1)/2}$ – противоречие. Если i четно, то $e_{i/2+1} = \bar{e}_{i/2}$, что противоречит ограничению что путь не может идти обратно по последнему пройденному ребру.

Таким образом каждому циклу соответствует ровно два пути в разных направлениях. Непосредственно инвертируя каждый путь несложно оставить ровно один из каждой пары.

По причине того, что число различных путей растет экспоненциально при построении всех возможных символьных комбинаций, лучше воспользоваться подходом meet-in-the-middle. Это позволяет для нахождения всех возможных циклов длины $2L$ рассматривать только пути длины L, в то время как описанный алгоритм рассматривает пути длины $2L$.

3.2. Фиксированные веса

В случае когда веса изначально зафиксированы можно считать, что веса коммутируют, поэтому достаточно считать не более M членов в каждом полиноме при умножении на матрицу A, по одному значению для каждой возможной суммы весов. После чего устранить дубликаты из результата с помощью обращения Мебиуса.

Рассмотрим для цикла p сколько раз он был учтен. Порядком цикла назовем максимальное r, такое что p можно представить как

$$p = \underbrace{ss\ldots s}_{r \text{ раз}}$$

Периодом цикла p назовем длину $|s| = \frac{|p|}{r}$. Таким образом цикл p периода l имеет l различных циклических сдвигов – следовательно будет учтен $2l$ раз, с учетом последовательностей в обратном порядке.

Последовательно для каждого i зафиксируем

$$a_0 = (\underbrace{0,0,...,0}_{i \text{ раз}},1,0,...,0)$$

и вычислим

$$a_{2L} = a_0 A^L$$

проводя все вычисления в кольце многочленов по модулю D^M.

Запомним количество циклов (всех возможных весов) проходящих через +i, содержащееся в многочлене $a_{2L,i}$ как $b_{2L,i}$.

Суммируя по всем возможным a_0 получаем

$$b_{2L} = \sum_i b_{2L,i}$$

- многочлен, с коэффициентами при D^w соответсвенно равными количеству циклов длины $2L$ и веса w, где каждый цикл учтен дважды столько, сколько он имеет различных циклических сдвигов (в обоих направлениях).

Далее временно забудем об ограничении инверсии, так как для устранения путей эквивалентных относительно разворота достаточно разделить результирующий спектр на два.

Обозначим за $g(l)$ – многочлен, коэффициент при D^w которого равен числу циклов длины и периода l веса w.

Определение 15. Введем операцию $T_d(p(D))$ где $p(D)$ – многочлен, а d – натуральное число как:

$$T_d(c_0 + c_1 D + c_2 D^2 + ... + c_n D^n) = c_0 + c_1 D^d + c_2 D^{2d} + ... + c_n D^{nd} \pmod{D^M}$$

Тогда при фиксированной длине L имеем равенство:

$$\sum_{d|L} d \cdot T_{\frac{L}{d}}(g(d)) = b_L$$

Равенство справедливо, так как каждый цикл длины L периода d веса W состоит из повторенного $\frac{L}{d}$ раз цикла длины и периода d веса w, такого что $w \cdot \frac{L}{d} = W \pmod{M}$ и однозначно им задается. Каждый такой цикл имеет d различных циклических сдвигов, поэтому учтен в b_L d раз. Таким образом суммирование ведется по всем возможных длинам периодов, после чего благодаря $T_{\frac{L}{d}}$ вес каждого цикла из $g(d)$ домножается на число повторений цикла для достижения длины L и каждый из циклов учитывается с коэффициентов d так как входит в b_L в виде d различных линейных последовательностей ребер.

В равенствах вида

$$f(n) = \sum_{d|n} g(d)$$

$g(d)$ может быть выражено с помощью формулы обращения Мебиуса:

$$g(n) = \sum_{d|n} \mu(d) f(n/d)$$

где

$$n = p_1^{e_1} \cdot p_2^{e_2} \cdot \ldots p_r^{e_r}$$

- разложение n на простые множители

$$\mu(n) = \begin{cases} 1 & \text{при } n = 1, \\ 0 & \text{если } \exists i \quad e_i > 1, \\ (-1)^r & \text{если } e_1 = e_2 = \ldots = e_r = 1 \end{cases}$$

В рассматриваемом случае можно ввести и использовать следующее обобщение формулы Мебиуса:

