МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

На правах рукописи

КИРЬЯНОВ Иван Андреевич

ДЕКОДИРОВАНИЕ КОДОВ С МАЛОЙ ПЛОТНОСТЬЮ ПРОВЕРОК НА ЧЕТНОСТЬ

Специальность 05.12.13 – Системы, сети и устройства телекоммуникаций

ДИССЕРТАЦИЯ
на соискание ученой степени
кандидата технических наук

Научный руководитель:
Кандидат технических наук
Важенин Н.А.

Москва
2015

Содержание

Введение

Общая характеристика работы

Работа относится к теории и технике помехоустойчивого кодирования. Рассматриваются пути повышения эффективности практической реализации декодеров кодов с малой плотностью проверок на четность (*LDPC от англ. Low-density parity-check*). Разрабатываются и исследуются варианты реализации LDPC декодеров, обеспечивающих наилучшие показатели по критерию помехоустойчивость-сложность технической реализации. Результаты исследований апробируются на примере спутниковой телекоммуникационной подсистемы передачи эфемеридной и служебной информации наземным потребителям с использованием сигнала L1C.

Актуальность диссертационной работы

В современных телекоммуникационных системах большое внимание уделяется помехозащищенности передаваемой информации. Помехозащищенность обеспечивается за счет применения помехоустойчивого кодирования информации. Примерно с начала 90-ых годов задачу помехоустойчивого кодирования решали с помощью турбо кодов. Однако с ростом объемов трафика и скоростей передачи информации возрос интерес к более эффективной технике помехоустойчивого кодирования – кодированию с помощью кодов с малой плотностью проверок на четность.

Коды с малой плотностью проверок на четность обладают эффективными алгоритмами декодирования, позволяющими быстро и надежно корректировать поврежденную при передаче в результате шумов информацию. Данные коды позволяют осуществлять работу цифровой линии связи при отношениях сигнал/шум, близких к границе Шеннона, опережая по эффективности коррекции ошибок турбо коды на относительно больших длинах кодовых слов.

Коды с малой плотностью проверок на четность рекомендованы для коррекции ошибок в современных стандартах связи DVB-S2, DVB-C2, Wi-Fi 802.11n, WiMAX 802.16e.

Степень разработанности темы диссертации

На данный момент можно выделить два направления исследований в области кодов с малой плотностью проверок на четность. Первое из них относится к синтезу конструкций LDPC кодов. В этом направлении следует отметить труды Афанасьева В.Б., Воробьева К.А., Зигангирова Д.К., Зигангирова К.Ш., Зяблова В.В., Иванова Ф.И., Крука Е.А., Овчинникова А.А., Пацей Н.В., Потапова В.Г., Трухачева Д.В., Costello D., Johnson S.J., Kou Y., Luby M.G., Richardson J., Weller S.R.

Второе направление исследует алгоритмическую составляющую LDPC кодеков. В этом направлении следует отметить труды Башкирова А.В., Белоголового А.В., Витязева В.В., Владимирова С.М., Климова А.И., Козлова А.В., Кравченко А.Н., Лихобабина Е.А., Муратова А.В., Овечкина Г.В., Проскурина А.А., Солтанова А.Г., Chen J., Fossorier M., Kim N., Urbanke R.L.

Общий вклад в развитие теории внесли труды Акулинина А.С., Золотарева В.В., Зубарева Ю.Б., Колесника В.Д., Eckford A.W., MacKay D., Tanner M.

Последние достижения в области кодирования кодами с малой плотностью проверок на четность позволили практически вплотную приблизиться к границе Шеннона. Как следствие, актуальной задачей на сегодня является не увеличение исправляющей способности, а разработка методики выбора алгоритма декодирования, обеспечивающего наилучший компромисс по определенным критериям качества в рамках рассматриваемой системы связи, и модификация существующих алгоритмов с целью повышения вычислительной эффективности декодирования и экономии используемых ресурсов памяти.

Цель диссертационной работы и решаемые задачи

Целью работы является разработка и исследование алгоритмов декодирования LDPC кодов. Для достижения поставленной цели решаются следующие задачи:

1. Анализ существующих алгоритмов декодирования LDPC кодов.
2. Оценка вычислительной сложности декодирования LDPC кодов.
3. Оценка статистических характеристик декодирования (BER, число итераций, сходимость синдрома) LDPC кодов.
4. Разработка модификаций и методов, позволяющих повысить вычислительную эффективность декодирования и сэкономить используемые при декодировании ресурсы памяти.

Методы исследования

В работе использовался аппарат теории вероятностей, теории электрической связи, дискретной математики и математического анализа.

Для проведения моделирования использовалась среда имитационного моделирования MATLAB с пакетом Simulink и среда разработки программных продуктов Microsoft Visual Studio 2010.

Научная новизна

1. Получены и проанализированы соотношения для расчета сложности итерации декодирования LDPC кодов для различных алгоритмов коррекции ошибок.
2. Получены и исследованы статистические характеристики декодирования (BER, число итераций, сходимость синдрома) для различных алгоритмов коррекции ошибок в рамках рассматриваемого LDPC кода.
3. Предложена методика выбора алгоритма декодирования, обеспечивающего заданную вероятность ошибки при наименьшей сложности декодирования.
4. Предложена методика компактного представления разряженной проверочной матрицы LDPC кода, позволяющая экономить ресурсы памяти для её хранения.
5. Предложены модификации алгоритмов, позволяющие повысить вычислительную эффективность декодирования без потери исправляющей способности при незначительном увеличении требований к памяти для хранения внутренних переменных декодера.
6. Предложен способ идентификации инверсии битового потока за счет внутренних ресурсов LDPC декодера и исследована его работа на реальном сигнале.

Практическая ценность диссертационной работы

Предложенная методика выбора алгоритма декодирования LDPC может использоваться при проектировании современных цифровых телекоммуникационных систем.

Компактное представление проверочной матрицы позволяет сэкономить в 2 раза ресурсы памяти, выделенные на её хранение.

Применение разработанных модификаций к алгоритмам коррекции ошибок позволяет без потери в качестве декодирования повысить скорость декодирования в 3 раза при незначительном увеличении требований к памяти для хранения внутренних переменных декодера.

Предложенный способ идентификации инверсии битового потока на входе LDPC декодера может применяться в системах связи, не имеющих в принимаемом сигнале фиксированную преамбулу для решения этой задачи.

Разработанные программные реализации LDPC и БЧХ кодеков могут быть внедрены в системы связи, использующие соответствующее помехоустойчивое кодирование информации.

Внедрение

Программные реализации декодера кодов с малой плотностью проверок на четность и БЧХ декодера были внедрены в ОАО «Топкон Позишионинг Системс».

На основе полученных результатов разработано учебное пособие «Принципы построения и алгоритмы реализации LDPC кодеков» для использования в учебном процессе по специальности 210402 «Средства связи с подвижными объектами» и направлению подготовки 210700 «Инфокоммуникационные технологии и системы связи».

Достоверность

Достоверность полученных результатов обусловлена сопоставлением результатов моделирования на имитационной модели с теорией, а также экспериментами других авторов и результатами декодирования и расшифровки реального сигнала.

Положения, выносимые на защиту

1. Предложенная методика выбора алгоритма декодирования LDPC кодов позволяет определить алгоритм, обеспечивающий заданную вероятность ошибки на выходе декодера при наименьшей сложности декодирования.
2. Предложенная методика представления разряженной проверочной матрицы LDPC кода позволяет уменьшить в 2 раза требуемые ресурсы памяти, предназначенные для её хранения.
3. Разработанные модификации алгоритмов декодирования LDPC кодов позволяют повысить скорость работы декодера в 3 раза при незначительном увеличении требований к памяти для хранения внутренних переменных декодера.
4. Предложенный способ идентификации инверсии битового потока позволяет определять инверсию за счет внутренних ресурсов LDPC декодера.

Апробация работы

Основные результаты доложены на Московской молодежной научно-практической конференции «Инновации в авиации и космонавтике -2012» (МАИ, Москва, 2012); на 1-ой межвузовской студенческой научно-технической конференции «Современные состояния и перспективы развития сложных радиоэлектронных систем» (ОАО «ГСКБ «Алмаз-Антей», Москва, 2012); на Московской молодежной научно-практической конференции «Инновации в авиации и космонавтике -2013» (МАИ, Москва, 2013); на 12-ой Международной конференции «Авиация и космонавтика - 2013» (МАИ, Москва, 2013); на Московской молодежной научно-

практической конференции «Инновации в авиации и космонавтике -2014» (МАИ, Москва, 2014); на 13-ой Международной конференции «Авиация и космонавтика - 2014» (МАИ, Москва, 2014).

Публикации

Основные результаты исследований опубликованы в 17 работах, в числе которых 7 статей в журналах, входящих в перечень ВАК, 1 свидетельство о государственной регистрации программы для ЭВМ и 9 других публикаций, не входящих в перечень ВАК.

Личный вклад автора

Все результаты, полученные в данной работе, являются личными достижениями автора.

Структура и объем работы

Работа состоит из введения, пяти глав, заключения, содержит 77 рисунков, 21 таблицу и 151 формулу. Работа размещена на 129 страницах.

Соответствие работы паспорту специальности

Работа соответствует паспорту специальности 05.12.13 – «Системы, сети и устройства телекоммуникаций» (пункт 8 – «Исследование и разработка новых сигналов, модемов, кодеков, мультиплексоров и селекторов, обеспечивающих высокую надежность обмена информацией в условиях воздействия внешних и внутренних помех»).

ГЛАВА 1. Анализ алгоритмов декодирования LDPC кодов

В данной главе поставлена задача декодирования сигнала L1C и сформулирована цель работы. Введены основные понятия и локализованы существующие алгоритмы декодирования кодов с малой плотностью проверок на четность. Приведено их описание и группировка по некоторым признакам. Отмечены плюсы и минусы алгоритмов декодирования. Приведены существующие результаты исследований других авторов.

1.1 Основные понятия

Любой блоковый код, в том числе и с малой плотностью проверок на четность, можно описать с помощью порождающей и проверочной матрицы. Порождающая матрица задает базис пространства кодовых слов. Проверочная матрица представляет собой систему проверочных уравнений на четность.

Коды с малой плотностью проверок на четность, известные как низкоплотностные коды, впервые были предложены Робертом Галлагером в [60]. Особенностью этих блоковых кодов является разреженная структура матрицы проверки на четность. Такая матрица состоит в основном из нулей и содержит малое число единиц. Согласно [87], матрица, содержащая больше половины нулевых элементов, считается низкоплотностной. В [30] отмечено, что вес столбцов и строк в низкоплотностной матрице много меньше длины кода. На практике используют матрицы, содержащие 99% нулевых элементов. К примеру, в соответствии со стандартом спутникового телевидения высокой четкости DVB-S2 при декодировании LDPC кодов используется низкоплотностная матрица размерностью 32400 на 64800, содержащая около 227000 ненулевых элементов.

Низкоплотностная матрица проверки на четность может быть регулярной или нерегулярной. Регулярные матрицы имеют одинаковое число «1» в каждой строке и одинаковое число «1» в каждом столбце. Нерегулярные матрицы могут иметь произвольную структуру. Отмечается [8], что нерегулярные конструкции кодов обеспечивают более высокую помехоустойчивость, чем регулярные за счет так называемого «эффекта волны», когда более защищенные символы, за которыми стоит больше «1» в вертикали матрицы проверки на четность, быстрее других декодируются и «помогают» менее защищенным символам.

Проверочную матрицу низкоплотностного кода принято [32,42,59] описывать с помощью графа Таннера. Это двудольный граф с двумя типами узлов (символьное узлы и проверочные узлы), соединёнными ребрами. Пример построения графа Таннера для проверочной матрицы $\boldsymbol{H}$ приведен на рисунке 1.1. Два противоположных узла графа Таннера

соединяются ребром, если на соответствующей позиции в матрице проверки на четность стоит «1».

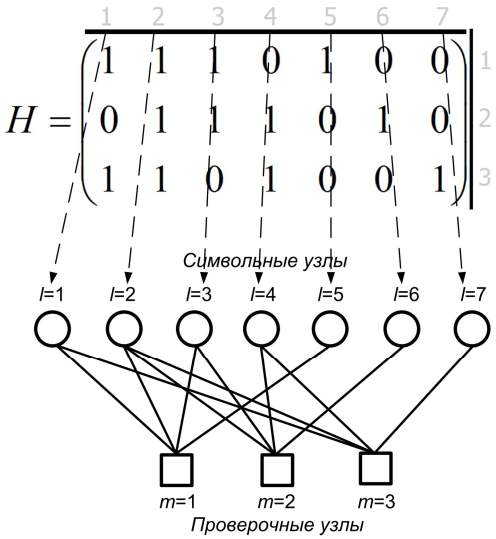

Рисунок 1.1 – Построение графа Таннера по проверочной матрице

Алгоритмы декодирования кодов с малой плотностью проверок на четность относятся к семейству алгоритмов «Massage passing», в работе которых происходит итеративный пересчет значений символьных (верхние на рисунке 1.1) и проверочных узлов графа, а также обмен рассчитываемыми значениями между узлами по ребрам. Это обуславливает удобство представления кода в виде графа.

1.2 Постановка задачи декодирования сигнала L1C

Коды с малой плотностью проверок на четность могут быть применены в широком спектре телекоммуникационных систем, начиная от мобильной связи и заканчивая сложными спутниковыми системами.

В данной работе LDPC коды исследуются на примере спутниковой телекоммуникационной подсистемы передачи эфемеридной и служебной информации наземным потребителям с использованием сигнала L1C.

Информация, которую несет сигнал L1C, закодирована БЧХ кодом и двумя кодами с малой плотностью проверок на четность различной длины. Последние обеспечивают наилучшую исправляющую способность среди всех существующих помехоустойчивых кодов. Кодирование применено с целью обеспечения целостности передаваемой информации. Структура кадра сигнала L1C представлена на рисунке 1.2.

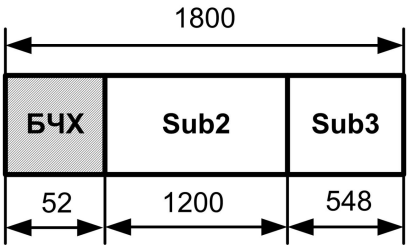

Рисунок 1.2 – Структура кадра сигнала L1C

Кадр состоит из 1800 отсчетов и делится на 3 части. Первая часть представляет собой закодированный БЧХ кодом номер кадра L1C. Информация в части Sub2 (Subframe2) закодирована кодом с малой плотностью проверок на четность длиной 1200 символов. Информация в части Sub3 (Subframe3) закодирована кодом с малой плотностью проверок на четность длиной 548 символов.

Матрицы проверки на четность, соответствующие кодам из частей Sub2 и Sub3, представлены на рисунке 1.3. Точки соответствуют расположению «1».

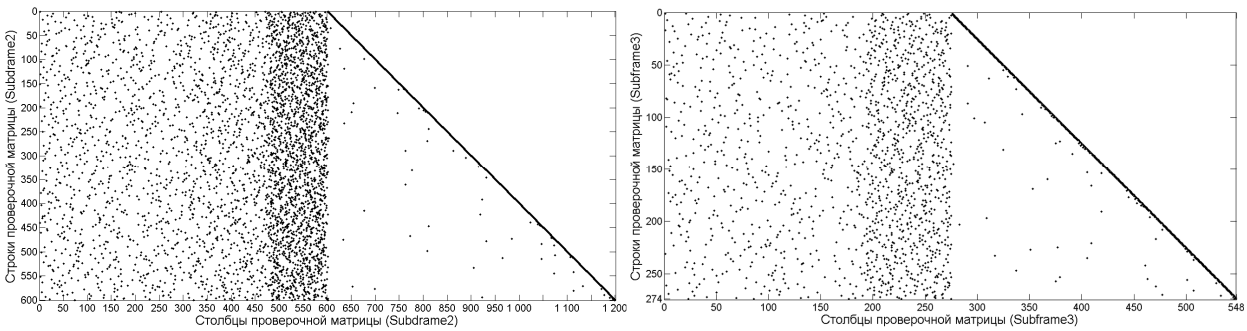

Рисунок 1.3 – Структуры матриц проверки на четность для Sub2 и Sub3

По отношению числа строк к числу столбцов делается вывод о том, что скорость кодирования используемых LDPC кодов равна 0.5. Скорость кодирования показывает степень избыточности кодирования. Оба кода систематические, то есть первые 600 символов в Sub2 и первые 274 символа в Sub3 содержат полезную информацию. Матрицы содержат 4818 и 2071 ненулевых элементов, соответственно.

Целью работы является разработка и исследование алгоритмов декодирования LDPC кодов для их дальнейшего применения при декодировании составных частей сигнала L1C. Для достижения цели в работе анализируются существующие алгоритмы декодирования, оценивается их сложность и статистических характеристики декодирования, предлагается методика выбора алгоритма декодирования, а также разрабатываются методы, позволяющие

повысить вычислительную эффективность декодирования и сэкономить используемые при декодировании ресурсы памяти.

Предлагаемая методика выбора алгоритма декодирования заключается в локализации алгоритмов декодирования, обеспечивающих требование по вероятности ошибки на выходе декодера при фиксированном отношении сигнал/шум и определении среди локализованных алгоритмов тех, которые обеспечивают это требование с минимальной сложностью. Как следствие, необходимо иметь реестр всех существующих алгоритмов декодирования LDPC (глава 1), из которых будет осуществляться выбор, а также оценить сложность (глава 2) и эффективность (глава 3) декодирования.

1.3 Известные алгоритмы декодирования

В своей работе [60] Галлагер предложил два алгоритма декодирования кодов с малой плотностью проверок на четность. Это алгоритм с инверсией бита «Bit flip», работающий с «жесткими» решениями демодулятора, и алгоритм с распространением доверия «Belief propagation», использующий в своей работе «мягкие» решения. Справедливо отметить, что алгоритм «жесткого» декодирования «Bit flip» обладает скромной корректирующей способностью. Алгоритм с распространением доверия «Belief propagation» достигает максимума правдоподобия в процессе декодирования, однако необходимость использования гиперболических функций при расчете поправок делает этот алгоритм сложным для практических приложений. По этой причине существуют альтернативные пути реализации низкоплотностного декодирования, обеспечивающие достойный компромисс между сложностью и эффективностью декодирования. В числе них:

- реализация гиперболических функций таблицей;
- кусочная аппроксимация гиперболических функций;
- использование субоптимальных алгоритмов декодирования.

В данной работе локализованы основные алгоритмы декодирования кодов с малой плотностью проверок на четность. Алгоритмы структурированы по семействам на рисунке 1.4.

Семейство алгоритмов минимума суммы «Min-sum» использует для формирования поправок в основном операции сравнения и умножения, в то время как семейство мажоритарных алгоритмов «UMP BP» использует операции сложения по модулю 2 (исключающее ИЛИ). Семейство алгоритмов с варьируемым порогом является модернизацией семейства мажоритарных алгоритмов.