Теорема 16. Если

$$f(n) = \sum_{d|n} T_{\frac{n}{d}}(g(d))$$

где T_k удовлетворяет свойствам:

$$T_k(c \cdot p(D)) = c \cdot T_k(p(D)) \tag{17}$$

$$T_k(p(D) + q(D)) = T_k(p(D)) + T_k(q(D)) \tag{18}$$

$$T_{k_1}(T_{k_2}(p(D))) = T_{k_1 \cdot k_2}(p(D)) \tag{19}$$

$$T_1(p(D)) = p(D) \tag{20}$$

Тогда $g(d)$ может быть выражено:

$$g(n) = \sum_{d|n} \mu(d) T_d(f(n/d))$$

Доказательство.

Лемма 17.

$$\sum_{d|n} \mu(d) = 0 \quad \text{при } n > 1$$

Доказательство.

$$n = p_1^{e_1} \cdot p_2^{e_2} \cdot \ldots p_r^{e_r}$$

$$n^* = p_1 \cdot p_2 \ldots p_r$$

$$\sum_{d|n} \mu(d) = \sum_{d|n^*} \mu(d)$$

так как любое $\mu(d)$ где $p_i^2|d$ равно 0.

$$\sum_{d|n^*} \mu(d) = 1 - \binom{r}{1} + \binom{r}{2} - \ldots + (-1)^k \cdot \binom{r}{k} + \ldots = (1-1)^r = 0$$

так как существует $\binom{r}{k}$ делителей n^* состоящих из k простых множителей, каждый из которых внесет вклад $\mu(d) = (-1)^k$.

$$\sum_{d|n} \mu(d) \cdot T_d(f(n/d)) =$$

$$\sum_{d|n} \mu(d) \cdot T_d(\sum_{d'|\frac{n}{d}} T_{\frac{n}{dd'}}(g(d'))) =$$

$$\sum_{d|n} \mu(d) \cdot \sum_{d'|\frac{n}{d}} T_d(T_{\frac{n}{dd'}}(g(d'))) =$$

$$\sum_{d|n} \mu(d) \cdot \sum_{d'|\frac{n}{d}} T_{\frac{n}{d'}}(g(d')) =$$

$$\sum_{d|n} \sum_{d'|\frac{n}{d}} \mu(d) \cdot T_{\frac{n}{d'}}(g(d')) =$$

$$\sum_{d'|n} \sum_{d|\frac{n}{d'}} \mu(d) \cdot T_{\frac{n}{d'}}(g(d')) =$$

$$\sum_{d'|n} T_{\frac{n}{d'}}(g(d')) \cdot \sum_{d|\frac{n}{d'}} \mu(d) =$$

$$T_{\frac{n}{n}}(g(n)) = T_1(g(n)) = g(n)$$

Нетрудно заметить что операция $T_d(p(D))$ из определения 15 удовлетворяет ограничениям из теоремы 16. Действительно, первые два свойства следуют из линейности кольца многочленов по модулю. Третье и четвертое свойства очевидно следуют из определения операции 15.

Таким образом, можно воспользоваться теоремой 16 для разрешения выражения 3.2 относительно $g(L)$. В качестве $f(L)$ выступает b_L, а в качестве $g(d)$ необходимо взять $d \cdot g(d)$, внеся множитель d внутрь операции T_d, согласно свойству один:

$$L \cdot g(L) = \sum_{d|L} \mu(d) T_d(b_{L/d})$$

$$g(L) = \frac{\sum_{d|L} \mu(d) T_d(b_{L/d})}{L}$$

Напомним, что $g(L)$ содержит многочлен, коэффициент при D^w которого равен числу циклов длины и периода L веса w. Обозначим результирующую величину количества циклов длины L веса 0 с учетом эквивалентности относительно сдвига за C_L. Для подсчета величины C_L с помощью $g(L)$ необходимо произвести суммирование по всем возможным длинам периодов d в циклах длины L и весам w таким, что период повторенный $\frac{L}{d}$ раз приведет к циклу нулевого веса:

$$C_L = \sum_{d|L} \sum_{\frac{w \cdot L}{d} = 0 \pmod M} g^{(w)}(0)$$

где $g^{(w)}$ – w-ая производная g, используемая для получения коэффициента при D^w.

Наконец для устранения дубликатов относительно инверсии достаточно разделить C_L на два.