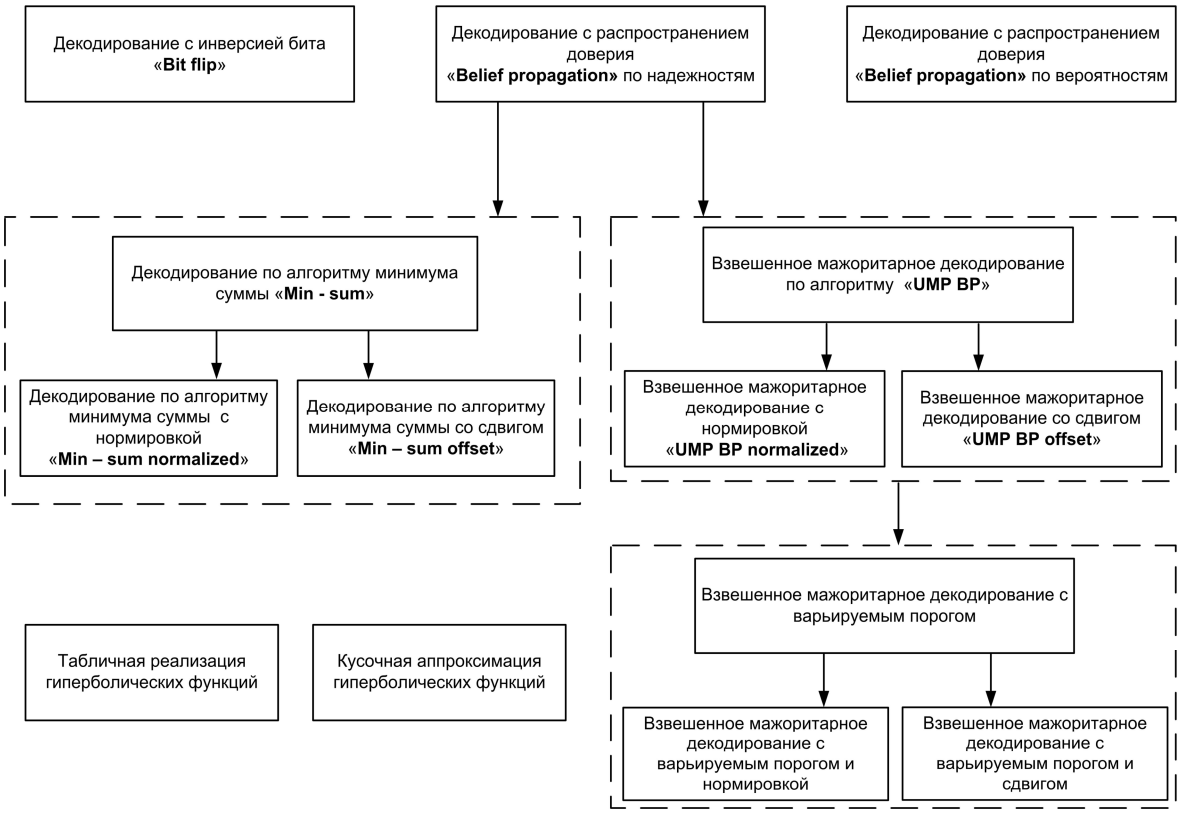

Рисунок 1.4 – Классификация алгоритмов декодирования низкоплотностных кодов

1.3.1 Алгоритм с инверсией бита «Bit flip»

Алгоритм декодирования «Bit flip», также известный как «Bit-flipping» и «жесткое решение по Галлагеру», был предложен Галлагером в [60] и позднее рассматривался в [30,32,38,53,61,75,94], как один из двух базовых алгоритмов декодирования низкоплотностных кодов. Низкоплотностный декодер, исправляющий ошибки по этому алгоритму, работает с «жесткими» решениями демодулятора. Декодер вычисляет все проверки (под проверкой понимается вычисление элемента синдрома на основе функционала от строки проверочной матрицы и принятой кодовой комбинации) и затем в кодовой комбинации инвертирует символы, которые участвовали в определенном числе невыполнившихся проверок, превышающим порог. Затем декодер вычисляет все проверки для новых значений символов и в случае, если все проверки выполняются, заканчивает декодирование. В противном случае начинается новая итерация декодирования. Итерации повторяются до тех пор, пока все ошибки не будут исправлены, либо, пока декодер не проделает максимальное число итераций, отведенное для декодирования.

Порог инвертирования, как правило [28,38], выставляется в половину от всего числа проверок, в которых участвует инвертируемый символ.

На рисунке 1.5 приведен пример регулярной низкоплотностной проверочной матрицы, где каждый символ участвует в трех проверках. Предположим, символ номер 1 кодового слова пришел с ошибкой, следовательно, проверки на четность с номерами 1, 7 и 15 не выполнятся, а ряд символов будут помечены, как участвующие в невыполнившихся проверках.

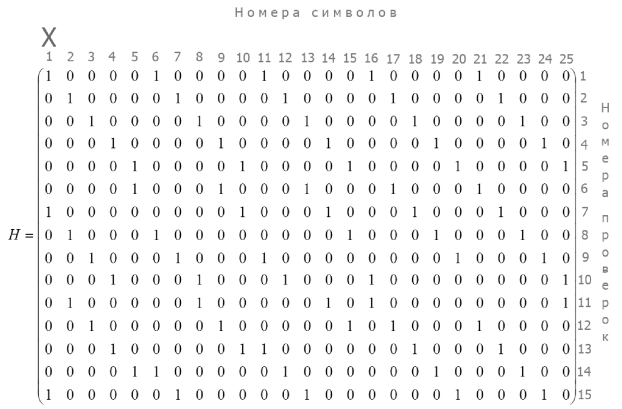

Рисунок 1.5 - Регулярная матрица низкоплотностного кода (25,3,5)

На рисунке 1.6 приводится гистограмма, показывающая количество участий всех символов кодового слова в невыполнившихся проверках. В данном случае установлен порог инвертирования 1,5, так как каждый символ участвует в 3 проверках на четность. Порог инвертирования превысил только символ номер 1. В соответствии с алгоритмом «Bit flip», он инвертируется и проверки перевычисляются. Итеративная процедура декодирования продолжается до тех пор, пока не будет получено кодовое слово или декодер не использует все итерации, отведенные для декодирования.

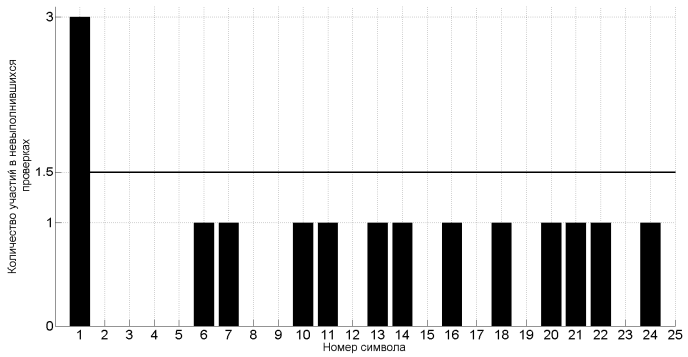

Рисунок 1.6 - Участие символов в невыполненных проверках

Важным требованием для эффективной работы алгоритма является отсутствие в проверочной низкоплотностной матрице циклов длиной 4. Наличие таких циклов может привести к инверсии правильно принятых символов.

Существует множество модификаций классического алгоритма декодирования с «жесткими» решениями. Например, «Weighted Bit-Flip» (WBF) [72] и его улучшенные версии

[69,76,93], в которых инвертирование символа зависит не только от количества участий в несошедшихся проверках, но и от надежностей символов, входящих в одну проверку, а также «Gradient Descent Bit-Flipping» [88] и его модификация [80], показывающие лучшие характеристики помехоустойчивости чем семейство «Weighted Bit-Flip» за счет учета корреляции между «жестким» решением и значением принятого «мягкого» решения.

Несмотря на существование ряда модификаций классического алгоритма с «жесткими» решениями, «жесткое» декодирование не раскрывает весь потенциал корректирующей способности LDPC кодов. Повысить корректирующую способность декодирования LDPC кодов позволяет переход от «жестких» решений к «мягким» и использование специальных техник декодирования, в основе которых лежит алгоритм декодирования с распространением доверия «Belief propagation».

1.3.2 Алгоритм с распространением доверия «Belief propagation» по вероятностям

Алгоритм декодирования «Belief propagation», также известный как «Sum-product» или «Сумма произведений», был предложен Галлагером в [60] и позднее рассматривался в [28,30,37,55,72,74,75,87]. Низкоплотностный декодер, исправляющий ошибки по этому алгоритму, работает с «мягкими» решениями демодулятора. В основе алгоритма лежит первая теорема Галлагера, сформулированная и доказанная в [60] и [61], которая гласит, что апостериорная вероятность того, что была принята «1» или «0» зависит от априорных вероятностей «1» и «0» символов, входящих в одну проверку с тем символом, для которого рассчитывается апостериорная вероятность. В результате своей работы алгоритм вычисляет апостериорные вероятности того, что была принята «1» или «0» для каждого символа. Для описания алгоритма используются обозначения из [30]:

m – номер проверки, то есть номер строки в проверочной матрице низкоплотностного кода;

l – номер принятого символа, то есть номер столбца в проверочной матрице низкоплотностного кода;

$\zeta(m)$ - множество символов, которые участвуют в m-ой проверке;

$\mu(l)$ - множество проверок, в которых участвует l-ый символ.

Алгоритм итеративно вычисляет сообщения от символьных узлов графа Таннера к проверочным узлам и наоборот:

- $q^{x}_{m,\ell}$ - сообщение, адресованное проверочному узлу m от символьного узла l, которое сформировано по информации, полученной от всех смежных проверочных узлов в графе Таннера, кроме узла m.

- $r_{m,\ell}^{x}$ - сообщение, адресованное символьному узлу *l* от проверочного узла *m*, которое сформировано по информации, полученной от всех смежных символьных узлов в графе Таннера, кроме узла *l*.

Начальные установки алгоритма. Для каждого принятого символа *l* следует установить априорные вероятности p_ℓ^1 и p_ℓ^0 того, что была принята «1» и «0», соответственно. Для каждого ненулевого элемента проверочной матрицы установить значения $q_{m,\ell}^0$ и $q_{m,\ell}^1$ так, что

$$q_{m,\ell}^0 = p_\ell^0 \qquad q_{m,\ell}^1 = p_\ell^1. \tag{1.1}$$

Шаг 1. Расчет сообщений от символьных узлов. Для каждого ненулевого элемента проверочной матрицы вычислить

$$q_{m,\ell}^0 = p_\ell^0 \prod_{m' \in \mu(\ell)\setminus m} r_{m',\ell}^0, \quad q_{m,\ell}^1 = p_\ell^1 \prod_{m' \in \mu(\ell)\setminus m} r_{m',\ell}^1, \tag{1.2}$$

и произвести нормировку с множителем $\alpha = 1/(q_{m,\ell}^0 + q_{m,\ell}^1)$,

$$q_{m,\ell}^0 = \alpha q_{m,\ell}^0, \quad q_{m,\ell}^1 = \alpha q_{m,\ell}^1. \tag{1.3}$$

Нормировка необходима для удовлетворения тождества $q_{m,\ell}^0 + q_{m,\ell}^1 = 1$. Для первой итерации вместо формулы (1.2) используют формулу (1.1).

Графически пересчет символьных узлов можно представить на рисунке 1.7.

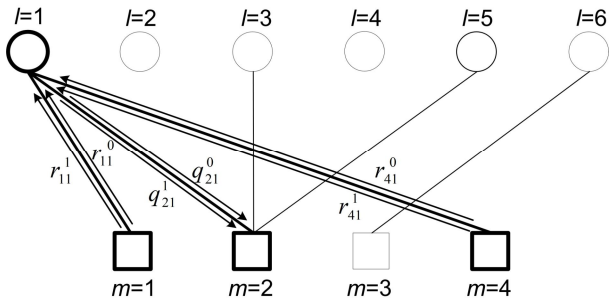

Рисунок 1.7 - Графическое представление пересчета символьных узлов графа Таннера

Шаг 2. Расчет сообщений от проверочных узлов. Для каждого ненулевого элемента проверочной матрицы вычислить

$$\delta r_{m,\ell} = \prod_{\ell' \in \xi(m) \setminus \ell} \left(q^0_{m,\ell'} - q^1_{m,\ell'} \right), \tag{1.4}$$

и

$$r^0_{m,\ell} = (1 + \delta r_{m,\ell})/2, \quad r^1_{m,\ell} = (1 - \delta r_{m,\ell})/2. \tag{1.5}$$

Графически пересчет проверочных узлов можно представить на рисунке 1.8.

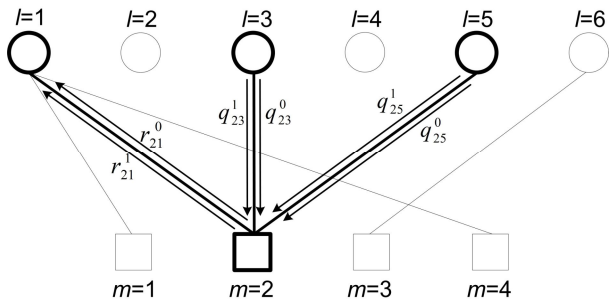

Рисунок 1.8 - Графическое представление пересчета проверочных узлов графа Таннера

Шаг 3. Вычисление апостериорных вероятностей.

Для каждого *l* вычислить апостериорные вероятности

$$q^0_\ell = p^0_\ell \prod_{m \in \mu(\ell)} r^0_{m,\ell}, \quad q^1_\ell = p^1_\ell \prod_{m \in \mu(\ell)} r^1_{m,\ell}, \tag{1.6}$$

и произвести нормировку с множителем $\alpha = 1/(q^0_\ell + q^1_\ell)$,

$$q^0_\ell = \alpha q^0_\ell, \quad q^1_\ell = \alpha q^1_\ell. \tag{1.7}$$

Нормировка необходима для удовлетворения тождества $q^0_\ell + q^1_\ell = 1$.

По полученным апостериорным вероятностям q^0_ℓ и q^1_ℓ формируется «жесткий» вектор, так, что *l*-ый символ равен «1», в случае, если $q^1_\ell > 0.5$, и «0» в остальных случаях. Если все

проверки сходятся, декодирование заканчивается. В противном случае следует вернуться к шагу 1.

1.3.3 Алгоритм с распространением доверия «Belief propagation» по надежностям

Как было показано в [60] и [61] и отмечалось в [30], используя алгоритм итеративного декодирования распространения доверия, более удобно оперировать логарифмическими отношениями правдоподобия вместо вероятностей. Это исключает необходимость нормировки сообщений (1.3) от символьных узлов графа, а также апостериорных вероятностей принятых символов (1.7) в конце каждой итерации. Эта модификация алгоритма рассматривалась ранее в [31,38,46,58,75,83,87,95].

Логарифмическое отношение правдоподобия для кодового символа C_l вычисляется по следующей формуле:

$$L(C_l) = \log \frac{P(C_l = 0 \mid Y)}{P(C_l = 1 \mid Y)}, \tag{1.8}$$

где Y – наблюдаемый на входе канала символ.

Для описания этого алгоритма используются те же обозначения, что и при описании вероятностного алгоритма распространения доверия «Belief propagation», заменив вероятности $q^x_{m,\ell}$ и $r^x_{m,\ell}$ на надежности $L(q_{m,l})$ и $L(r_{m,l})$, где [87]

$$L(q_{m,l}) = \log(\frac{q^0_{m,l}}{q^1_{m,l}}), \tag{1.9}$$

$$L(r_{m,l}) = \log(\frac{r^0_{m,l}}{r^1_{m,l}}). \tag{1.10}$$

Для качественного пояснения работы алгоритма с распространением доверия следует обратиться к матрице проверки на четность на рисунке 1.5. В первой строке проверочной матрицы ненулевые элементы стоят на позициях 1,6,11,16 и 21, соответственно. Это говорит о том, что в первой проверке на четность участвуют символы принятого слова с номерами 1,6,11,16 и 21. Фрагмент графа Таннера для описываемого случая представлен на рисунке 1.9.

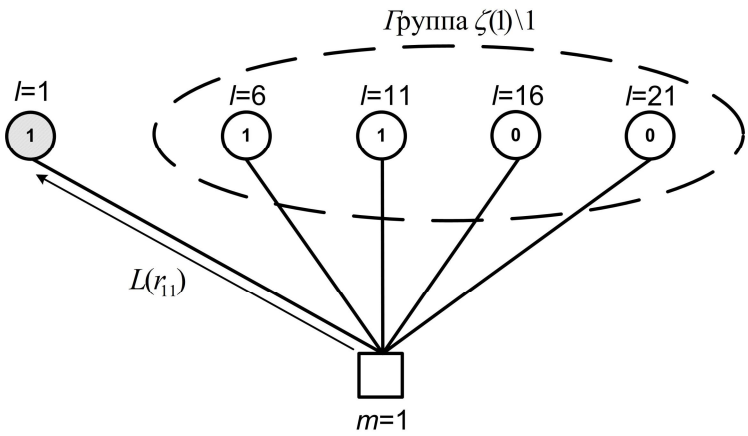

Рисунок 1.9 - Фрагмент графа Таннера для первой проверки на четность.

Предположим, что на первой итерации демодулятор приписал символам в первой проверке на четность m такие логарифмические отношения правдоподобия $L(C_l)$, что их «жесткие» решения представлены в окружностях, а первый символ пришел с ошибкой. Очевидно, что в этом случае проверка на четность не будет удовлетворена. Поправка $L(r_{11})$ для первого символьного узла от первого проверочного узла, в соответствии с теоремой Галлагера, будет зависеть от всех остальных сообщений от символьных узлов, входящих в проверку. Эти узлы входят в локализованную группу $\zeta(1)\backslash 1$, где число в скобках показывает номер проверки на четность, а число после черты показывает номер символа, которому рассчитывается поправка относительно локализованной группы. Сумма по модулю 2 символов в группе $\zeta(1)\backslash 1$ равна 0, следовательно, для удовлетворения проверки на четность первый символ так же должен быть равен 0, о чем первый проверочный узел «сообщает» первому символьному узлу посредствам поправки $L(r_{11})$.

В случае работы с логарифмическими отношениями правдоподобия по алгоритму с распространением доверия высчитывается количественная поправка $L(r_{11})$, представляющая собой логарифмическое отношение правдоподобия суммы по модулю 2 символов, входящих в группу $\zeta(1)\backslash 1$.

В соответствии с [35,64,83] логарифмическое отношение правдоподобия суммы по модулю 2 двух статистически независимых случайных величин C_1 и C_2 выражается формулой:

$$L(C_1 \oplus C_2) = \log \frac{1 + e^{L(C_1)} \cdot e^{L(C_2)}}{e^{L(C_1)} + e^{L(C_2)}}. \qquad (1.11)$$

В случае сложения по модулю 2 большего числа статистически независимых случайных величин для логарифмического отношения правдоподобия их суммы используют формулу [64,83]:

$$L(C_1 \oplus C_2 \oplus ... \oplus C_n) = 2 \cdot \text{atanh}\left(\prod_{i=1}^{n} \tanh(L(C_i)/2) \right). \quad (1.12)$$

Более строго, логарифмическая модификация алгоритма декодирования с распространением доверия описана ниже.

Начальные установки. Каждому принятому символу l приписываются логарифмические отношения правдоподобия $L(C_l)$ соответствующих символов принятого «мягкого» слова. Затем декодер осуществляет итерации, каждая из которых состоит из следующих шагов.

Шаг 1. Расчет сообщений от символьных узлов.

Для каждого ненулевого элемента проверочной матрицы вычислить [58,87]

$$L(q_{m,l}) = L(C_l) + \sum_{m' \in \mu(l) \setminus m} L(r_{m',l}). \quad (1.13)$$

Для первой итерации $L(q_{m,l})$ устанавливается как $L(C_l)$.

Графически пересчет символьных узлов можно представить на рисунке 1.10.

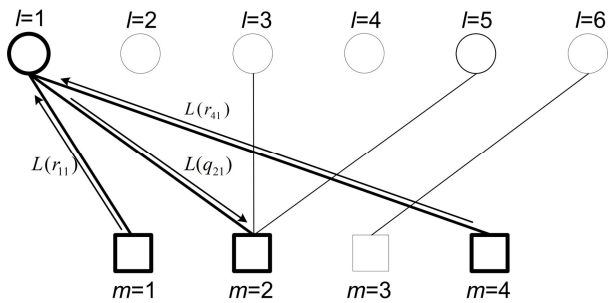

Рисунок 1.10 - Графическое представление пересчета символьных узлов графа Таннера

Шаг 2. Расчет сообщений от проверочных узлов.

Для каждого ненулевого элемента проверочной матрицы вычислить

$$L(r_{m,l}) = 2 \cdot \operatorname{atanh}\Big(\prod_{l' \in \xi(m) \setminus l} \tanh\Big(\frac{1}{2} \cdot L(q_{m,l'})\Big)\Big). \quad (1.14)$$

Графически пересчет проверочных узлов можно представить на рисунке 1.11.

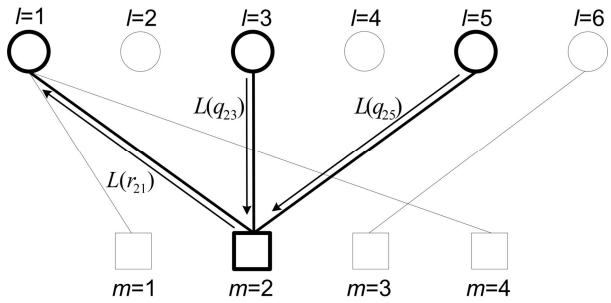

Рисунок 1.11 - Графическое представление пересчета проверочных узлов графа Таннера

Шаг 3. Вычисление апостериорных надежностей.

Для каждого l вычислить апостериорную величину $L(Q_l)$:

$$L(Q_l) = L(C_l) + \sum_{m \in \mu(l)} L(r_{m,l}). \quad (1.15)$$

Можно заметить, что обновленное «мягкое» решение $L(Q_l)$, рассчитываемое по формуле (1.15), отличается от сообщений $L(q_{m,l})$, которые пересылаются от символьных узлов графа Таннера к проверочным узлам (формула 1.13) на следующей итерации, на величину соответствующих сообщений $L(r_{m,l})$. Учитывая это, можно переписать формулу (1.13) следующим образом:

$$L(q_{m,l}) = L(Q_l) - L(r_{m,l}). \quad (1.16)$$

Подобное упрощение не влияет на помехоустойчивость и позволяет сократить количество операций сложения на этапе расчета сообщений от символьных узлов графа Таннера.

«Мягкие» апостериорные решения преобразуются в «жесткие» по следующему правилу:

$$C_l = \begin{cases} 0, & \text{если } L(Q_l) > 0 \\ 1, & \text{если } L(Q_l) \le 0 \end{cases}. \qquad (1.17)$$

Если все проверки сходятся, декодирование заканчивается. В противном случае следует вернуться к шагу 1. Результаты моделирования данного алгоритма приведены в третьей главе.

1.3.4 Семейство алгоритмов минимума суммы «Min-sum»

Процесс декодирования кодов с малой плотностью проверок на четность упирается в задачу расчета совместного логарифмического отношения правдоподобия суммы по модулю 2 ряда величин, входящих в локализованную группу. Согласно базовому алгоритму декодирования с распространением доверия, для решения этой задачи используется формула (1.12), обуславливающая необходимость использования сложных функций гиперболических тангенсов и арктангенса. В общем случае функция гиперболического тангенса рассчитывается через отношение экспонент [86]:

$$\tanh(x) = \frac{(e^{2x} - 1)}{(e^{2x} + 1)}, \qquad (1.18)$$

а функция гиперболического арктангенса через натуральный логарифм:

$$\operatorname{atanh}(x) = \frac{1}{2} \cdot \ln\left(\frac{1+x}{1-x}\right). \qquad (1.19)$$

Несмотря на то, что функции могут быть реализованы в виде таблиц [30,45,82,85] или аппроксимированы [63,70,79], в [35,64,75,83] рассмотрено упрощение расчета логарифмического отношения правдоподобия суммы по модулю 2 двух случайных статистически независимых величин, которое используется при обновлении проверочных узлов графа Таннера:

$$L(C_1 \oplus C_2) \approx \operatorname{sign}\left(L(C_1)\right) \cdot \operatorname{sign}\left(L(C_2)\right) \cdot \min\left(\left|L(C_1)\right|, \left|L(C_2)\right|\right), \qquad (1.20)$$

где оператор sign() возвращает знак аргумента, а оператор min() возвращает минимальное значение аргументов. Для большего числа слагаемых расчёт аналогичен:

$$L(C_1 \oplus C_2 \oplus ... \oplus C_n) \approx \left(\prod_{i=1}^{n} \operatorname{sign}\left(L(C_i)\right) \right) \cdot \min_{i=1...n} \left(\left| L(C_i) \right| \right). \quad (1.21)$$

Семейство алгоритмов «Min-sum» используют данный упрощенный расчет совместного логарифмического отношения правдоподобия, основанный на том, что абсолютное значение, полученное по формуле (1.12), будет всегда меньше наименьшего абсолютного значения сообщения от символьного узла, входящего в локализованную группу.

1.3.4.1 Алгоритм минимума суммы «Min-sum»

Алгоритм декодирования «Min-sum», также известный как «Минимум суммы», рассмотренный ранее в [31,38,46,66,87,89], отличается от алгоритма итеративного логарифмического распространения доверия только упрощенным в соответствии с (1.21) расчётом сообщений от проверочных узлов:

$$L(r_{m,l}) = \left(\prod_{l' \in \zeta(m) \setminus l} \operatorname{sign}\left(L\left(q_{m,l'}\right)\right) \right) \cdot \min_{l' \in \zeta(m) \setminus l} \left(\left| L\left(q_{m,l'}\right) \right| \right). \quad (1.22)$$

Благодаря упрощенному обновлению проверочных узлов графа Таннера (использование формулы 1.22 вместо 1.14) исчезает необходимость использования функций гиперболического тангенса и арктангенса. Отмечается [31,38,75], что платой за это упрощение является энергетический проигрыш при декодировании порядка 0.5 дБ по сравнению с алгоритмом итеративного распространения доверия. Результаты моделирования данного алгоритма приведены в третьей главе.

1.3.4.2 Алгоритм минимума суммы «Min-sum normalized»

Несмотря на то, что алгоритм декодирования «Min-sum» существенно проще алгоритма с распространением доверия «Belief propagation», энергетический проигрыш от применения упрощения (1.22) инициирует поиск других путей декодирования низкоплотностных кодов. Одним из таких путей является модификация алгоритма «Min-sum» в части обновления проверочных узлов графа Таннера. Можно заметить, что абсолютное значение совместного логарифмического отношения правдоподобия нескольких статистически независимых случайных величин (1.12) будет всегда меньше, чем наименьшая по модулю величина, учитываемая при расчете этого совместного логарифмического отношения правдоподобия [47,48]. Учитывая это можно брать сообщения, вычисляемые по формуле (1.22), с

определенным весом меньше 1, тем самым сокращая разницу в значениях, вычисляемых по формулам (1.22) и (1.14).

Алгоритм декодирования «Min-sum normalized», также известный как «Scaled Min-sum» является модификацией алгоритма «Min-sum» и отличается от последнего расчетом сообщений от проверочных узлов графа Таннера. Здесь, так же как и в случае декодирования по алгоритму «Min-sum», происходит расчет сообщений от проверочных узлов графа Таннера по формуле (1.22), но при этом сообщения учитывается с весом $\frac{1}{\alpha}$. Формула расчета сообщений от проверочных узлов графа Таннера имеет вид:

$$L(r_{m,l}) = \frac{1}{\alpha} \cdot \left(\prod_{l' \in \xi(m) \setminus l} \operatorname{sign}\left(L\left(q_{m,l'}\right)\right) \right) \cdot \min_{l' \in \xi(m) \setminus l} \left(\left| L\left(q_{m,l'}\right) \right| \right). \quad (1.23)$$

Подбор значения α может осуществляться с помощью процедуры, описанной в [41,48,52], или экспериментально. Анализ литературы показал [41,45,46,48], что α варьирует свое значение в диапазоне $1.25 \ldots 1.6$. При этом энергетический проигрыш по сравнению с алгоритмом «Belief propagation» по уровню 10^{-5} не превышает 0.1 дБ [45,46,47]. Результаты моделирования данного алгоритма приведены в третьей главе.

1.3.4.3 Алгоритм минимума суммы «Min-sum offset»

В отличие от алгоритма «Min-sum normalized», в версии алгоритма «Min-sum offset» вместо операции умножения на множитель $\frac{1}{\alpha}$ используется вычитание константы. Такой подход так же позволяет сократить разницу в значениях, вычисляемых по формулам (1.22) и (1.14). Формула расчета сообщений от проверочных узлов графа Таннера, в соответствии с [92], имеет вид:

$$L(r_{m,l}) = \left(\prod_{l' \in \xi(m) \setminus l} \operatorname{sign}\left(L\left(q_{m,l'}\right)\right) \right) \cdot \max\left(\left(\min_{l' \in \xi(m) \setminus l} \left(\left| L\left(q_{m,l'}\right) \right| \right) - c \right), 0 \right). \quad (1.24)$$

Как видно из формулы (1.24), в том случае, если абсолютное значение выбранной константы c превышает минимальное абсолютное значение сообщения $L\left(q_{m,l}\right)$, сообщение

$L(r_{m,l})$ обращается в 0. Эта мера необходима для того, чтобы не вносить ошибки при декодировании.

Подбор значения константы *c* может осуществляться с помощью процедуры, описанной в [48,52], или экспериментально. Анализ литературы показал [55,65,87,92], что корректирующая константа варьирует свое значение в диапазоне $0\ldots0.8$. При этом энергетический проигрыш по сравнению с алгоритмом «Belief propagation» по уровню 10^{-5} не превышает 0.05 дБ [55,65,87]. Результаты моделирования данного алгоритма приведены в третьей главе.

1.3.5 Семейство алгоритмов мажоритарного декодирования «UMP BP»

Данное семейство алгоритмов принципиально отличается от семейства «Min-sum». В случае декодирования по алгоритмам семейства «UMP BP» на каждой итерации проверки на четность «голосуют» «за» или «против» по каждому принятому символу. В качестве весов голосов используются сообщения от проверочных узлов графа Таннера, а в качестве «стартовой цены» каждого символа используются абсолютные значения априорных «мягких» решений демодулятора. Если суммы голосов «против» достаточно для того, чтобы сбить «стартовую цену» рассматриваемого символа, его «жесткое» решение инвертируется.

1.3.5.1 Мажоритарное декодирование «UMP BP»

Алгоритм взвешенного мажоритарного декодирования «Uniformly Most Powerful Belief Propagation», упоминавшийся ранее в [71,74,78], в процессе своей работы использует как «мягкие», так и «жесткие» решения демодулятора. Для описания алгоритма введем дополнительные обозначения из [78]:

x_{ml} - «жесткое» решение о символе *l* по информации от всех проверок, кроме проверки *m*.

x_l^c - «жесткое» решение по априорной информации из канала;

Начальные установки. Каждому принятому символу *l* приписываются логарифмические отношения правдоподобия $L(C_l)$ соответствующих символов принятого «мягкого» слова и величины C_l, являющиеся «жесткими» решениями принятых символов. Далее величины x_l^c устанавливаются в соответствии с «жесткими» априорными решениями C_l и не меняются на протяжении всего декодирования. Затем, декодер осуществляет итерации, каждая из которых состоит из следующих шагов.

Шаг 1. Расчет сообщений от символьных узлов.

Для каждого ненулевого элемента проверочной матрицы вычислить

$$L(q_{m,l}) = \left|L(C_l)\right| + \sum_{m' \in \mu(l) \setminus m} L(r_{m',l}) . \quad (1.25)$$

Для первой итерации $L(q_{m,l})$ устанавливается как $\left|L(C_l)\right|$.

Шаг 2. Расчет сообщений от проверочных узлов.

Для каждого ненулевого элемента проверочной матрицы вычислить

$$x_{ml} = \begin{cases} x_l^c, & \text{если } L(q_{m,l}) \geq 0 \\ x_l^c \oplus 1, & \text{если } L(q_{m,l}) < 0 \end{cases} . \quad (1.26)$$

Для каждого l - ого символа и каждой проверки m, в которой участвует символ l необходимо рассчитать величины

$$\sigma_{m,l} = x_l^c \oplus \sum_{l' \in \xi(m) \setminus l} x_{m,l'} \ \left[\mathrm{mod}\, 2\right], \quad (1.27)$$

которые впоследствии примут участие в формировании знаков для поправок $L(r_{m,l})$. Для каждого ненулевого элемента проверочной матрицы вычислить

$$L(r_{m,l}) = \begin{cases} \min\limits_{l' \in \xi(m) \setminus l} \left(\left|L\left(q_{m,l'}\right)\right|\right), & \text{если } \sigma_{m,l} = 0 \\ -\min\limits_{l' \in \xi(m) \setminus l} \left(\left|L\left(q_{m,l'}\right)\right|\right), & \text{если } \sigma_{m,l} = 1 \end{cases} . \quad (1.28)$$

Шаг 3. «Голосование» проверок на четность.

Для каждого l вычислить апостериорную величину $L(Q_l)$:

$$L(Q_l) = \left|L(C_l)\right| + \sum_{m \in \mu(l)} L(r_{m,l}) . \quad (1.29)$$

«Мягкие» апостериорные решения преобразуются в «жесткие» по следующему правилу:

$$C_l = \begin{cases} x_l^c, & \text{если } L(Q_l) \geq 0 \\ x_l^c \oplus 1, & \text{если } L(Q_l) < 0 \end{cases} . \quad (1.30)$$

Если все проверки сходятся, декодирование заканчивается. В противном случае следует вернуться к шагу 1.

Алгоритмы «UMP BP» и «Min-sum» идентичны по всем характеристикам декодирования, но обеспечивают результат разным набором элементарных операций, выполняемых в процессе декодирования.

1.3.5.2 Мажоритарное декодирование «UMP BP normalized»

По аналогии с модификацией алгоритма минимума суммы «Min-sum», алгоритм взвешенного мажоритарного декодирования «UMP BP» может быть модернизирован весовым взятием сообщений от проверочных узлов графа. Для этого вместо формулы (1.28) следует использовать:

$$L(r_{m,l}) = \begin{Bmatrix} \frac{1}{\alpha} \cdot \min\limits_{l' \in \xi(m) \setminus l} \left(\left| L\left(q_{m,l'}\right) \right| \right), & \text{если } \sigma_{m,l} = 0 \\ -\frac{1}{\alpha} \cdot \min\limits_{l' \in \xi(m) \setminus l} \left(\left| L\left(q_{m,l'}\right) \right| \right), & \text{если } \sigma_{m,l} = 1 \end{Bmatrix}. \quad (1.31)$$

Значение коэффициента α и характеристики на выходе декодера не претерпели изменений по сравнению с алгоритмом «Min-sum normalized». Результаты моделирования данного алгоритма аналогичны результатам моделирования алгоритма «Min-sum normalized», приведенным в третьей главе.

1.3.5.3 Мажоритарное декодирование «UMP BP offset»

Декодирование по этому алгоритму обуславливает необходимость использования вместо формулы (1.28) формулу:

$$L(r_{m,l}) = \begin{Bmatrix} \max\left(\left(\min\limits_{l' \in \xi(m) \setminus l} \left(\left| L\left(q_{m,l'}\right) \right| \right) - c \right), 0 \right), & \text{если } \sigma_{m,l} = 0 \\ -\max\left(\left(\min\limits_{l' \in \xi(m) \setminus l} \left(\left| L\left(q_{m,l'}\right) \right| \right) - c \right), 0 \right), & \text{если } \sigma_{m,l} = 1 \end{Bmatrix}. \quad (1.32)$$

Значение константы c и характеристики на выходе декодера не претерпели изменений по сравнению с алгоритмом «Min-sum offset». Результаты моделирования данного алгоритма аналогичны результатам моделирования алгоритма «Min-sum offset», приведенным в третьей главе.

1.3.6 Мажоритарное декодирование с варьируемым порогом

Алгоритм взвешенного мажоритарного декодирования с варьируемым порогом, будучи модификацией классического мажоритарного алгоритма «UMP BP», рассматривался ранее в [28,38]. Алгоритм имеет два отличия от классического «UMP BP».

Первое отличие касается формирования «жестких» символов $x_{m,l}$. Согласно классическому алгоритму «UMP BP» решение о формировании того или иного «жесткого» символа $x_{m,l}$ принимается по средствам сравнения сообщений от символьных узлов графа Таннера $L\left(q_{m,l}\right)$ с нулем (1.26). В этом случае нуль являлся порогом на всех итерациях декодирования. Идея мажоритарного декодирования с варьируемым порогом состоит в установлении ненулевого порога $T_{Iter} < 0$, который бы изменялся от итерации к итерации (рисунок 1.12).

На начальных итерациях сообщения $L\left(q_{m,l}\right)$ от символьных узлов графа сформированы на основе ненадежной информации $L(r_{m,l})$ от проверочных узлов графа. Зачастую это приводит к ошибочному формированию величины $x_{m,l}$, что сказывается на ошибочном формировании знака поправки к априорной надежности принятого символа. Как следствие, надежность ошибочно принятого символа может возрасти, а правильно принятого символа уменьшиться. Если на начальных итерациях порог опустить ниже нуля, негативный эффект от действия ненадежных сообщений $L(r_{m,l})$ на сообщения $L\left(q_{m,l}\right)$, и, как следствие, на $x_{m,l}$, можно смягчить или вообще нивелировать.

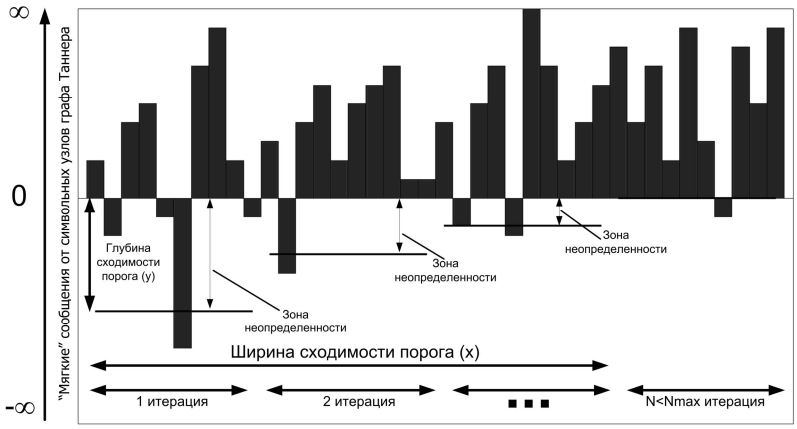

Рисунок 1.12 – Изменение порога

Второе отличие заключается в формировании «зоны неопределенности» протяженностью от 0 до порогового значения T_{Iter}. Все сообщения $L(q_{m,l})$, попавшие в эту зону в результате расчета по (1.25), устанавливаются в 0. В алгоритм необходимо добавить условие, определяющее попадание сообщения $L(q_{m,l})$ в «зону неопределенности». Условие выглядит следующим образом:

$$L(q_{m,l}) = \begin{Bmatrix} 0, & \left(L(q_{m,l}) > T_{Iter}\right) \& \left(L(q_{m,l}) < 0\right) \\ L(q_{m,l}), & в\ остальных\ случаях \end{Bmatrix}, \tag{1.33}$$

а формула (1.26) примет вид:

$$x_{ml} = \begin{Bmatrix} x_l^c, & если\ L(q_{m,l}) \geq T_{Iter} \\ x_l^c \oplus 1, & если\ L(q_{m,l}) < T_{Iter} \end{Bmatrix}. \tag{1.34}$$

Под шириной сходимости порога (x) (рисунок 1.12) понимается число итераций, на которых значение порога отлично от нуля. Под глубиной сходимости порога (y) понимается значение порога на первой итерации декодирования. Очевидно, что варьируя пару значений (x,y) можно прийти к различным результатам декодирования.

В [28,38] отмечается, что энергетический выигрыш от применения мажоритарного декодирования с варьируемым порогом по сравнению со стандартным мажоритарным алгоритмом «UMP BP» составляет 0.1 – 0.4 дБ. Следует отметить, что этот алгоритм можно модифицировать нормировкой сообщений от проверочных узлов графа Таннера или вычитанием корректирующей константы из их абсолютного значения.

В главе 3 приводится методика изменения порогового значения и результаты моделирования данного алгоритма декодирования при разных порогах, а также исследование влияния глубины порога на сходимость синдрома и среднее число итераций декодирования.

1.3.7 Реализация декодирования кусочной аппроксимацией

Помимо использования субоптимальных алгоритмов декодирования существуют другие пути упрощения процесса декодирования, позволяющие избежать аналитического расчета гиперболических функций. Одним из таких путей является аппроксимация функций $\tanh(x)$ и $\operatorname{atanh}(x)$. Линейная аппроксимация этих функций рассматривалась ранее в [63,79], а квадратичная в [70]. В [45,65,83] представлены варианты аппроксимации экспоненциальной

функции, используемой для расчета гиперболических функций. В главе 3 предлагаются различные варианты аппроксимаций гиперболических функций tanh(x) и atanh(x) и исследуется влияние ошибки аппроксимации на характеристики декодирования.

1.4 Выводы по главе

1. Анализ существующих алгоритмов декодирования показал их тесную взаимосвязь друг с другом в рамках отдельных семейств, а также принципиальные отличия семейств друг от друга.

2. Исследование работы алгоритмов декодирования позволило выявить их характерные особенности в части расчета поправок, модификация которых позволяет повысить вычислительную эффективность декодирования.

ГЛАВА 2. Оценка вычислительной сложности декодирования

В данной главе проведена оценка сложности декодирования LDPC кодов. Получены соотношения для оценки вычислительной сложности для случая регулярных и нерегулярных матриц для различных алгоритмов декодирования. На основе анализа работы алгоритмов декодирования разработаны две модификации, позволяющие использовать меньшее число элементарных операций без потери в качестве декодирования при незначительном увеличении требований к памяти для хранения внутренних переменных декодера.

Основные результаты, полученные в рамках данной главы, опубликованы автором в [12,15].

2.1 Методика оценки сложности декодирования

В качестве критерия сложности алгоритмов декодирования низкоплотностных кодов часто используют количество элементарных операций различного типа, выполняемых декодером за одну итерацию декодирования. Существующие методики оценки сложности алгоритмов декодирования по данному критерию обладают двумя недостатками.