3.3. Алгоритм

$$a_0^i = (\underbrace{0,0,...,0}_{i \text{ раз}},1,0,...,0) \tag{21}$$

$$a_{2L}^i = a_0^i \cdot A^L \tag{22}$$

$$b_{2L} = \sum_i a_{2L}^i \tag{23}$$

$$g(L) = \frac{\sum_{d|L} \mu(d) T_d(b_{L/d})}{L} \tag{24}$$

$$C_L = \sum_{d|L} \sum_{\frac{w \cdot L}{d} = 0 \pmod M} g^{(w)}(0) \tag{25}$$

3.4. Оценка сложности

Пусть исходная базовая матрица имела размер $b \times c$, а коэффициент расширения равен M. Для простоты будем рассматривать (J,K)-регулярный МППЧ код с весом столбцов и строк J и K соответственно.

Тогда число ребер в графе Таннера обозначим за $E = b \cdot K = c \cdot J$. Ограничение на максимальную длину в спектре обозначим S.

Таким образом размер матрицы A составляет $E \times E$, каждый из элементов которой представляет собой многочлен степени не больше M. И для получения всех интересующих A^L, посредством перемножения матриц, достаточно затратить $O(S \cdot E^3 \cdot M)$ операций, так как перемножение и сложение многочленов по модулю D^M требует $O(M)$ времени в отличии от $O(1)$ для обычных чисел.

При каждом фиксированном стартовом ребре i и длине циклов $2L$, многочлен $a_{2L}^i = a_0^i \cdot A^L$ 22 может быть получен за время $E^2 \cdot M$, посредством перемножения вектора a_0 длины E на матрицу A^L размера $E \times E$, опять же по причине того что перемножение и сложение многочленов по модулю требует $O(M)$ операций. Таким образом суммарно получение всех необходимых a_{2L} займет время $O(E \cdot S \cdot E^2 \cdot M) = O(S \cdot E^3 \cdot M)$.

Сложение всех необходимых a_{2L} для получения b_{2L} (23) может быть осуществлено за суммарный размер полиномов a, а именно $O(E \cdot S \cdot M)$.

Все необходимые значения функции Мебиуса $\mu(d)$ могут быть подсчитаны за время $O(S \cdot \ln S)$ с помощью решета Эратосфена. При оценке времени суммарно затраченного на подсчет $g(L)$ заметим, что суммирование ведется по всем делителям чисел L от 1 до S. Как известно суммарное количество делителей чисел от 1 до n имеет порядок $O(n \cdot \ln n)$. Действительно, число d является делителем для $\lfloor \frac{n}{d} \rfloor$ чисел: $d, 2d, 3d, \ldots, \lfloor \frac{n}{d} \rfloor \cdot d$. Получаем сумму гармонического ряда $\sum_d \lfloor \frac{n}{d} \rfloor = O(n \cdot \ln n)$. Таким образом для подсчета всех $g(L)$ необходимо затратить $O(M \cdot S \cdot \ln S)$, так как операция T_d и сложение многочленов степени M может быть осуществлено за $O(M)$.

Пользуясь оценкой суммы гармонического ряда для 25 с учетом суммирования по весам получаем время необходимое для подсчета C_L – $O(M \cdot S \cdot \ln S)$, так как теперь складываются коэффициенты многочлена, а не целые многочлены.

Итого по всем шагам алгоритма получаем:

$$O(S \cdot E^3 \cdot M) + O(S \cdot E^3 \cdot M) + O(E \cdot S \cdot M) + O(M \cdot S \cdot \ln S) + O(M \cdot S \cdot \ln S) =$$

$$O(S \cdot E^3 \cdot M) + O(M \cdot S \cdot \ln S) = O(M \cdot S \cdot max(\ln S, E^3))$$

Очевидно для всех разумных входных данных член E^3 мажорирует $\ln S$ таким образом итоговая сложность:

$$O(M \cdot S \cdot E^3)$$

ГЛАВА 4. ЧИСЛЕННЫЙ АНАЛИЗ ВЛИЯНИЯ СПЕКТРОВ ЦИКЛОВ НА ВЕРОЯТНОСТЬ ОШИБКИ БП-ДЕКОДИРОВАНИЯ

Общий подход к генерации случайного регулярного МППЧ-кода состоит из следующих шагов:

а) Выбор весов столбцов и строк базовой матрицы (J,K). В исследовании будут рассмотрены (3,6) и (4,8)-регулярные МППЧ-коды, как наиболее применяемые на практике.