Первый из них связан с отсутствием возможности дать количественную оценку сложности алгоритмов декодирования для случая нерегулярных матриц проверки на четность. Существующие формулы для оценки сложности предназначены для регулярных структур кодов. В случае нерегулярности проверочной матрицы низкоплотностного кода количество «1» в строках и столбцах усредняется. Такой подход позволяет дать лишь приближенную оценку вычислительной сложности декодирования.

Второй недостаток это отсутствие унификации. Под этим понимается различие в подходах к оценке сложности, при которых, например в [38], происходит оценка сложности с учетом всей итерации декодирования, а в [50,90] оценивается лишь сложность расчета сообщений от символьных и проверочных узлов графа Таннера. В [74] опускаются операции, связанные с арифметикой по модулю 2. Все это затрудняет сравнение алгоритмов декодирования низкоплотностных кодов по критерию сложности между собой.

В данной главе приводится обобщение формул расчета сложности на случай нерегулярности проверочной матрицы для различных алгоритмов декодирования и унификация подхода к оценке сложности декодирования.

Для оценки сложности алгоритмов декодирования в качестве базовой используется методика, предложенная в [38]. Данная методика в полной мере учитывает операции сложения (N_+), умножения ($N_\times$), сравнения ($N_/$), взятия модуля числа ($N_{|a|}$) и сложения по модулю 2

($N_{\oplus}$) на одну итерацию декодирования. Операция вычитания приравнивается к операции сложения. Число операций различного типа будет зависеть от «весов» строк (k) и столбцов (j) проверочной матрицы, а также от длины кодового слова (l), количества уравнений в матрице проверки на четность (m) и алгоритма декодирования.

Работу любого алгоритма декодирования кодов с малой плотностью проверок на четность можно разделить на несколько этапов, представленных на рисунке 2.1. Подсчет элементарных операций будет вестись для этапов 1-4, выполняемых итеративно.

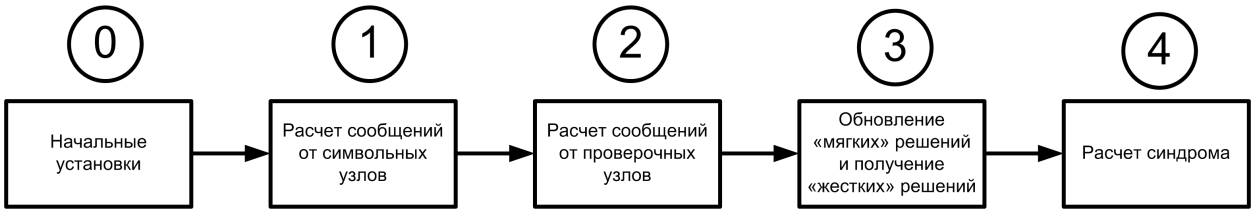

Рисунок 2.1 – Этапы декодирования LDPC кодов

2.2 Оценка сложности алгоритмов декодирования

Далее будет проведена оценка сложности для большинства вариантов декодирования кодов с малой плотностью проверок на четность.

Вычислительная сложность «жесткого» декодирования «Bit Flip» не рассматривается в данной работе. Оценки вычислительной сложности данного алгоритма и его различных модификаций проведены в [32,80].

2.2.1 Алгоритм минимума суммы «Min-sum»

Шаг 1. Расчет сообщений от символьных узлов графа Таннера.

Для первой итерации сообщения от символьных узлов $L(q_{m,l})$ устанавливаются как входные мягкие решения $L(C_l)$. На последующих итерациях на данном шаге используются только операции сложения, количество которых выражается формулой:

$$N_{+} = \sum_{i=1}^{l} (j_i - 1) \cdot j_i \,, \tag{2.1}$$

где j_i - вес i-ого столбца проверочной матрицы. Количество столбцов в матрице соответствует длине кодового слова l.

Шаг 2. Расчет сообщений от проверочных узлов графа Таннера.

Согласно формуле (1.22) для формирования сообщений от проверочных узлов к символьным узлам используются операции сравнения, умножения и взятия модуля числа. Количество умножений, выполняемых на данном этапе, выражается формулой:

$$N_{\times} = \sum_{i=1}^{m} \mathrm{k_i} \cdot (k_i - 1), \tag{2.2}$$

где k_i - вес *i*-ой строки проверочной матрицы.

Количество сравнений, выполняемых на данном этапе, выражается формулой:

$$N_{/} = \sum_{i=1}^{m} \mathrm{k_i} \cdot (2 \cdot k_i - 3). \tag{2.3}$$

Количество взятий модуля числа, выполняемых на данном этапе, выражается формулой:

$$N_{|a|} = \sum_{i=1}^{m} \mathrm{k_i} \cdot (k_i - 1). \tag{2.4}$$

Шаг 3. Обновление «мягких» решений и получение «жестких» решений.

На данном этапе используются операции сложения для обновления «мягких» решений и операции сравнения с нулем для получения «жестких» решений. Количество сложений, выполняемых при обновлении «мягких» решений, выражается формулой:

$$N_{+} = \sum_{i=1}^{l} j_i \,. \tag{2.5}$$

Количество сравнений, выполняемых для получения «жестких» решений, выражается формулой:

$$N_{/} = l \,. \tag{2.6}$$

Шаг 4. Расчет синдрома.

На данном этапе используются только операции сложения по модулю 2 для расчета синдрома. Количество сложений по модулю 2 , выполняемых на данном этапе, выражается формулой:

$$N_{\oplus} = \sum_{i=1}^{m} (k_i - 1). \tag{2.7}$$

Итоговые формулы для расчета количества операций.

Общее число операций сложения на одну итерацию декодирования выражается формулой:

$$N_{+} = \sum_{i=1}^{l} j_i^2 . \tag{2.8}$$

Общее число операций умножения на одну итерацию декодирования выражается формулой:

$$N_{\times} = \sum_{i=1}^{m} \mathrm{k_i} \cdot (k_i - 1). \tag{2.9}$$

Общее число операций сравнения на одну итерацию декодирования выражается формулой:

$$N_{/} = 1 + l + \sum_{i=1}^{m} \mathrm{k_i} \cdot (2 \cdot k_i - 3). \tag{2.10}$$

Здесь, и далее для других алгоритмов, первый член суммы, равный 1, как и в [38], учитывает сравнение счетчика итераций с максимальным числом.

Общее число операций взятия модуля числа на одну итерацию декодирования выражается формулой:

$$N_{|a|} = \sum_{i=1}^{m} \mathrm{k_i} \cdot (k_i - 1). \tag{2.11}$$

Общее число операций сложения по модулю 2 на одну итерацию декодирования выражается формулой:

$$N_{\oplus} = \sum_{i=1}^{m} (k_i - 1). \qquad (2.12)$$

Общее число операций различного типа, необходимых для выполнения одной итерации декодирования по алгоритму «Min-sum» с низкоплотностной матрицей проверки на четность из Subframe2 размерностью 600 на 1200, содержащей 4818 ненулевых элементов, представлено в таблице 2.1. Приводимые далее количественные оценки вычислительной сложности для других алгоритмов декодирования получены для этой же матрицы проверки на четность.

Таблица 2.1 – Сложность алгоритма «Min-sum»

Операция	Количество на итерацию декодирования
Сложение	37024
Умножение	33888
Сравнение	64159
Взятие модуля	33888
Сложение по модулю 2	4218

2.2.2 Алгоритм минимума суммы «Min-sum normalized»

Алгоритм «Min-sum normalized» отличается от алгоритма «Min-sum» только расчетом сообщений от проверочных узлов графа Таннера (шаг 2). Как следствие, шаги 1,3 и 4 имеют одинаковую сложность и не описываются здесь.

Шаг 2. Расчет сообщений от проверочных узлов графа Таннера.

Согласно формуле (1.23) для формирования сообщений от проверочных узлов к символьным узлам используются операции сравнения, умножения и взятия модуля числа. Количество умножений, выполняемых на данном шаге, выражается формулой:

$$N_{\times} = \sum_{i=1}^{m} \mathrm{k}^2{}_{\mathrm{i}} . \qquad (2.13)$$

Количество сравнений и взятий модуля числа на данном шаге не претерпело изменений по сравнению с алгоритмом «Min - sum» и может быть вычислено по формулам (2.3) и (2.4), соответственно.

Итоговые формулы для расчета количества операций.

Общее число операций сложения на одну итерацию декодирования выражается формулой:

$$N_{+} = \sum_{i=1}^{l} j_i^2 \,. \tag{2.14}$$

Общее число операций умножения на одну итерацию декодирования выражается формулой:

$$N_{\times} = \sum_{i=1}^{m} \mathrm{k}^2{}_{\mathrm{i}} \,. \tag{2.15}$$

Общее число операций сравнения на одну итерацию декодирования выражается формулой:

$$N_{/} = 1 + l + \sum_{i=1}^{m} \mathrm{k}_{\mathrm{i}} \cdot (2 \cdot k_i - 3). \tag{2.16}$$

Общее число операций взятия модуля числа на одну итерацию декодирования выражается формулой:

$$N_{|a|} = \sum_{i=1}^{m} \mathrm{k}_{\mathrm{i}} \cdot (k_i - 1). \tag{2.17}$$

Общее число операций сложения по модулю 2 на одну итерацию декодирования выражается формулой:

$$N_{\oplus} = \sum_{i=1}^{m} (k_i - 1) \,. \tag{2.18}$$

Общее число операций различного типа, необходимых для выполнения одной итерации декодирования по алгоритму «Min-sum normalized», представлено в таблице 2.2.

Таблица 2.2 – Сложность алгоритма «Min-sum normalized»

Операция	Количество на итерацию декодирования
Сложение	37024
Умножение	38706
Сравнение	64159
Взятие модуля	33888
Сложение по модулю 2	4218

Сложность алгоритма «Min-sum normalized» отличается от сложности алгоритма «Min-sum» на число умножений, равное количеству элементов в проверочной матрице низкоплотностного кода, то есть на количество отнормированных сообщений, которое формируется от проверочных узлов графа Таннера к символьным узлам.

2.2.3 Алгоритм минимума суммы «Min-sum offset»

Алгоритм «Min-sum offset» отличается от алгоритма «Min-sum» только расчетом сообщений от проверочных узлов графа Таннера (шаг 2). Как следствие, шаги 1,3 и 4 имеют одинаковую сложность и не описываются здесь.

Шаг 2. Расчет сообщений от проверочных узлов графа Таннера.

Согласно формуле (1.24) для формирования сообщений от проверочных узлов к символьным узлам используются операции сложения, сравнения, умножения и взятия модуля числа.

Количество сложений, выполняемых на данном шаге, выражается формулой:

$$N_{+} = \sum_{i=1}^{m} k_i \, . \tag{2.19}$$

Количество сравнений, выполняемых на данном шаге, выражается формулой:

$$N_{/} = \sum_{i=1}^{m} \mathrm{k_i} \cdot \left(2 \cdot k_i - 2\right). \tag{2.20}$$

Количество умножений и взятий модуля числа на данном шаге не претерпело изменений по сравнению с алгоритмом «Min - sum» и может быть вычислено по формулам (2.2) и (2.4), соответственно.

Итоговые формулы для расчета количества операций.

Общее число операций сложения на одну итерацию декодирования выражается формулой:

$$N_{+} = \sum_{i=1}^{l} j_i^2 + \sum_{i=1}^{m} k_i \, . \tag{2.21}$$

Общее число операций умножения на одну итерацию декодирования выражается формулой:

$$N_{\times} = \sum_{i=1}^{m} \mathrm{k_i} \cdot (k_i - 1). \tag{2.22}$$

Общее число операций сравнения на одну итерацию декодирования выражается формулой:

$$N_{/} = 1 + l + \sum_{i=1}^{m} \mathrm{k_i} \cdot (2 \cdot k_i - 2). \tag{2.23}$$

Общее число операций взятия модуля числа на одну итерацию декодирования выражается формулой:

$$N_{|a|} = \sum_{i=1}^{m} \mathrm{k_i} \cdot (k_i - 1). \tag{2.24}$$

Общее число операций сложения по модулю 2 на одну итерацию декодирования выражается формулой:

$$N_{\oplus} = \sum_{i=1}^{m} (k_i - 1). \tag{2.25}$$

Общее число операций различного типа, необходимых для выполнения одной итерации декодирования по алгоритму «Min-sum offset», представлено в таблице 2.3.

Таблица 2.3 – Сложность алгоритма «Min-sum offset»

Операция	Количество на итерацию декодирования
Сложение	41842
Умножение	33888
Сравнение	68977
Взятие модуля	33888
Сложение по модулю 2	4218

Сложность алгоритма «Min-sum offset» отличается от сложности алгоритма «Min-sum» на число сложений и сравнений, каждое из которых равно количеству элементов в проверочной матрице низкоплотностного кода, то есть количеству сообщений, которое формируется от проверочных узлов графа Таннера к символьным узлам.

2.2.4 Мажоритарное декодирование «UMP BP»

Перед оценкой вычислительной сложности алгоритма «UMP BP» следует отметить, что в формуле расчета сообщений от символьных узлов графа Таннера (1.25) и при «голосовании» (1.29) участвует абсолютное значение априорной надежности, которое было приписано демодулятором каждому принятому символу перед декодированием. Как отмечено в [78], целесообразно заранее рассчитать это абсолютное значение на этапе инициализации алгоритма, чтобы во время работы алгоритма с каждой новой итерацией не проделывать операцию взятия модуля числа. Шаги 1 и 4 алгоритма мажоритарного декодирования «UMP BP» аналогичны шагам алгоритма «Min-sum». Как следствие, шаги имеют одинаковую сложность и не описываются здесь.

Шаг 2. Расчет сообщений от проверочных узлов графа Таннера.

Для формирования сообщений от проверочных узлов графа Таннера алгоритм последовательно формирует 3 величины. Вначале по формуле (1.26) формируется массив $x_{m,l}$. Учитывая, что все сообщения от всех символьных узлов графа могут иметь отрицательную полярность, для вычисления величин $x_{m,l}$ потребуются операции сравнения и сложения по модулю 2.

Количество сравнений для вычисления массива $x_{m,l}$ выражается формулой:

$$N_{/} = \sum_{i=1}^{m} \mathrm{k_i} . \tag{2.26}$$

Количество сложений по модулю 2 для вычисления массива $x_{m,l}$ выражается формулой:

$$N_{\oplus} = \sum_{i=1}^{m} \mathrm{k_i} . \tag{2.27}$$

Далее, по формуле (1.27) происходит расчет массива $\sigma_{m,l}$. Используются операции сложения по модулю 2. Количество сложений по модулю 2 для вычисления массива $\sigma_{m,l}$ выражается формулой:

$$N_{\oplus} = \sum_{i=1}^{m} \mathrm{k_i} \cdot (\mathrm{k_i} - 1). \quad (2.28)$$

После этого следует расчет сообщений от проверочных узлов графа Таннера по формуле (1.28). Используются операции сравнения и взятия модуля числа.

Количество сравнений для вычисления поправок $L(r_{m,l})$ складывается из количества сравнений, необходимых для поиска минимального элемента в проверке, и из количества сравнений, определяющих знак поправки $L(r_{m,l})$ в зависимости от значения $\sigma_{m,l}$. В результате, формула для оценки количества сравнений при вычислении поправок $L(r_{m,l})$ примет вид:

$$N_{/} = \sum_{i=1}^{m} (\mathrm{k_i} \cdot (\mathrm{k_i} - 2) + \mathrm{k_i}). \quad (2.29)$$

Количество взятий модуля числа, необходимых для вычисления поправок $L(r_{m,l})$ оценивается следующей формулой:

$$N_{|a|} = \sum_{i=1}^{m} \mathrm{k_i} \cdot (k_i - 1). \quad (2.30)$$

Общее число операций сравнения на данном шаге выражается формулой:

$$N_{/} = \sum_{i=1}^{m} k_i^2 \ . \quad (2.31)$$

Общее число операций сложения по модулю 2 на данном шаге выражается формулой:

$$N_{\oplus} = \sum_{i=1}^{m} k_i^2 \ . \quad (2.32)$$

Общее число операций взятия модуля числа на данном шаге выражается формулой:

$$N_{|a|} = \sum_{i=1}^{m} \mathrm{k_i} \cdot (\mathrm{k_i} - 1). \quad (2.33)$$

Шаг 3. «Голосование» проверок на четность и получение «жестких» решений.

Согласно формуле (1.29) на данном этапе используются операции сложения, число которых выражается формулой:

$$N_{+} = \sum_{i=1}^{l} j_i \,. \tag{2.34}$$

Согласно формуле (1.30) на данном этапе используются операции сравнения и операции сложения по модулю 2 для получения «жестких» решений. Количество сравнений, выполняемых на данном шаге, выражается формулой:

$$N_{/} = l \,. \tag{2.35}$$

Количество сложений по модулю 2, выполняемых на данном шаге, выражается формулой:

$$N_{\oplus} = l \,. \tag{2.36}$$

Итоговые формулы для расчета количества операций.

Общее число операций сложения на одну итерацию декодирования выражается формулой:

$$N_{+} = \sum_{i=1}^{l} j_i^2 \,. \tag{2.37}$$

Общее число операций сравнения на одну итерацию декодирования выражается формулой:

$$N_{/} = 1 + l + \sum_{i=1}^{m} \mathrm{k}_\mathrm{i}^2 \,. \tag{2.38}$$

Общее число операций взятия модуля числа на одну итерацию декодирования выражается формулой:

$$N_{|a|} = \sum_{i=1}^{m} \mathrm{k_i} \cdot (k_i - 1). \quad (2.39)$$

Общее число операций сложения по модулю 2 на одну итерацию декодирования выражается формулой:

$$N_{\oplus} = \sum_{i=1}^{m} (k_i^2 + k_i - 1) + l\,. \quad (2.40)$$

Общее число операций различного типа, необходимых для выполнения одной итерации декодирования по алгоритму «UMP BP», представлено в таблице 2.4.

Таблица 2.4 – Сложность алгоритма «UMP BP»

Операция	Количество на итерацию декодирования
Сложение	37024
Сравнение	39907
Взятие модуля	33888
Сложение по модулю 2	44124

2.2.5 Мажоритарное декодирование «UMP BP normalized»

Алгоритм «UMP BP normalized» отличается от алгоритма «UMP BP» только расчетом сообщений от проверочных узлов графа Таннера (шаг 2). Как следствие, шаги 1,3 и 4 имеют одинаковую сложность и не описываются здесь.

Шаг 2. Расчет сообщений от проверочных узлов графа Таннера.

В соответствии с алгоритмом «UMP BP normalized» сообщения от проверочных узлов, рассчитанные алгоритмом «UMP BP» по формуле (1.28), нормируются коэффициентом $\frac{1}{\alpha}$. При условии, что дробь может быть рассчитана заранее, нормировка добавляет по 1-ой операции умножения на каждое рассчитываемое сообщение от проверочных узлов. Таким образом, количество операций умножения на шаге 2 выражается формулой:

$$N_{\times} = \sum_{i=1}^{m} \mathrm{k_i}\,. \quad (2.41)$$

Общее число операций сравнения, сложений по модулю 2 и взятий модуля числа не претерпело изменений по сравнению с алгоритмом «UMP BP» и выражается формулами (2.31), (2.32) и (2.33), соответственно.

Итоговые формулы для расчета количества операций.

Общее число операций сложения на одну итерацию декодирования выражается формулой:

$$N_{+} = \sum_{i=1}^{l} j_i^2 \ . \tag{2.42}$$

Общее число операций умножения на одну итерацию декодирования выражается формулой:

$$N_{\times} = \sum_{i=1}^{m} \mathrm{k_i} \ . \tag{2.43}$$

Общее число операций сравнения на одну итерацию декодирования выражается формулой:

$$N_{/} = 1 + l + \sum_{i=1}^{m} \mathrm{k_i^2} \ . \tag{2.44}$$

Общее число операций взятия модуля числа на одну итерацию декодирования выражается формулой:

$$N_{|a|} = \sum_{i=1}^{m} \mathrm{k_i} \cdot (k_i - 1). \tag{2.45}$$

Общее число операций сложения по модулю 2 на одну итерацию декодирования выражается формулой:

$$N_{\oplus} = \sum_{i=1}^{m} (k_i^2 + k_i - 1) + l \ . \tag{2.46}$$

Общее число операций различного типа, необходимых для выполнения одной итерации декодирования по алгоритму «UMP BP normalized», представлено в таблице 2.5.

Таблица 2.5 – Сложность алгоритма «UMP BP normalized»

Операция	Количество на итерацию декодирования
Сложение	37024
Умножение	4818
Сравнение	39907
Взятие модуля	33888
Сложение по модулю 2	44124

Сложность алгоритма «UMP BP normalized» отличается от сложности алгоритма «UMP BP» на число умножений, которое равно количеству элементов в проверочной матрице низкоплотностного кода, то есть количеству отнормированных сообщений, которое формируется от проверочных узлов графа Таннера к символьным узлам.

2.2.6 Мажоритарное декодирование «UMP BP offset»

Алгоритм «UMP offset» отличается от алгоритма «UMP BP» только расчетом сообщений от проверочных узлов графа Таннера (шаг 2). Как следствие, шаги 1,3 и 4 имеют одинаковую сложность и не описываются здесь.

Шаг 2. Расчет сообщений от проверочных узлов графа Таннера.

В соответствии с алгоритмом «UMP BP offset» значения сообщений от проверочных узлов, рассчитанные алгоритмом «UMP BP» по формуле (1.28), должны быть скорректированы константой *c*. Разность, получившуюся в результате такой коррекции, как и в алгоритме «Min-sum offset», необходимо сравнить с нулем. Таким образом, по сравнению с алгоритмом «UMP BP», в его модернизированной версии «UMP BP offset» на данном шаге добавятся операции сложения и сравнения.

Количество операций сложения, выполняемых на данном шаге, оценивается формулой:

$$N_{+} = \sum_{i=1}^{m} \mathrm{k_i} \,. \tag{2.47}$$

Количество операций сравнения, выполняемых на данном шаге, оценивается формулой:

$$N_{/} = \sum_{i=1}^{m} \left(\mathrm{k_i}^2 + k_i \right). \tag{2.48}$$

Общее число операций сложения по модулю 2 и взятий модуля числа не претерпело изменений по сравнению с алгоритмом «UMP BP» и выражается формулами (2.32) и (2.33), соответственно.

Итоговые формулы для расчета количества операций.

Общее число операций сложения на одну итерацию декодирования выражается формулой:

$$N_{+} = \sum_{i=1}^{l} j_i^2 + \sum_{i=1}^{m} k_i \,. \tag{2.49}$$

Общее число операций сравнения на одну итерацию декодирования выражается формулой:

$$N_{/} = 1 + l + \sum_{i=1}^{m} \left(\mathrm{k}_\mathrm{i}^2 + k_i \right). \tag{2.50}$$

Общее число операций взятия модуля числа на одну итерацию декодирования выражается формулой:

$$N_{|a|} = \sum_{i=1}^{m} \mathrm{k}_\mathrm{i} \cdot \left(k_i - 1 \right). \tag{2.51}$$

Общее число операций сложения по модулю 2 на одну итерацию декодирования выражается формулой:

$$N_{\oplus} = \sum_{i=1}^{m} (k_i^2 + k_i - 1) + l \,. \tag{2.52}$$

Общее число операций различного типа, необходимых для выполнения одной итерации декодирования по алгоритму «UMP BP offset», представлено в таблице 2.6.

Таблица 2.6 – Сложность алгоритма «UMP BP offset»

Операция	Количество на итерацию декодирования
Сложение	41842
Сравнение	44725
Взятие модуля	33888
Сложение по модулю 2	44124

Сложность алгоритма «UMP BP offset» отличается от сложности алгоритма «UMP BP» на число сложений и сравнений, каждое из которых равно количеству элементов в проверочной матрице низкоплотностного кода, то есть количеству сообщений, которое формируется от проверочных узлов графа Таннера к символьным узлам.

2.2.7 Мажоритарное декодирование с варьируемым порогом

Данный алгоритм декодирования отличается от стандартного алгоритма «UMP BP» расчетом сообщений от символьных и проверочных узлов графа Таннера (шаг 1,2). Как следствие, шаги 3 и 4 имеют одинаковую сложность и не описываются здесь. Справедливо заметить: на шаге 2 алгоритм «UMP BP» и его модификация с варьируемым порогом сравнивают величины сообщений от символьных узлов графа с разными порогами, однако количество таких сравнений будет одинаково. Как следствие, шаг 2 так же имеет идентичную сложность и не описывается здесь.

Шаг 1. Расчет сообщений от символьных узлов графа Таннера.

Значение каждого сообщения от символьных узлов необходимо проверить двойным условием на попадание в «зону неопределенности» согласно формуле (1.33), что добавляет 2 дополнительные операции сравнения на каждое сообщение от символьного узла на данном шаге.

$$N_{/} = \sum_{i=1}^{m} 2 \cdot \mathrm{k_i} \, . \tag{2.53}$$

Число операций сложения на данном шаге не претерпело изменений по сравнению со стандартным алгоритмом «UMP BP» и рассчитывается по формуле (2.1).

Итоговые формулы для расчета количества операций.

Общее число операций сложения на одну итерацию декодирования выражается формулой:

$$N_{+} = \sum_{i=1}^{l} j_i^2 \ . \tag{2.54}$$

Общее число операций сравнения на одну итерацию декодирования выражается формулой:

$$N_{/} = 1 + l + \sum_{i=1}^{m} \left(\mathrm{k}_\mathrm{i}^2 + 2 \cdot \mathrm{k}_i \right). \tag{2.55}$$

Общее число операций взятия модуля числа на одну итерацию декодирования выражается формулой:

$$N_{|a|} = \sum_{i=1}^{m} \mathrm{k}_\mathrm{i} \cdot \left(k_i - 1 \right). \tag{2.56}$$

Общее число операций сложения по модулю 2 на одну итерацию декодирования выражается формулой:

$$N_{\oplus} = \sum_{i=1}^{m} (k_i^2 + k_i - 1) + l \ . \tag{2.57}$$

Общее число операций различного типа, необходимых для выполнения одной итерации декодирования по данному алгоритму, представлено в таблице 2.7.

Таблица 2.7 – Сложность мажоритарного алгоритма с варьируемым порогом

Операция	Количество на итерацию декодирования
Сложение	37024
Сравнение	49543
Взятие модуля	33888
Сложение по модулю 2	44124

Сложность алгоритма взвешенного мажоритарного декодирования с варьируемым порогом отличается от сложности стандартного мажоритарного алгоритма «UMP BP» на число сравнений, которое равно удвоенному количеству элементов в проверочной матрице

низкоплотностного кода. Это связано с дополнительным двойным условием, проверяющим попадание сообщения от символьного узла графа Таннера в «зону неопределенности».

2.2.8 Мажоритарное декодирование с варьируемым порогом и нормировкой

По аналогии с алгоритмами «UMP BP» и «UMP BP normalized», алгоритм взвешенного мажоритарного декодирования с варьируемым порогом и его нормированная версия отличаются только количеством умножений на шаге расчета сообщений от проверочных узлов графа Таннера (шаг 2). Как следствие, шаги 1,3 и 4 имеют одинаковую сложность и не описываются здесь.

Шаг 2. Расчет сообщений от проверочных узлов графа Таннера.

В соответствии с данным алгоритмом сообщения от проверочных узлов нормируются коэффициентом $\frac{1}{\alpha}$. Количество операций умножения на шаге 2 выражается формулой:

$$N_{\times} = \sum_{i=1}^{m} \mathrm{k}_{\mathrm{i}} \ . \tag{2.58}$$

Общее число операций сравнения, сложений по модулю 2 и взятий модуля числа на данном шаге не претерпело изменений по сравнению с алгоритмом «UMP BP» и выражается формулами (2.31), (2.32) и (2.33), соответственно.

Итоговые формулы для расчета количества операций.

Общее число операций сложения на одну итерацию декодирования выражается формулой:

$$N_{+} = \sum_{i=1}^{l} j_i^2 \ . \tag{2.59}$$

Общее число операций умножения на одну итерацию декодирования выражается формулой:

$$N_{\times} = \sum_{i=1}^{m} \mathrm{k}_{\mathrm{i}} \ . \tag{2.60}$$

Общее число операций сравнения на одну итерацию декодирования выражается формулой:

$$N_{/} = 1 + l + \sum_{i=1}^{m} \left(\mathrm{k}_{\mathrm{i}}^{2} + 2 \cdot \mathrm{k}_{i} \right). \quad (2.61)$$

Общее число операций взятия модуля числа на одну итерацию декодирования выражается формулой:

$$N_{|a|} = \sum_{i=1}^{m} \mathrm{k}_{\mathrm{i}} \cdot \left(k_i - 1 \right). \quad (2.62)$$

Общее число операций сложения по модулю 2 на одну итерацию декодирования выражается формулой:

$$N_{\oplus} = \sum_{i=1}^{m} (k_i^{2} + k_i - 1) + l \,. \quad (2.63)$$

Общее число операций различного типа, необходимых для выполнения одной итерации декодирования по данному алгоритму, представлено в таблице 2.8. Сложность алгоритма взвешенного мажоритарного декодирования с варьируемым порогом и нормировкой отличается от сложности аналогичного алгоритма без нормировки на число умножений, которое равно количеству элементов в проверочной матрице низкоплотностного кода.

Таблица 2.8 – Сложность мажоритарного алгоритма с варьируемым порогом и нормировкой

Операция	Количество на итерацию декодирования
Сложение	37024
Умножение	4818
Сравнение	49543
Взятие модуля	33888
Сложение по модулю 2	44124

2.2.9 Мажоритарное декодирование с варьируемым порогом и сдвигом

По аналогии с алгоритмами «UMP BP» и «UMP BP offset», мажоритарный алгоритм с варьируемым порогом и его модификация со сдвигом отличаются количеством сложений и

сравнений на шаге расчета сообщений от проверочных узлов графа Таннера (шаг 2). Как следствие, шаги 1,3 и 4 имеют одинаковую сложность и не описываются здесь.

Шаг 2. Расчет сообщений от проверочных узлов графа Таннера.

Количество операций сложения, выполняемых на данном шаге, оценивается формулой:

$$N_{+} = \sum_{i=1}^{m} \mathrm{k_i} \, . \tag{2.64}$$

Количество операций сравнения, выполняемых на данном шаге, оценивается формулой:

$$N_{/} = \sum_{i=1}^{m} \left(\mathrm{k_i}^2 + k_i \right). \tag{2.65}$$

Общее число операций сложения по модулю 2 и взятий модуля числа не претерпело изменений по сравнению с алгоритмом «UMP BP» и выражается формулами (2.32) и (2.33), соответственно.

Итоговые формулы для расчета количества операций.

Общее число операций сложения на одну итерацию декодирования выражается формулой:

$$N_{+} = \sum_{i=1}^{l} j_i^2 + \sum_{i=1}^{m} k_i \, . \tag{2.66}$$

Общее число операций сравнения на одну итерацию декодирования выражается формулой:

$$N_{/} = 1 + l + \sum_{i=1}^{m} \left(\mathrm{k_i}^2 + 3 \cdot \mathrm{k}_i \right). \tag{2.67}$$

Общее число операций взятия модуля числа на одну итерацию декодирования выражается формулой:

$$N_{|a|} = \sum_{i=1}^{m} \mathrm{k_i} \cdot (k_i - 1). \quad (2.68)$$

Общее число операций сложения по модулю 2 на одну итерацию декодирования выражается формулой:

$$N_{\oplus} = \sum_{i=1}^{m} (k_i^2 + k_i - 1) + l\,. \quad (2.69)$$

Общее число операций различного типа, необходимых для выполнения одной итерации декодирования по данному алгоритму, представлено в таблице 2.9.

Таблица 2.9 – Сложность мажоритарного алгоритма с варьируемым порогом и сдвигом

Операция	Количество на итерацию декодирования
Сложение	41842
Сравнение	54361
Взятие модуля	33888
Сложение по модулю 2	44124

Сложность алгоритма взвешенного мажоритарного декодирования с варьируемым порогом и сдвигом отличается от сложности аналогичного алгоритма без сдвига на число сложений и сравнений, каждое из которых равно количеству элементов в проверочной матрице низкоплотностного кода, то есть количеству сообщений, которое формируется от проверочных узлов графа Таннера к символьным узлам.

2.3 Оценка сложности алгоритма «Belief propagation» с линейной аппроксимацией

Шаги 1,3 и 4 алгоритма с распространением доверия «Belief propagation» совпадают с аналогичными шагами алгоритма минимума суммы «Min-sum». Далее будет рассмотрен шаг 2, на котором происходит расчет совместного логарифмического отношения правдоподобия локализованной группы $\zeta(1)\setminus 1$ на рисунке 1.9 посредствам кусочной линейной аппроксимации.

Шаг 2. Расчет сообщений от проверочных узлов графа Таннера.

В соответствии с алгоритмом распространения доверия на данном шаге используются операции умножения, а также расчет значений функций гиперболического тангенса и арктангенса. В алгоритме распространения доверия с аппроксимацией эти функции представляются линейными отрезками. Как следствие, для расчета значения

аппроксимированной функции, представляющей из себя несколько прямых линий, понадобится 1 операция умножения и 1 сложение для решения уравнения прямой первого порядка. Для определения отрезка линии, который аппроксимирует регион аргумента функции, потребуются операции сравнения, причем, чем сложнее аппроксимация, тем больше операций сравнения понадобится. Оценка сложности будет проведена для варианта аппроксимации (*tanh(x)* 2 + *atanh(x)* 3) и для варианта (*tanh(x)* 4 + *atanh(x)* 5) (рис.3.28). Для простоты варианты обозначены как **A** и **B**, соответственно.

На рисунках 2.2 и 2.3 представлены блок схемы алгоритмов для расчета значений аппроксимированных гиперболических функций тангенса и арктангенса для варианта **A**. На вход алгоритма расчета приближенного значения *tanh(x)* подается величина *x*. Вначале алгоритм берет модуль этого значения (используется свойство нечетности функции), формируя положительную величину *arg*. Далее происходит поиск участка линейной аппроксимации, который описывает область аргумента *arg*. При поиске происходит сравнение *arg* с границами линейных участков аппроксимаций. После этого алгоритм рассчитывает значение функции гиперболического тангенса *tanh(arg)* посредствам 1 умножения и 1 сложения (по уравнению прямой $tanh(arg) = k \cdot arg + b$). Результат расчета умножается на -1 в случае, если на входе алгоритма $x < 0$, и выдается на выход. Оценка сложности проводится для самого ресурсоемкого прохода по блок схемам алгоритма.

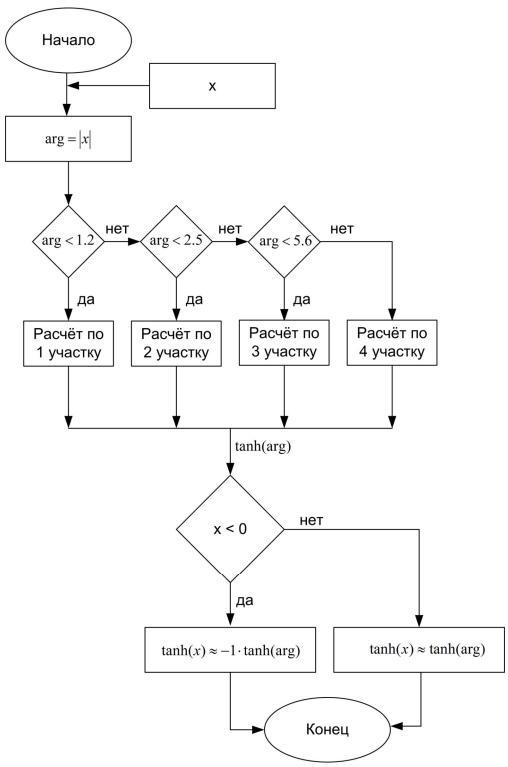

Рисунок 2.2 – Алгоритм расчета tanh(x) для варианта **A**

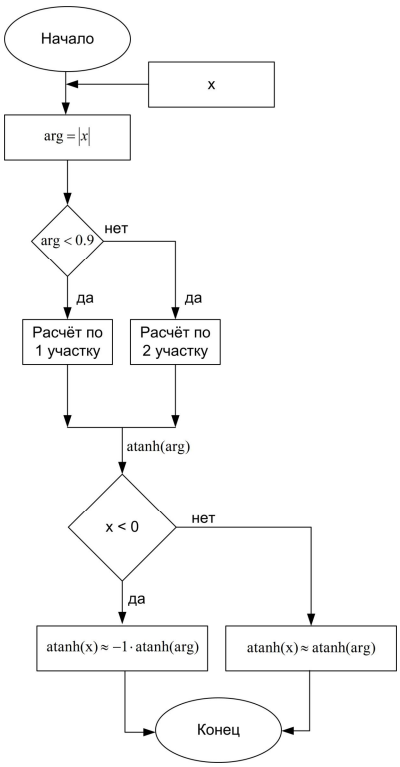

Рисунок 2.3 – Алгоритм расчета atanh(x) для варианта **A**

Аналогичные рассуждения справедливы для расчета значения функции *atanh(x)*. Блок схемы для варианта аппроксимации **B** приведены на рисунке 2.4 и 2.5, соответственно.

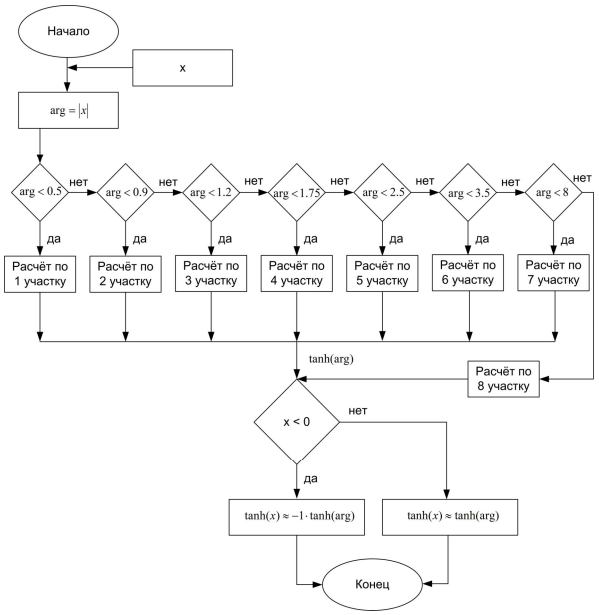

Рисунок 2.4 – Алгоритм расчета tanh(x) для варианта **B**

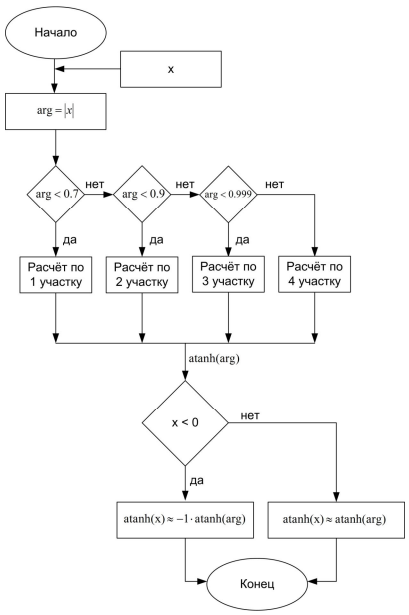

Рисунок 2.5 – Алгоритм расчета atanh(x) для варианта **B**

Необходимое число операций различного типа для расчета значений функций гиперболического тангенса и арктангенса для вариантов аппроксимаций **A** и **B** приведено в таблице 2.10.

Таблица 2.10 – Количество операций для расчета гиперболических функций

Операция	Вариант **A**		Вариант **B**	
	tanh(x)	atanh(x)	tanh(x)	atanh(x)
Сложение	1	1	1	1
Умножение	2	2	2	2
Сравнение	4	2	8	4
Взятие модуля	1	1	1	1

Согласно (1.14) перед расчетом значения функции *tanh(x)* аргумент функции делится на 2, а после расчета значения функции *atanh(x)* результат умножается на 2. Количество операций сложения, умножения и взятия модуля числа для всех вариантов аппроксимации одинаково и выражается формулами:

$$N_{+} = \sum_{i=1}^{m} k_i^2 , \qquad (2.70)$$

$$N_{\times} = \sum_{i=1}^{m} 2 \cdot k_i \cdot (2 \cdot k_i - 1), \tag{2.71}$$

$$N_{|a|} = \sum_{i=1}^{m} k_i^2 \ . \tag{2.72}$$

Число операций сравнения для варианта аппроксимации **A** выражается формулой:

$$N_{/A} = \sum_{i=1}^{m} 2 \cdot k_i \cdot (2 \cdot k_i - 1). \tag{2.73}$$

Число операций сравнения для варианта аппроксимации **B** вдвое больше:

$$N_{/B} = \sum_{i=1}^{m} 4 \cdot k_i \cdot (2 \cdot k_i - 1). \tag{2.74}$$

Итоговые формулы для расчета количества операций:

Общее число операций сложения на одну итерацию декодирования выражается формулой:

$$N_{+} = \sum_{i=1}^{j} j_i^2 + \sum_{i=1}^{m} k_i^2 \ . \tag{2.75}$$

Общее число операций умножения на одну итерацию декодирования выражается формулой:

$$N_{\times} = \sum_{i=1}^{m} 2 \cdot k_i \cdot (2 \cdot k_i - 1). \tag{2.76}$$

Общее число операций сравнения на одну итерацию декодирования для вариантов аппроксимации **A** и **B** выражается формулами:

$$N_{/A} = 1 + l + \sum_{i=1}^{m} 2 \cdot k_i \cdot (2 \cdot k_i - 1). \tag{2.77}$$

$$N_{/B} = 1 + l + \sum_{i=1}^{m} 4 \cdot k_i \cdot (2 \cdot k_i - 1). \quad (2.78)$$

Общее число операций взятия модуля числа на одну итерацию декодирования выражается формулой:

$$N_{|a|} = \sum_{i=1}^{m} k_i^2 \ . \quad (2.79)$$

Общее число операций сложения по модулю 2 на одну итерацию декодирования выражается формулой:

$$N_{\oplus} = \sum_{i=1}^{m} (k_i - 1) \, . \quad (2.80)$$

Общее число операций различного типа, необходимых для выполнения одной итерации декодирования, представлено в таблице 2.11.