б) Выбор размера базовой матрицы (n_b,r_b). Размер должен удовлетворять соотношению $J \cdot n_b = K \cdot r_b$ (два способа подсчитать число единиц в матрице — по строкам и по столбцам). В исследовании рассмотрен размер (24,12), который удовлетворяет (3,6) и (4,8) с коэффициентам 4 и 3 соответственно.

в) Выбор базовой матрицы из фиксированного ансамбля кодов. В исследовании рассмотрены ансамбли Галлагера, Ричардсона-Урбанке и Квазициклических кодов.

г) Выбор размера результирующего кода после лифтинга. В исследовании рассмотрены размеры 576 и 2304, используемые в стандарте 802.16e WiMAX. Коэффициенты лифтинга $M = 24$ и $M = 96$ соответственно.

д) Разметка базовой матрицы весами по модулю M. Все разметки приняты равновероятными.

4.1. Описание ансамблей кодов

Генерация случайных матриц заданного размера с фиксированным числом единиц в строках и столбцах для задания регулярного МППЧ-кода может производиться различными способами. При проведении тестирования были рассмотрены следующие ансамбли кодов.

4.1.1. Ансамбль Галлагера

Матрицы в ансамбле Галлагера состоят из полос с фиксированным числом строк в каждой. Каждый столбец полосы содержит ровно одну единицу. Таким образом число полос равно весу столбца.

Например, рассмотрим (3,6)-код, $M = 4$. Такой код состоит из 6 полос, каждая из которых состоит из $M = 4$ строк. Первая строка имеет вид

$$\begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Остальные $M-1$ строк этой полосы — сдвиги первой строки на 6 позиций. Таким образом строится первая полоса. Оставшиеся 2 полосы — случайные перестановки первой полосы.

В результате получен (24,12)-код, он же (3,6)-регулярный МППЧ-код.

4.1.2. Ансамбль Ричардсона-Урбанке

В ансамбле Ричардсона-Урбанке все (3,6)-регулярные коды равновероятные.

Рассмотрим способ на примере того же (24,12)-кода. Возьмем последовательность (номера строк единиц):

$$\begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 2 & 2 & 2 & 2 & 2 & 2 & 3 & 3 & 3 & 3 & 3 & 3 & \dots & 12 & 12 & 12 & 12 & 12 & 12 \end{pmatrix}$$

Возьмем случайную перестановку чисел этой последовательности. Берем первые $J=3$ числа, скажем 7,3,11. Они указывают номера строк единиц первого столбца. Берем еще 3 числа и находим второй столбец и так далее.

Этот способ не гарантирует регулярность кода, так как на одной позиции может оказаться две единицы и таким образом столбец будет содержать меньше чем J строк с единицами. Это оказалось существенно при проведении экспериментов. Матрицы в которых столбец содержит меньше J строк с единицами более разряженные и соответственно их граф Таннера содержит меньше циклов. Таким образом такие матрицы становятся лучшими согласно спектру, но вполне ожидаемо показывают довольно плохие результаты при моделировании.

Таким образом был рассмотрен немного модифицированный ансамбль Ричардсона-Урбанке, который контролировал регулярность кода, отбрасывая иррегулярные МППЧ-коды. Так как регулярные коды составляют значительную часть кодов из ансамбля Ричардсона-Урбанке, данная модификация незначительно увеличивает время генерации.