Таблица 2.11 – Сложность декодирования с аппроксимацией

Операция	Количество на итерацию декодирования	
	Вариант **A**	Вариант **B**
Сложение	75730	75730
Умножение	145188	145188
Сравнение	146389	291577
Взятие модуля	38706	38706
Сложение по модулю 2	4218	4218

Можно отметить, что алгоритм с более точной аппроксимацией в среднем требует большее число операций сравнения для поиска региона, к которому относится аргумент рассчитываемой функции. Алгоритм с неточной аппроксимацией имеет меньше кусочных регионов. Как следствие, количество сравнений для поиска региона рассчитываемого аргумента функции у такого алгоритма меньше.

2.4 Сравнительный анализ сложности алгоритмов декодирования

В результате проведенной оценки сложности алгоритмов декодирования для каждого из них было получено количество операций сложения, умножения, сравнения, взятия модуля

числа и сложения по модулю 2, выполняемых за одну итерацию декодирования для рассматриваемой матрицы проверки на четность. Результаты сведены в таблицу 2.12.

В рамках семейства «Min-sum» самым простым алгоритмом является алгоритм минимума суммы «Min-sum». Алгоритм «Min-sum normalized» сложнее алгоритма «Min-sum» на 4818 операций умножения, так как именно столько сообщений от проверочных узлов графа Таннера необходимо отнормировать для исследуемой в данной работе матрицы проверки на четность. Алгоритм «Min-sum offset» сложнее алгоритма «Min-sum» на 4818 сложений и 4818 сравнений. Добавление 4818 сложений обусловлено тем, что каждое сообщение от проверочного узла графа Таннера необходимо скорректировать константой (вычесть константу из абсолютного значения). Добавление 4818 сравнений обусловлено тем, что результат разности необходимо сравнить с нулем. В рамках семейства «UMP BP» между алгоритмами «UMP BP», «UMP BP normalized» и «UMP BP offset» аналогичный паритет.

Мажоритарные алгоритмы с варьируемым порогом, будучи модификациями алгоритмов семейства «UMP BP», требуют больше операций сравнения за счет необходимости дополнительной проверки попадания сообщения от символьного узла графа Таннера в «зону неопределенности».

Анализ показал, что кусочная линейная аппроксимация не является хорошей альтернативой другим алгоритмам декодирования по критерию сложности.

На рисунке 2.6 представлена гистограмма, отображающая суммарную сложность рассмотренных выше алгоритмов декодирования. В целом, семейство мажоритарных алгоритмов «UMP BP» демонстрирует меньшую суммарную сложность, чем семейство алгоритмов «Min-sum». Однако важно учитывать, что в данном случае суммирование количества операций разного типа произведено с «весом» 1. В общем случае «веса» типов операций должны подбираться индивидуально под конкретную аппаратуру.

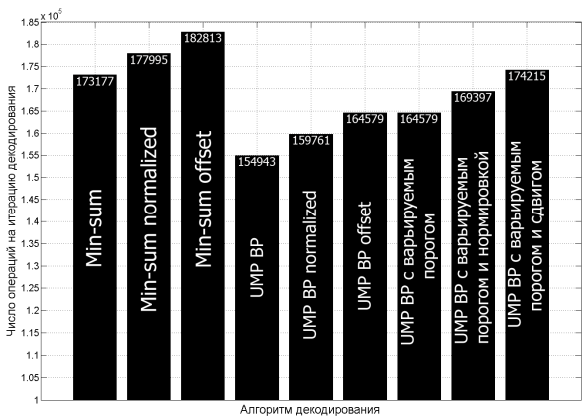

Рисунок 2.6 – Сложность различных алгоритмов декодирования LDPC кодов

Таблица 2.12 – Сводная таблица сложности алгоритмов декодирования для кода из Subframe2

Тип операции	Количество операций										
	Семейство «Min - sum»			Семейство «UMP BP»			Алгоритмы с варьируемым порогом			Кусочная аппроксимация	
	Стандарт	Normalized	Offset	Стандарт	Normalized	Offset	Стандарт	Normalized	Offset	A	B
Сложение	37024	37024	1842	37024	37024	1842	37024	37024	41842	75730	75730
Умножение	33888	38706	3888	0	4818	0	0	4818	0	145188	145188
Сравнение	64159	64159	8977	39907	39907	4725	49543	49543	54361	146389	291577
Взятие модуля числа	33888	33888	3888	33888	33888	3888	33888	33888	33888	38706	38706
Сложение по модулю 2	4218	4218	218	44124	44124	4124	44124	44124	44124	4218	4218

2.5 Повышение вычислительной эффективности декодирования

Алгоритмы декодирования кодов с малой плотностью проверок на четность могут быть реализованы таким образом, что их вычислительная сложность по сравнению с представленной в таблице 2.12 будет значительно ниже. В [74] отмечено, что минимальные значения, вычисляемые при формировании сообщения $L(r_{m,l})$, могут быть определены предварительно сразу для всех сообщений $L(r_{m,l})$, направляемых к символьным узлам l в рамках одной проверки m. Для этого достаточно определить два минимальных абсолютных значения среди сообщений $L(q_{m,l})$, пришедших от символьных узлов l в смежный проверочный узел m.

2.5.1 Повышение вычислительной эффективности алгоритма «Min-sum»

На рисунке 2.7 изображен тот же фрагмент графа Таннера, что и на рисунке 1.9, за исключением того, что теперь в окружностях представлены значения $L(q_{m,l})$, адресуемые символьными узлами l проверочному узлу m.

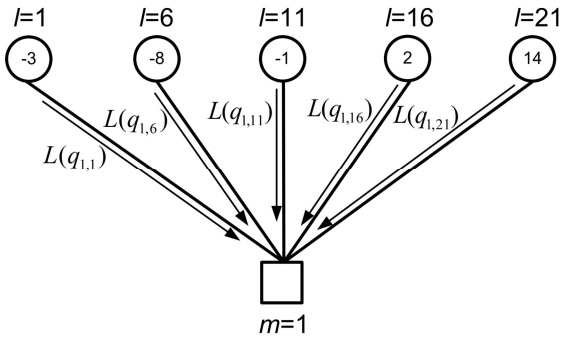

Рисунок 2.7 – Адресация сообщений символьными узлами

Поправка $L(r_{1,1})$ для первого символьного узла рассчитывается по формуле (2.81). При расчете этой поправки используются сообщения локализованной группы $\zeta(1)\setminus 1$ (сплошные окружности на рисунке 2.8).

$$\begin{aligned} L(r_{1,1}) = &\left(sign(L(q_{1,6}))\cdot sign(L(q_{1,11}))\cdot sign(L(q_{1,16}))\cdot sign(L(q_{1,21}))\right)\cdot \ldots \\ \ldots &\cdot \min\left(\left|L(q_{1,6})\right|,\left|L(q_{1,11})\right|,\left|L(q_{1,16})\right|,\left|L(q_{1,21})\right|\right) \end{aligned} \qquad (2.81)$$

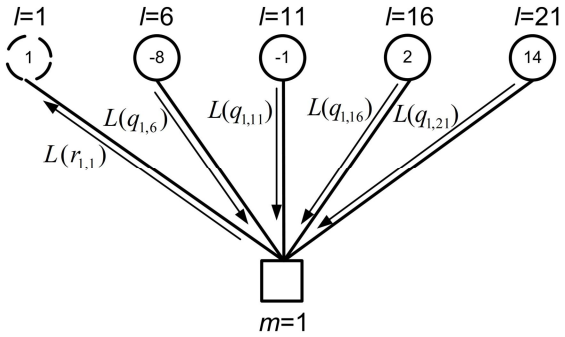

Рисунок 2.8 – Формирование поправки первому символьному узлу

Поправка $L(r_{1,6})$ для шестого символьного узла рассчитывается по (2.82). При расчете этой поправки используются сообщения локализованной группы $\zeta(1) \setminus 6$ (сплошные окружности на рисунке 2.9).

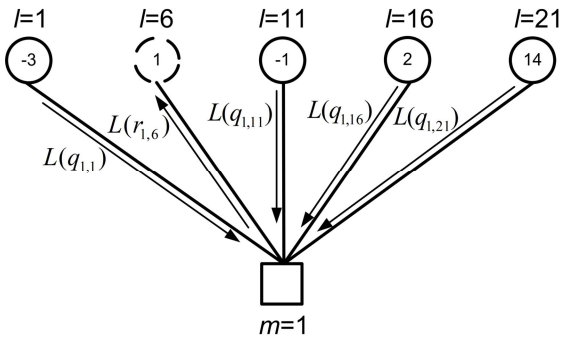

Рисунок 2.9 – Формирование поправки шестому символьному узлу

$$L(r_{1,6}) = \left(sign(L(q_{1,1})) \cdot sign(L(q_{1,11})) \cdot sign(L(q_{1,16})) \cdot sign(L(q_{1,21}))\right) \cdot \ldots$$
$$\ldots \cdot \min\left(|L(q_{1,1})|, |L(q_{1,11})|, |L(q_{1,16})|, |L(q_{1,21})|\right). \quad (2.82)$$

После вычисления всех поправок фрагмент графа примет вид на рисунке 2.10.

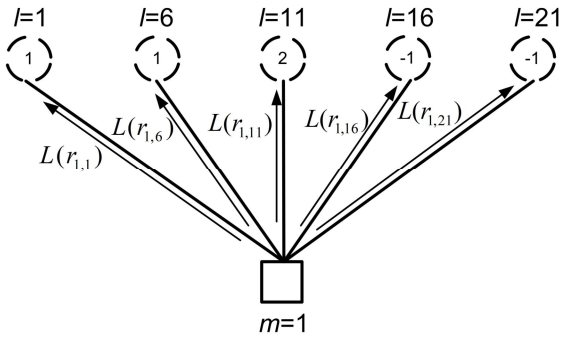

Рисунок 2.10 – После вычисления всех поправок

Можно заметить общее черты приведенных выше формул расчета поправок (2.81) и (2.82). В левых частях этих формул, помимо прочего, происходит перемножение знаков сообщений $L(q_{1,11}), L(q_{1,16}), L(q_{1,21})$, а в правых частях сравнение абсолютных значений сообщений $L(q_{1,11}), L(q_{1,16}), L(q_{1,21})$. Это приводит к избыточности алгоритма, ведь частично формула (2.82) уже была рассчитана в (2.81). Более того, поправки $L(r_{1,1}), L(r_{1,6}), L(r_{1,16}), L(r_{1,21})$ на рисунке 2.10 имеют одно абсолютное значение, равное абсолютному значению сообщения $L(q_{1,11})$ на рисунке 2.7. Это связано с тем, что сообщение $L(q_{1,11})$ имеет минимальное абсолютное значение во всех локализованных группах кроме группы $\zeta(1)\setminus 11$. Относительно группы $\zeta(1)\setminus 11$ формируется сообщение $L(r_{1,11})$, абсолютное значение которого равно абсолютному значению $L(q_{1,16})$, которое является минимальным в рамках этой группы.

На основании вышеописанного, при формировании поправок от проверочного узла *m* существуют всего два типа сообщений: с минимальным абсолютным значением сообщения от символьных узлов и «со вторым с конца». В данной работе такие типы сообщений предлагается рассчитывать по формулам (2.83) и (2.84):

$$\underset{l'\in\xi(m)/l_{min1}}{L(r_{m,l'})} = min1_m \cdot Product_m \cdot Sign_{m,l'}. \tag{2.83}$$

$$L(r_{m,l\min 1}) = min2_m \cdot Product_m \cdot Sign_{m,l\min 1}. \tag{2.84}$$

В формулах (2.83) и (2.84):

$min1_m, min2_m$ - два наименьших абсолютных значения сообщений от символьных узлов *l*, пришедших в проверочный узел *m*. Причем $min1_m < min2_m$.

$Product_m$ - произведение знаков сообщений, пришедших от всех символьных узлов *l* в проверочный узел *m*.

$Sign_{m,l}$ - знак сообщения, пришедшего от *l*-ого символьного узла в проверочный узел *m*.

$l_{\min 1}$ - номер символьного узла, от которого пришло сообщение с минимальным абсолютным значением.

Алгоритм расчета приведенных выше переменных для формул (2.83) и (2.84) показан на рисунке 2.11. На вход алгоритма подаются следующие величины:

- значения сообщений $L(q_{m,l})$, адресованные проверочному узлу m от смежных символьных узлов и их количество k_m.
- знаки пришедших сообщений $Sign_{m,l}$ и их общее произведение $\mathrm{Product_m}$ по умолчанию устанавливаются положительными («1»);
- устанавливается вспомогательный счетчик i.

Алгоритм устанавливает значением $\min 1_m$ абсолютное значение сообщения $L(q_{m,l})$ от первого входящего в проверку на четность символьного узла. Если сообщение имеет отрицательное значение, соответствующий знак сообщения в массиве $Sign_{m,l}$ и произведение $\mathrm{Product_m}$ принимают значение «-1». Абсолютное значение следующего сообщения $L(q_{m,l})$, пришедшего в узел m, устанавливается как $\min 2_m$, и если текущее $L(q_{m,l})$ меньше нуля, соответствующий знак этого сообщения $Sign_{m,l}$ устанавливается в «-1», а произведение $\mathrm{Product_m}$ умножается на -1. В случае если $\min 2_m$ оказалось больше $\min 1_m$, они меняются местами. Далее обрабатываются третье и последующие сообщения от символьных узлов графа Таннера поочередным сравнением абсолютных значений со значениями $\min 1_m$, $\min 2_m$ и формированием знаков $Sign_{m,l}$ и $\mathrm{Product_m}$.

В соответствии с описанным выше упрощением расчета сообщений от проверочных узлов графа Таннера формулы расчета количества операций умножения (2.2), сравнения (2.3) и взятия модуля числа (2.4) при формировании сообщений от проверочных узлов (шаг 2 алгоритма «Min-sum») претерпят изменения. Новые формулы получены для наибольшего пути вычислений по алгоритму на рисунке 2.11.

Количество операций умножения вместо (2.2) оценивается по формуле:

$$N_{\times} = \sum_{i=1}^{m} (2 \cdot k_i + 1). \qquad (2.85)$$

Здесь учитываются операции умножения, которые использует алгоритм на рисунке 2.11 и операции умножения в (2.83) и (2.84), причем операция $min1_m \cdot Product_m$ для формулы (2.83) подсчитана заранее с целью уменьшения количества операций умножения при многократном обращении к формуле. Таким образом, для расчета по (2.83) потребуется всего одно умножение. Для (2.84) такое упрощение не целесообразно, так как расчет сообщений по (2.84) используется единожды в рамках одной проверки на четность m.

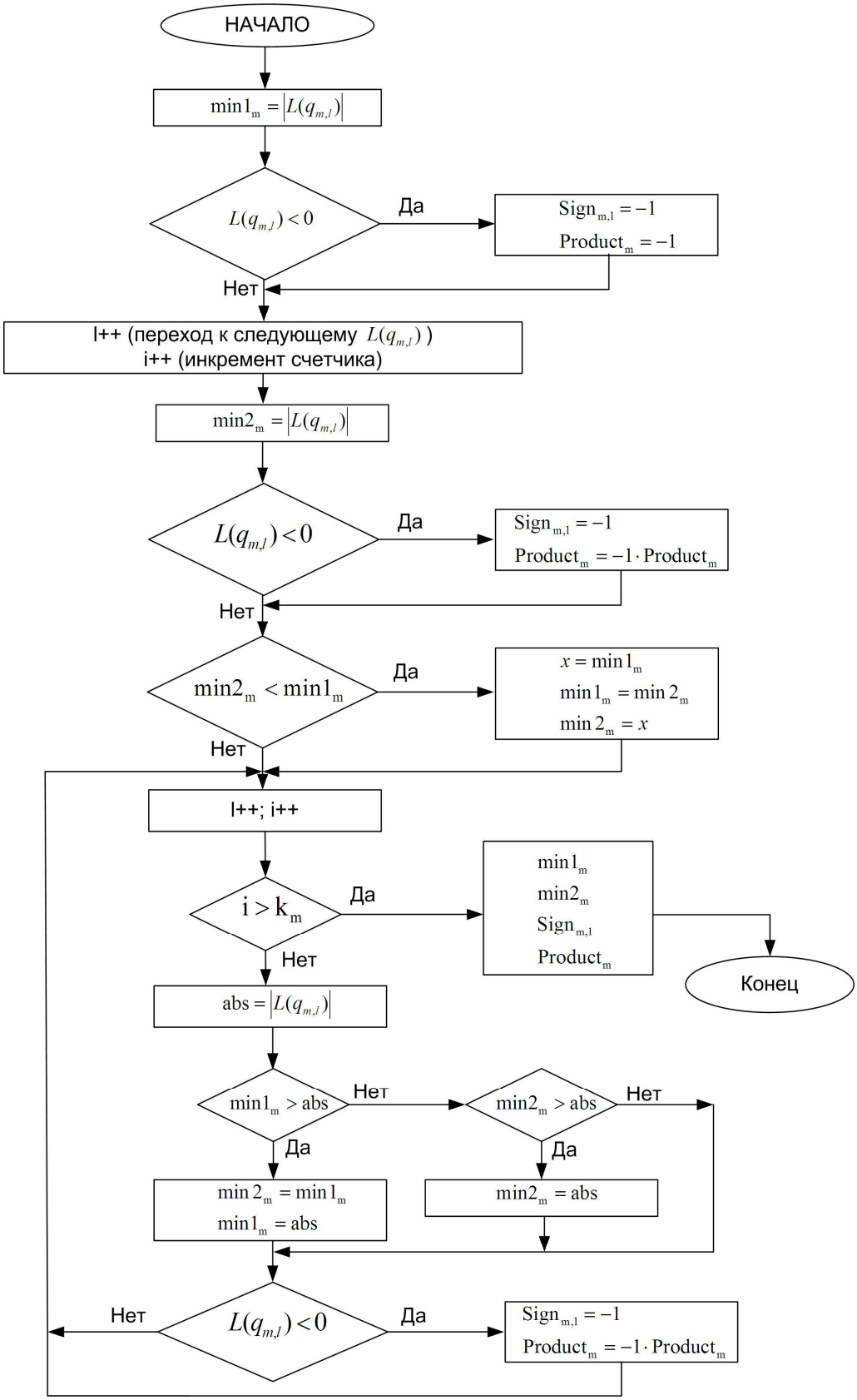

Рисунок 2.11 – Алгоритм поиска переменных для расчета поправок

Количество операций сравнения вместо (2.3) оценивается по формуле:

$$N_{/} = \sum_{i=1}^{m} 3 \cdot (k_i - 1). \quad (2.86)$$

Количество операций взятия модуля числа вместо (2.4) оценивается по формуле:

$$N_{|a|} = \sum_{i=1}^{m} \mathrm{k_i}\,. \quad (2.87)$$

Формулы (2.86) и (2.87) учитывают операции сравнения и взятия модуля числа, которые использует алгоритм на рисунке 2.11.

Итоговые формулы для расчета количества операций.

Общее число операций сложения на одну итерацию декодирования не претерпело изменений по сравнению с классическим алгоритмом «Min-sum» и выражается формулой:

$$N_{+} = \sum_{i=1}^{l} j_i^2\,. \quad (2.88)$$

Общее число операций умножения на одну итерацию декодирования выражается формулой:

$$N_{\times} = \sum_{i=1}^{m} (2 \cdot k_i + 1)\,. \quad (2.89)$$

Общее число операций сравнения на одну итерацию декодирования выражается формулой:

$$N_{/} = 1 + l + \sum_{i=1}^{m} 3 \cdot (k_i - 1). \quad (2.90)$$

Общее число операций взятия модуля числа на одну итерацию декодирования выражается формулой:

$$N_{|a|} = \sum_{i=1}^{m} \mathrm{k}_{\mathrm{i}} \,. \tag{2.91}$$

Общее число операций сложения по модулю 2 на одну итерацию декодирования не претерпело изменений по сравнению с классическим «Min-sum» и выражается формулой:

$$N_{\oplus} = \sum_{i=1}^{m} (k_i - 1) \,. \tag{2.92}$$

Сравнение вычислительной сложности классического и модернизированного алгоритмов «Min-sum» (MS) приведено в таблице 2.13. Выигрыш по скорости декодирования составил в 3 раза.

Таблица 2.13 – Сравнение вычислительной сложности

Операция	Количество на итерацию декодирования	
	Классический MS	Модернизированный MS
Сложение	37024	37024
Умножение	33888	10236
Сравнение	64159	13855
Взятие модуля	33888	4818
Сложение по модулю 2	4218	4218

2.5.2 Повышение вычислительной эффективности алгоритма «UMP BP»

По аналогии с повышением вычислительной эффективности алгоритма «Min-sum», алгоритм «UMP BP» так же может быть модифицирован в части формирования сообщений от проверочных узлов графа Таннера (шаг 2) поиском двух минимальных абсолютных значений в рамках рассматриваемой проверки на четность. Однако, в отличие от алгоритма «Min-sum», алгоритм «UMP BP» формирует знаки сообщений от проверочных узлов через операции сложения по модулю 2, а не через операции умножения. С учетом этого тезиса приведенная в данном разделе модификация коснется формул (1.27) и (1.28), а шаг 2 модифицированного алгоритма «UMP BP» предстанет в следующем виде:

Шаг 2. Расчет сообщений от проверочных узлов.

Формирование массива $x_{m,l}$, как и в классическом алгоритме «UMP BP», осуществляется по формуле (1.26):

$$x_{ml} = \begin{Bmatrix} x_l^c, & если\ L(q_{m,l}) > 0 \\ x_l^c \oplus 1, & если\ L(q_{m,l}) < 0 \end{Bmatrix}.$$

Формула расчета константы $\sigma_{m,l}$ (1.27) имеет избыточный характер, так как для каждого значения $\sigma_{m,l}$ необходимо считать сумму $\sum_{l' \in \xi(m) \setminus l} x_{m,l'} \ [\mathrm{mod}\, 2]$, состоящую почти из одних и тех же слагаемых в рамках одной проверки m. В данной работе предложено вместо (1.27) использовать:

$$\sigma_{m,l} = x_l^c \oplus sumXOR_m \oplus x_{m,l}, \tag{2.93}$$

где $sumXOR_m = \sum_{l \in \xi(m)} x_{m,l} \ [\mathrm{mod}\, 2]$. Использование (2.93) вместо (1.27) позволяет рассчитать единожды сумму по модулю 2 $sumXOR_m$ значений $x_{m,l}$ в рамках одной проверки на четность m, и далее использовать её для формирования $\sigma_{m,l}$ для всех l в рамках проверки m.

Формирование абсолютных значений сообщений $L(r_{m,l})$ осуществляется по следующим формулам:

$$\underset{l' \in \xi(m)/l_{min1}}{L(r_{m,l'})} = min1_m. \tag{2.94}$$

$$L(r_{m,l\,\min 1}) = min2_m. \tag{2.95}$$

Использование (2.94) и (2.95), как и в случае (2.83) и (2.84) для модифицированного «Min-sum», обусловлено тем, что при формировании поправок от проверочного узла m существуют всего два типа сообщений: с минимальным абсолютным значением сообщения от символьных узлов и «со вторым с конца». Алгоритм поиска переменных, входящих в (2.94) и (2.95) аналогичен представленному на рисунке 2.11. Знак сформированного сообщения $L(r_{m,l})$ определяется по формуле:

$$L(r_{m,l}) = \begin{Bmatrix} L(r_{m,l}), & если\ \sigma_{m,l} = 0 \\ -L(r_{m,l}), & если\ \sigma_{m,l} = 1 \end{Bmatrix}. \tag{2.96}$$

Для расчета (1.26) по-прежнему требуются операции сравнения и сложения по модулю 2, число которых оценивается формулами (2.26) и (2.27), соответственно. Для расчета массива $\sigma_{m,l}$ по (2.93) потребуются операции сложения по модулю 2, число которых оценивается формулой:

$$N_{\oplus} = \sum_{i=1}^{m} (3 \cdot k_i - 1). \tag{2.97}$$

Для расчета переменных в (2.94) и (2.95) по алгоритму на рисунке 2.11 для всех проверок на четность потребуются операции сравнения и взятия модуля числа, число которых оценивается по формулам:

$$N_{/} = \sum_{i=1}^{m} (2 \cdot k_i - 3). \tag{2.98}$$

$$N_{|a|} = \sum_{i=1}^{m} \mathrm{k_i} \ . \tag{2.99}$$

Для определения знака поправки по (2.96) используются операции сравнения, число которых оценивается формулой:

$$N_{/} = \sum_{i=1}^{m} k_i \ . \tag{2.100}$$

Итоговые формулы для расчета количества операций.

Общее число операций сложения на одну итерацию декодирования выражается формулой:

$$N_{+} = \sum_{i=1}^{l} j_i^2 \ . \tag{2.101}$$

Общее число операций сравнения на одну итерацию декодирования выражается формулой:

$$N_{/} = 1 + l + \sum_{i=1}^{m} (4 \cdot k_i - 3). \quad (2.102)$$

Общее число операций взятия модуля числа на одну итерацию декодирования выражается формулой:

$$N_{|a|} = \sum_{i=1}^{m} \mathrm{k_i} \, . \quad (2.103)$$

Общее число операций сложения по модулю 2 на одну итерацию декодирования выражается формулой:

$$N_{\oplus} = l + \sum_{i=1}^{m} (5 \cdot k_i - 2). \quad (2.104)$$

Сравнение вычислительной сложности классического и модернизированного алгоритма «UMP BP» приведено в таблице 2.14.

Таблица 2.14 – Сравнение вычислительной сложности

Операция	Количество на итерацию декодирования	
	Классический «UMP BP»	Модернизированный «UMP BP»
Сложение	37024	37024
Сравнение	39907	18673
Взятие модуля	33888	4818
Сложение по модулю 2	44124	24090

2.5.3 Сравнительный анализ сложности модифицированных алгоритмов

В таблице 2.15 приведено количество операций различного типа, используемых модифицированными алгоритмами «Min-sum» (MS) и «UMP BP» за одну итерацию декодирования.

Таблица 2.15 – Сравнение сложности модифицированных алгоритмов

Операция	Количество на итерацию декодирования	
	Модифицированный MS	Модифицированный «UMP BP»
Сложение	37024	37024
Умножение	10236	0
Сравнение	13855	18673
Взятие модуля	4818	4818
Сложение по модулю 2	4218	24090

Модифицированные алгоритмы используют операции сложения только на шагах 1 и 3. Их количество идентично.

Модифицированный алгоритм «Min-sum» формирует знаки сообщений $L(r_{m,l})$ преимущественно с помощью операций умножения, в то время как модифицированный алгоритм «UMP BP» делает это преимущественно с использованием операций сложения по модулю 2. По этой причине модифицированный алгоритм «Min-sum» требует больше операций умножения, а модифицированный алгоритм «UMP BP» больше операций сложения по модулю 2.

Модифицированный алгоритм «UMP BP», при прочих равных с модифицированным алгоритмом «Min-sum», использует большее число операций сравнения за счет необходимости расчета формул (1.26) и (2.96).

Следует подчеркнуть, что платой за снижение вычислительной сложности классических алгоритмов декодирования «Min-sum» и «UMP BP» является необходимость хранения дополнительных переменных, участвующих в расчете по формулам (2.83) и (2.84), а также по формулам (2.93-2.95).

2.6 Выводы по главе

1. Полученные в рамках данной главы аналитические соотношения для расчета числа операций различного типа, выполняемых за одну итерацию декодирования для различных алгоритмов декодирования LDPC, позволяют сравнивать между собой алгоритмы по критерию сложности.

2. Предложенные модификации алгоритмов декодирования в части расчета поправок к «мягким» априорным решениям демодулятора позволяют повысить скорость работы декодера при незначительном увеличении требований к памяти для хранения внутренних переменных.

ГЛАВА 3. Исследование характеристик декодирования LDPC кодов на имитационной модели

В данной главе на имитационной модели получены и проанализированы статистические характеристики декодирования кодов с малой плотностью проверок на четность на примере одного из двух кодов, используемых для кодирования информации, которую несет сигнал L1C.

Основные результаты, полученные в рамках данной главы, опубликованы автором в [9-13 и 21].

3.1 Планирование экспериментов с имитационными моделями

Измеряя какой-либо параметр, всегда встает вопрос о точности измерения. В данной работе следует говорить о том, насколько точно получены в результате имитационного моделирования те или иные параметры, характеризующие работу декодера. На основе проведенного объема выборки делается вывод о точности и достоверности измеренного параметра, а на основании требуемого значения точности и достоверности можно сделать вывод о необходимом объеме выборки.

Точность полученного значения вероятности битовой ошибки p зависит от объема выборки N следующим образом [36]:

$$\varepsilon_{Д} = t_{\varphi}\sqrt{\frac{p(1-p)}{N}}, \tag{3.1}$$

где t_{φ} - квантиль гауссовского закона распределения вероятностей. Это значение берется в зависимости от доверительной вероятности Q из таблицы 3.1.

Таблица 3.1 – Взаимосвязь значений Q и квантиля закона распределения.

Доверительная вероятность, Q	0.68	0.9	0.95	0.99	0.997
Квантиль гауссовского закона распределения вероятностей, t_{φ}	0.995	1.645	1.960	2.576	2.968

Количество реализаций, требуемое для оценки вероятности p с точностью $\varepsilon_{Д}$ и достоверностью Q, рассчитывается как

$$N = \frac{t_{\varphi}^{2}\,p(1-p)}{\varepsilon_{Д}^{2}}. \tag{3.2}$$

Из (3.2) следует, что оценка малых вероятностей ошибки на выходе декодера с высокой точностью требует проведения большого количества экспериментов. Так, например, при оценке вероятности $p=10^{-7}$ с точностью 10% и доверительной вероятностью Q=0.997 требуемый объем выборки будет равен $8.8\cdot10^{9}$.

3.2 Представление низкоплотностной матрицы проверки на четность

Низкоплотностная матрица проверки на четность представляет собой двумерный массив. Чем больше длина кодового слова, тем больше матрица проверки на четность. Этот факт делает затруднительным процедуру хранения матриц в памяти декодера, особенно для больших длин кодовых слов. Кроме того, низкоплотностная матрица содержит в основном нулевые элементы, которые не участвуют в декодировании. Как следствие, их хранение нерентабельно. Как правило, низкоплотностную матрицу проверки на четность представляют в компактном виде. К примеру, в пакете MATLAB с этой целью используется функция *sparse*. Применение этой функции позволяет представить любую матрицу в виде координат её ненулевых элементов и значений, находящихся на соответствующих позициях. Такой подход позволяет существенно сократить требуемые ресурсы памяти для хранения матрицы проверки на четность. В данной работе предложена альтернативная техника представления матрицы проверки на четность, позволяющая еще больше сократить требуемые ресурсы памяти для хранения матрицы.

Обратимся к схематично изображенному на рисунке 3.1 примеру низкоплотностной матрицы проверки на четность размерностью 600 на 1200, содержащей 4818 ненулевых элементов. Здесь поля с цифрами соответствуют ненулевым элементам, а без цифр – нулевым.

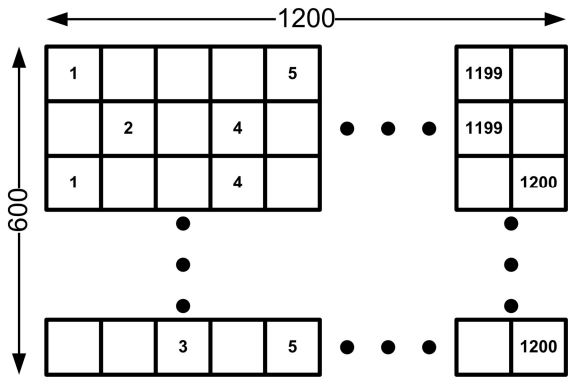

Рисунок 3.1 - Схематичное изображение матрицы проверки на четность

Идея состоит в представлении матрицы проверки на четность в виде двух целочисленных одномерных массивов: массива *position* координат ненулевых элементов матрицы по горизонтали и массива *weight row* весов строк матрицы. Под весом строки

понимается количество единиц в строке. При этом массив *position* будет иметь размер, равный числу ненулевых элементов матрицы, а массив *weight row* будет иметь размер, равный количеству строк матрицы.

К матрице проверки на четность, загруженной в MATLAB, применяется функция *sparse*, выдающая на выходе координаты и значения ненулевых элементов матрицы в формате [(X, Y) Z]. Здесь X – координата ненулевого элемента матрицы по горизонтали. Y - координата ненулевого элемента матрицы по вертикале. Z – Значение ненулевого элемента. Используя полученный выход функции *sparse*, создается массив *position*, состоящий из первых координат (X), а также массив *weight row*, содержащий количество единиц в каждой строке матрицы проверки на четность.

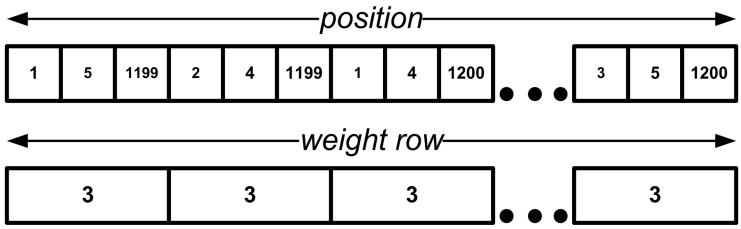

Рисунок 3.2 - Представление матрицы двумя массивами

Итеративный обмен между символьными и проверочными узлами графа Таннера происходит по ребрам графа, которые расставлены в соответствии с ненулевыми элементами матрицы проверки на четность. Следовательно, для хранения сообщений от символьных $L(q_{m,l})$ и от проверочных $L(r_{m,l})$ узлов достаточно двух массивов, размеры которых равны размеру массива *position*, то есть числу ненулевых элементов матрицы проверки на четность.

Описанная структура матрицы и внутренних переменных декодера используется далее при моделировании низкоплотностного декодирования.

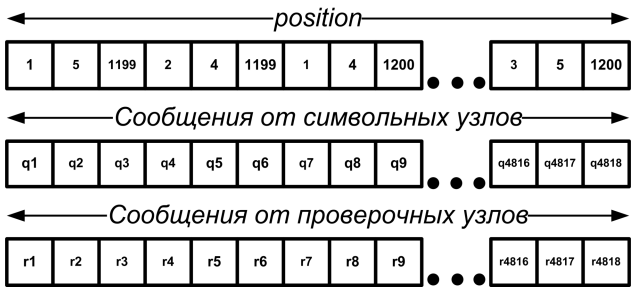

Рисунок 3.3 - Компактное представление символьных и проверочных сообщений графа

3.3 Описание имитационной модели

Целью моделирования являлось получение ряда статистических характеристик декодирования низкоплотностного декодера, работающего по описанным выше алгоритмам. Структурная схема модели односторонней линии цифровой радиосвязи приведена на рисунке 3.4.

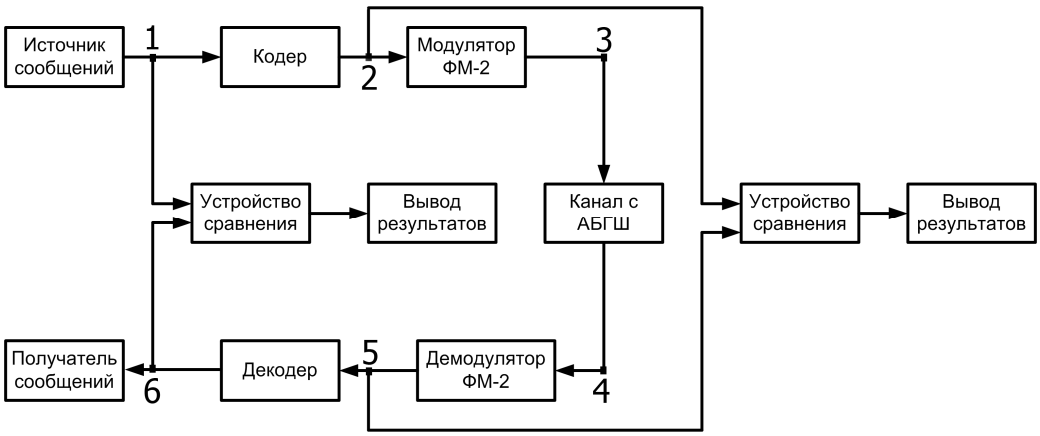

Рисунок 3.4 – Структурная схема цифровой линии радиосвязи

Источник сообщений формирует бинарную последовательность блоками по 600 символов. Размер блока обусловлен размерностью порождающей и проверочной матрицы.

Моделирование работы всех алгоритмов проводилось для LDPC кода, используемого в части Sub2 кадра сигнала L1C. Структура проверочной матрицы моделируемого кода приведена на рисунке 1.3. Пример эпюры сигнала в точке 1 представлен на рисунке 3.5.

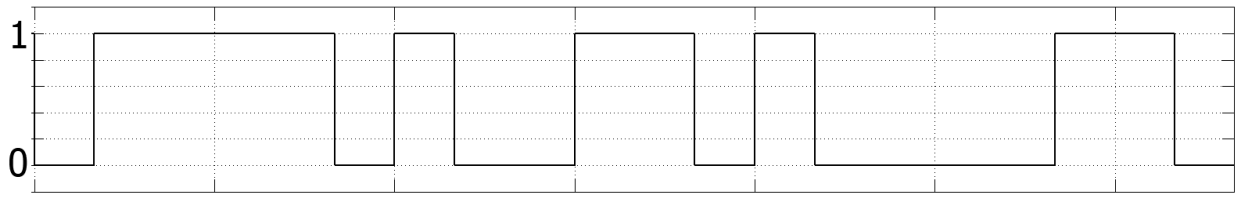

Рисунок 3.5 – Пример эпюры сигнала на выходе источника сообщений

Последовательность поступает на LDPC кодер, осуществляющий перемножение последовательности на порождающую матрицу. Пример эпюры сигнала на выходе кодера (точка 2) представлен на рисунке 3.6.

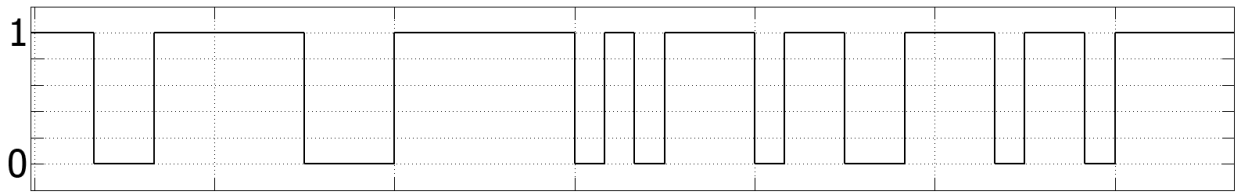

Рисунок 3.6 – Пример эпюры сигнала на выходе LDPC кодера

Закодированная последовательность длиной в 1200 символов поступает на модулятор ФМ-2. При моделировании использовался метод комплексной огибающей, в котором сигнал представляется в виде двух составляющих: синфазной и квадратурной, причем для незашумленного сигнала ФМ-2 квадратурная составляющая равна 0, а синфазная изменяется в диапазоне от 1 до -1. Передаче символа «1» будет соответствовать сигнал с амплитудой по синфазной составляющей -1, а по квадратурной 0. В случае передачи символа «0», сигнал будет иметь амплитуду по синфазной составляющей 1, а по квадратурной 0. Векторное представление сигналов для модуляции ФМ-2 приведено на рисунке 3.7. Пример эпюры сигнала в точке 3 представлен на рисунке 3.8. Здесь пунктирная линия соответствует амплитуде сигнала для квадратурной составляющей (**Q**), а сплошная для синфазной (**I**).