4.1.3. Ансамбль квазициклических кодов

4.2. Описание эксперимента

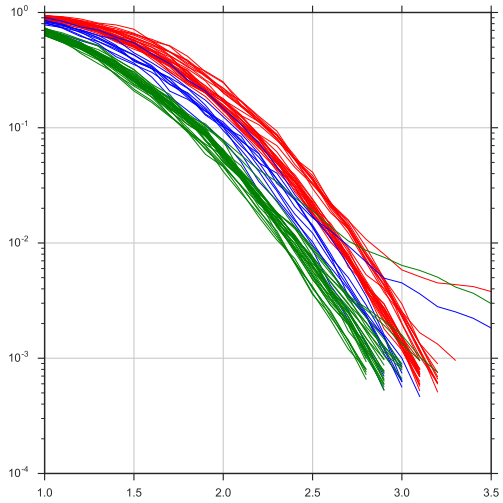


Рисунок 7 – Ричардсон-Урбанке 4x8 576

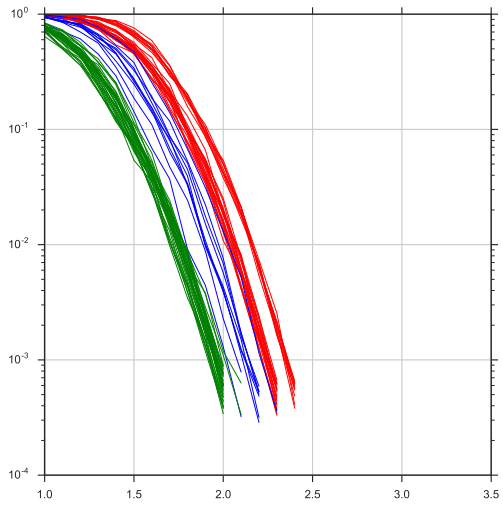


Рисунок 8 – Ричардсон-Урбанке 4x8 2304

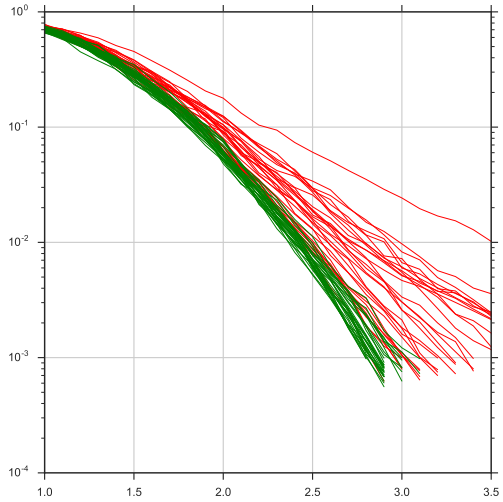


Рисунок 9 – Галлагер 3x6 576

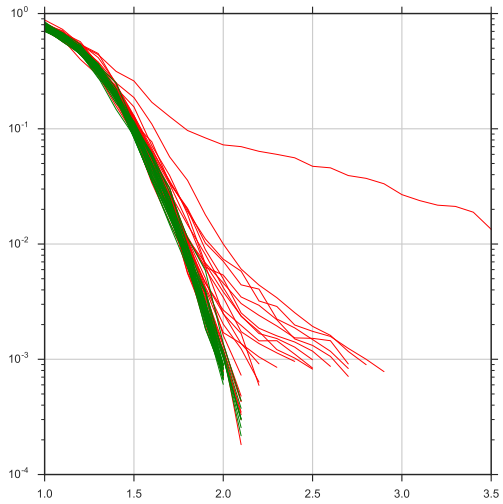


Рисунок 10 – Галлагер 3x6 2304

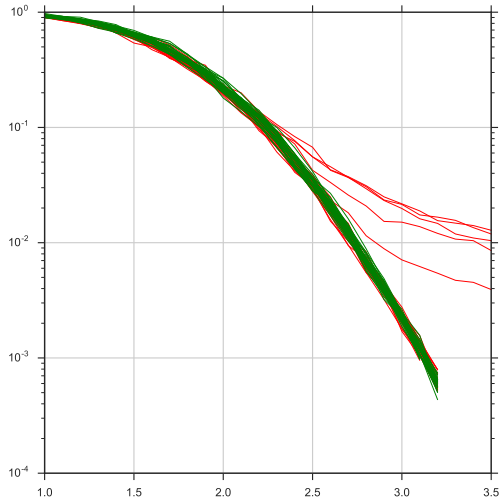


Рисунок 11 – Галлагер 4x8 576

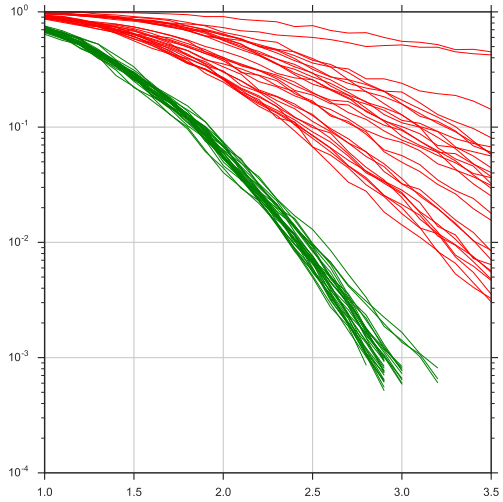


Рисунок 12 – Квазициклические коды 3x6 576

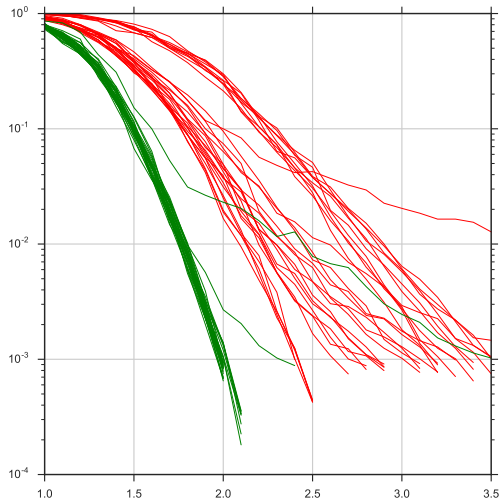


Рисунок 13 – Квазициклические коды 3x6 2304

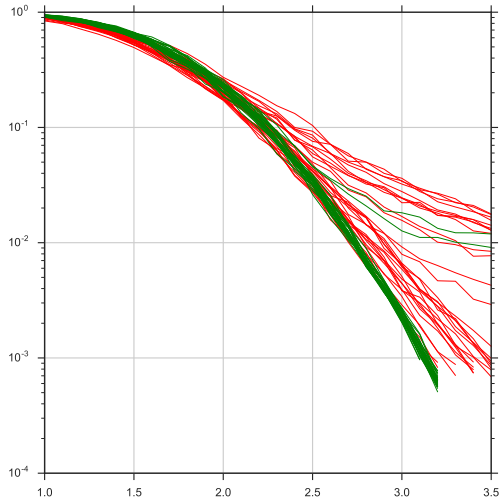


Рисунок 14 – Квазициклические коды 4x8 576

ЗАКЛЮЧЕНИЕ

а) Получены экспериментальные доказательства прямой зависимости между количеством коротких циклов и вероятностью ошибки на блок для различных ансамблей кодов

б) Разработан вычислительно эффективный алгоритм подсчета спектра графа Таннера, который может быть использован для ускорения отбора кодов.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 *Gallager R.* Low-Density Parity-Check Codes. — 1963.

2 *Холл М.* Комбинаторика. — Издательство "Мир"Москва, 1970.

3 *Кудряшов Б.* Основы теории кодирования. — БХВ-Петербург, 2016.

4 *Alon N.*, *Yuster R.*, *Zwick U.* Finding and counting given length cycles // Algorithmica. — 1997. — P. 209–223.

5 *Karimi M.*, *Banihashemi A.* Counting short cycles of quasi cyclic protograph LDPC-codes // IEEE Communications Letters. — 2012. — Mar. — Vol. 16, no. 3. — P. 400–403.

6 *Karimi M.*, *Banihashemi A.* A message-passing algorithm for counting short cycles in graph. — 2013.

7 *Monien B.* How to find long paths efficiently // Annals of Discrete Mathematics. — 1985. — No. 25. — P. 239–254.

8 *Alon N.*, *Yuster R.*, *Zwick U.* Color-coding. — 1995.

9 *Halford T.*, *Chugg K.* An algorithm for counting short cycles in bipartite graphs // IEEE Transactions on Information Theory. — 2006. — Jan. — Vol. 52, no. 1.

10 *Harray F.*, *Manvel B.* On the number of cycles in a graph // Matematicky casopis. — 1971. — Vol. 21, no. 1. — P. 55–63.

11 *Yedidia J.*, *Freeman W.*, *Weiss Y.* Understanding Belief Propagation and its Generalizations. — 2001. — Нояб.

12 *MacKay D.* Encyclopedia of Sparse Graph Codes. — URL: `http://www.inference.phy.cam.ac.uk/mackay/codes/data.html` (visited on 05/08/2017).

13 Stressing the BER simulation of LDPC codes in the error floor region using GPU clusters / G. Falcao [et al.]. — 2013.

14 *Broulim J.*, *Ayriyan A.*, *Georgiev V.* OpenCL/CUDA algorithms for parallel decoding of any irregular LDPC code using GPU // Journal of LaTeXclass files. — 2015. — Aug. — Vol. 14, no. 8.