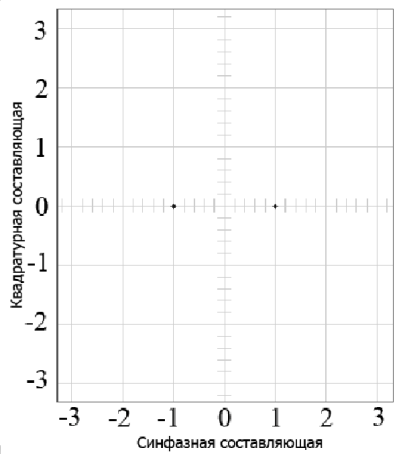

Рисунок 3.7 – Векторное представление сигнала ФМ-2

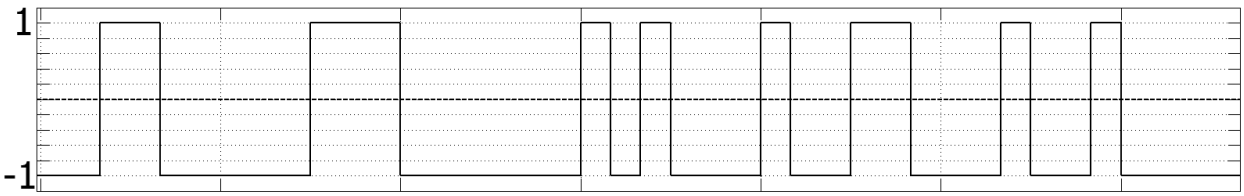

Рисунок 3.8 – Пример эпюры сигнала на выходе модулятора

В канале сигнал подвергается воздействию белого шума, в результате чего сигнал меняет свои амплитуды по синфазной и квадратурной составляющей, соответственно. Векторное представление зашумленного сигнала приведено на рисунке 3.9. Пример эпюры сигнала в точке 4 показан на рисунке 3.10.

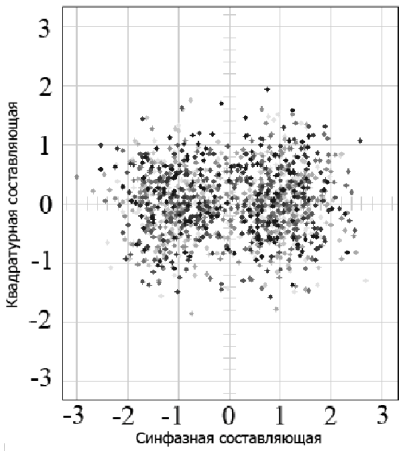

Рисунок 3.9 – Векторное представление зашумленного сигнала ФМ-2

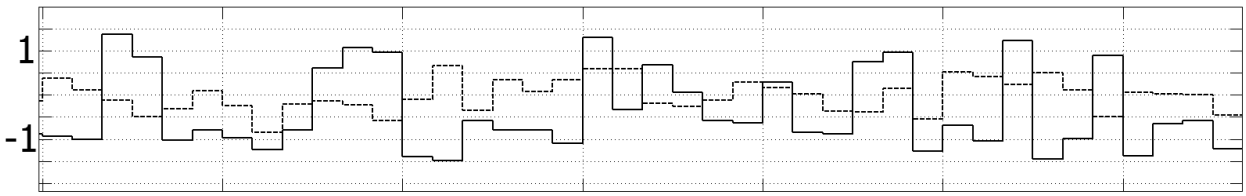

Рисунок 3.10 – Пример эпюры сигнала на входе демодулятора

Сигнал поступает на демодулятор, который может выдавать как «жесткие» решения «0» и «1», так и «мягкие» решения, представляющие собой надежности принятых символов. Пример эпюры сигнала в точке 5 в случае «жестких» решений приведен на рисунке 3.11, а в случае «мягких» решений на рисунке 3.12. Следует подчеркнуть, что эпюры на рисунках 3.11 и 3.12 не коррелированны между собой, а представляют разные фрагменты принимаемого сигнала и призваны качественно показать вид сигнала на входе декодера.

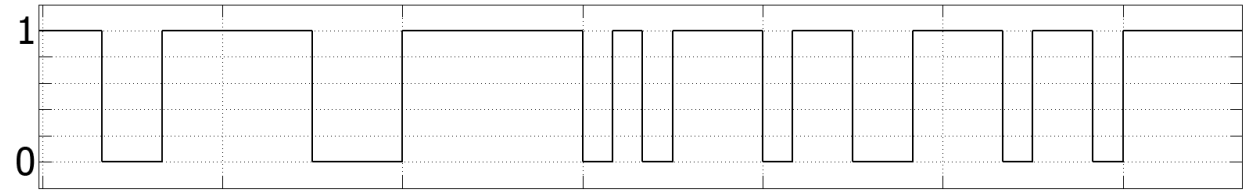

Рисунок 3.11 – Пример эпюры сигнала на входе «жесткого» декодера

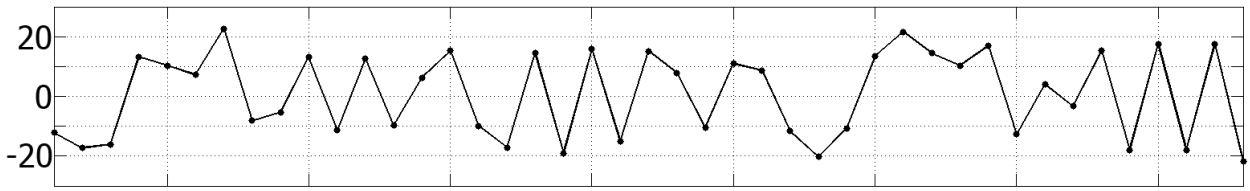

Рисунок 3.12 – Пример эпюры сигнала на входе «мягкого» декодера

Сигнал в точке 6, в случае если ошибок не произошло или они были исправлены, должен соответствовать сигналу в точке 1.

3.4 Результаты имитационного моделирования

Для каждого из алгоритмов декодирования был разработан C++ код, внедряемый в сегмент декодера имитационной модели. Результаты моделирования приведены в виде графиков зависимостей вероятности битовой ошибки (BER) от битового отношения сигнал/шум, среднего числа итераций от битового отношения сигнал/шум и сходимости среднего веса синдрома. Полученные характеристики вероятности ошибок имеют допуск порядка 10% по уровню 10^{-6}, а зависимости среднего числа итераций и сходимости синдрома получены на выборке в 10000 слов.

3.4.1 Подбор весового коэффициента для алгоритма «Min-sum normalized»

Осуществляя декодирование по алгоритму «Min-sum normalized» важно правильно подобрать коэффициент нормировки α. Как отмечалось в п.1.3.4.2, оптимальное значение α лежит в диапазоне $1.25\ldots1.6$. На рисунке 3.13 демонстрируется зависимость вероятности битовой ошибки от битового отношения сигнал/шум для низкоплотностного кода из Subframe2 сигнала L1C при различных коэффициентах нормировки α.

Моделирование показало, что алгоритм «Min-sum normalized», в рамках исследуемого кода, демонстрирует наилучшую исправляющую способность при $\alpha = 1.33$. Энергетический проигрыш при этом не превышает 0.1 дБ по уровню 10^{-5} по сравнению с алгоритмом «Belief propagation». Для сравнения на график помещены кривые при различном значении α.

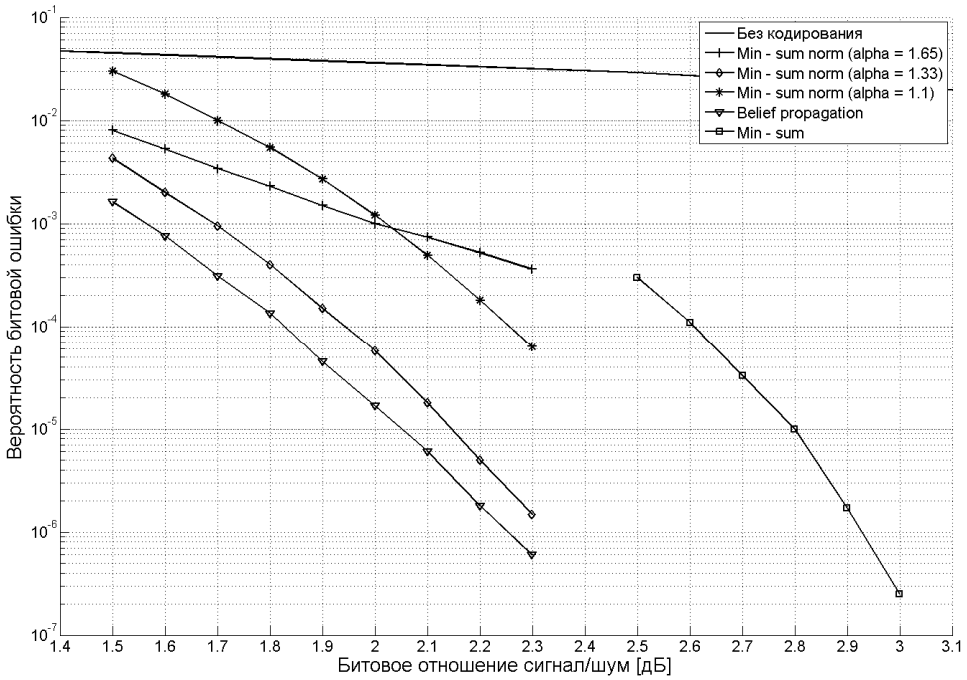

Рисунок 3.13 – BER для исследуемого кода при различных α

Пересечение двух кривых со значениями $\alpha = 1.65$ и $\alpha = 1.1$ обусловлено следующим: на относительно малых отношениях сигнал/шум (1.5 дБ – 2 дБ) слишком сильно влияние надежностей символов, принятых с ошибкой. Действительно, если вернуться к фрагменту графа Таннера на рисунке 1.9 и предположить, что в локализованной группе $\zeta(1)\backslash 1$ есть ошибка, поправка $L(r_{11})$ будет иметь противоположный характер, что еще больше увеличит надежность первого символа, принятого с ошибкой. Большой коэффициент нормировки α «режет» это влияние за счет того, что поправка, сформированная с участием символов, принятых с ошибками, берется с весом $\frac{1}{\alpha}$. С ростом отношения сигнал/шум количество ошибок в локализованных группах сводится к нулю, однако большой коэффициент α по-прежнему «режет» поправки, которые в регионе относительно высоких отношений сигнал/шум (2 дБ – 2.3 дБ) являются уже, в большинстве случаев, «полезными», то есть сформированными на основе локализованных групп, не содержащих ошибок. За счет этого алгоритм с $\alpha = 1.65$ проигрывает алгоритму с $\alpha = 1.1$ на относительно высоких отношениях сигнал/шум.

3.4.2 Подбор корректирующей константы для алгоритма «Min- sum offset»

По аналогии с коэффициентом нормировки алгоритма «Min-sum normalized» в случае декодирования по алгоритму «Min-sum offset» важно правильно выбрать значение константы *c*. Как отмечалось в п. 1.3.4.3, значение константы, как правило, лежит в диапазоне $0 \ldots 0.8$. На

рисунке 3.14 демонстрируется зависимость вероятности битовой ошибки от битового отношения сигнал/шум для исследуемого низкоплотностного кода при различных значениях константы *c*.

Моделирование показало, что алгоритм «Min-sum offset», в рамках исследуемого кода, демонстрирует наилучшую исправляющую способность при *c*=0.6. Энергетический проигрыш от применения данного алгоритма с *c*=0.6 не превышает 0.05 дБ по уровню 10^{-5} по сравнению с алгоритмом «Belief propagation». Для сравнения на график помещены кривые при различном значении *c*.

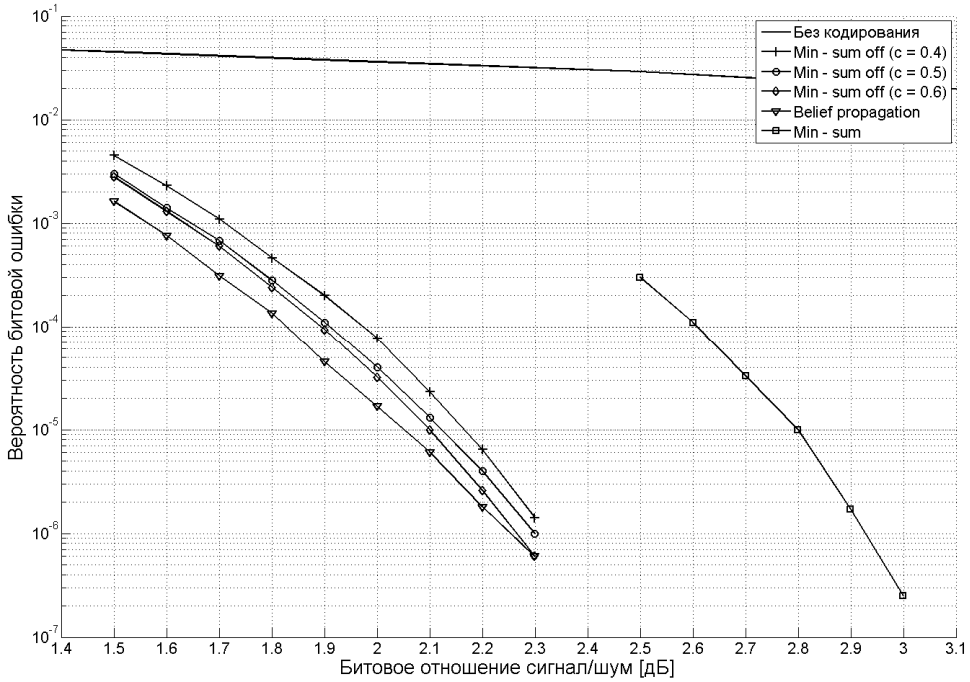

Рисунок 3.14 – BER для исследуемого кода при различных *c*

3.4.3 Влияние порога на декодирование по мажоритарному алгоритму

Введение ненулевого порога при обработке сообщений от символьных узлов графа Таннера (1.34) позволяет повысить исправляющую способность мажоритарного алгоритма декодирования. В данной работе изменять порог от итерации к итерации предлагается по формуле:

$$T_{Iter} = \begin{cases} \dfrac{y}{Iter}, & если\ 1 \le Iter \le x \\ 0, & если\ x < Iter \le Iter_{\max} \end{cases}, \qquad (3.3)$$

где *Iter* – номер текущей итерации.

Значение порога будет зависеть от начального положения (глубина порога *y* на рисунке 1.12) и от количества итераций, на которых этот порог будет отличен от нуля (ширина сходимости порога *x* на рисунке 1.12). На рисунке 3.15 приводится зависимость помехоустойчивости мажоритарного алгоритма с варьируемым порогом от параметров порога (*x*,*y*) для случаев высокого (20,-20), среднего (50,-50) и низкого (80, -80) порогов.

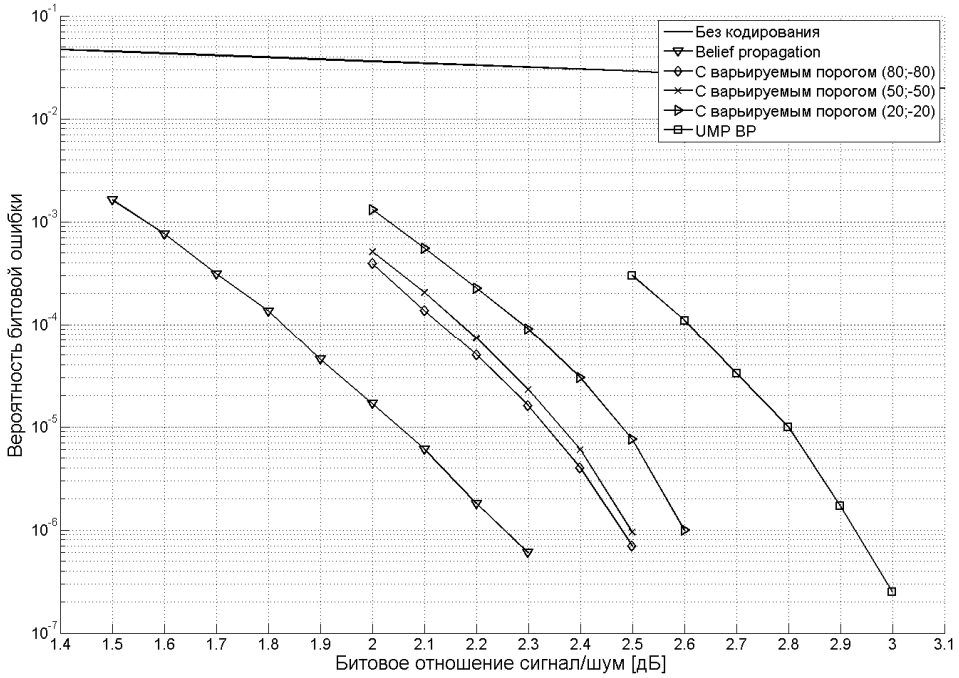

Рисунок 3.15 – BER для исследуемого низкоплотностного кода при различных порогах

Все три варианта демонстрируют лучшие характеристики BER, чем стандартный алгоритм декодирования «UMP BP». Алгоритм с параметрами порога (20,-20) показывает лучшую на ~0.3 дБ характеристику BER по сравнению с классическим алгоритмом «UMP BP» по уровню 10^{-5}. Энергетический выигрыш от применения порогов (50,-50) и (80,-80) составил ~0.44 и ~0.47 дБ, соответственно.

При выставлении низкого порога на начальных итерациях вообще не происходит инвертирования символов, или таких символов единицы. Это положительно сказывается на качестве декодирования, так как не происходит внесения ошибок. Минусом низких значений порога является тот факт, что первые итерации декодер работает «вхолостую», формируя значения поправок, но, не исправляя ошибок. Как следствие, декодер, работающий с низкими значениями порогов, будет использовать больше итераций, чем декодер с порогами выше. Это наглядно показано на рисунке 3.16, где приведено среднее число итераций, используемое декодером при различных порогах для полной достоверности на выборке 10000 слов.

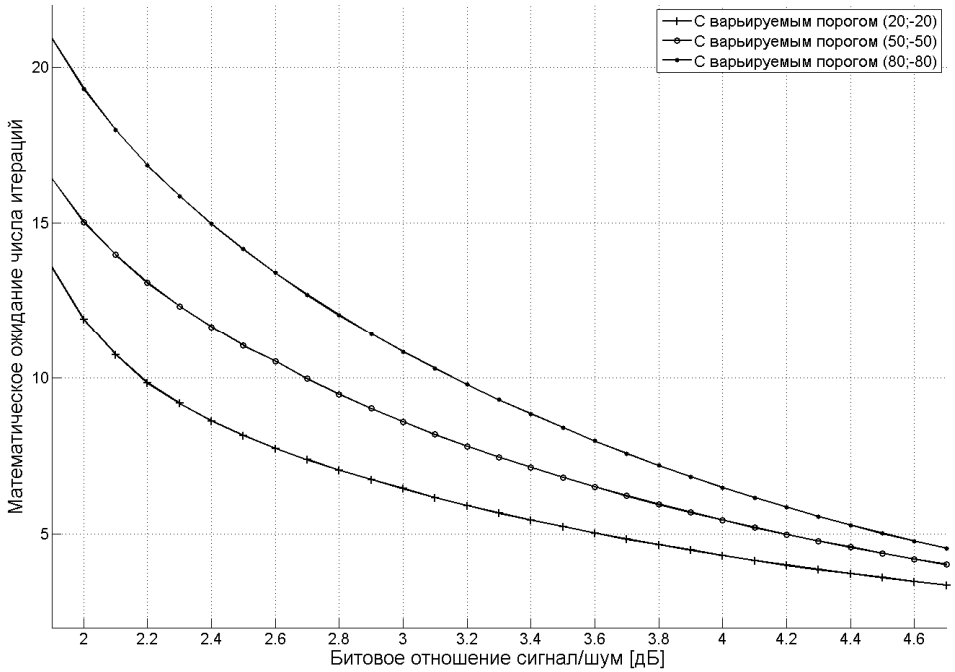

Рисунок 3.16 – Зависимость среднего числа итераций при различных порогах

Сходимость среднего веса синдрома в зависимости от выставленного порога инвертирования для фиксированного отношения сигнал/шум 3 дБ демонстрирует рисунок 3.17. Работа с низким порогом инвертирования инициируют медленное схождение синдрома к нулю. При декодировании с высокими порогами средний вес синдрома сходится быстрее, однако при ухудшении отношения сигнал/шум средний вес синдрома может не сойтись вообще в силу относительно слабой исправляющей способности алгоритма с высоким выставленным порогом.

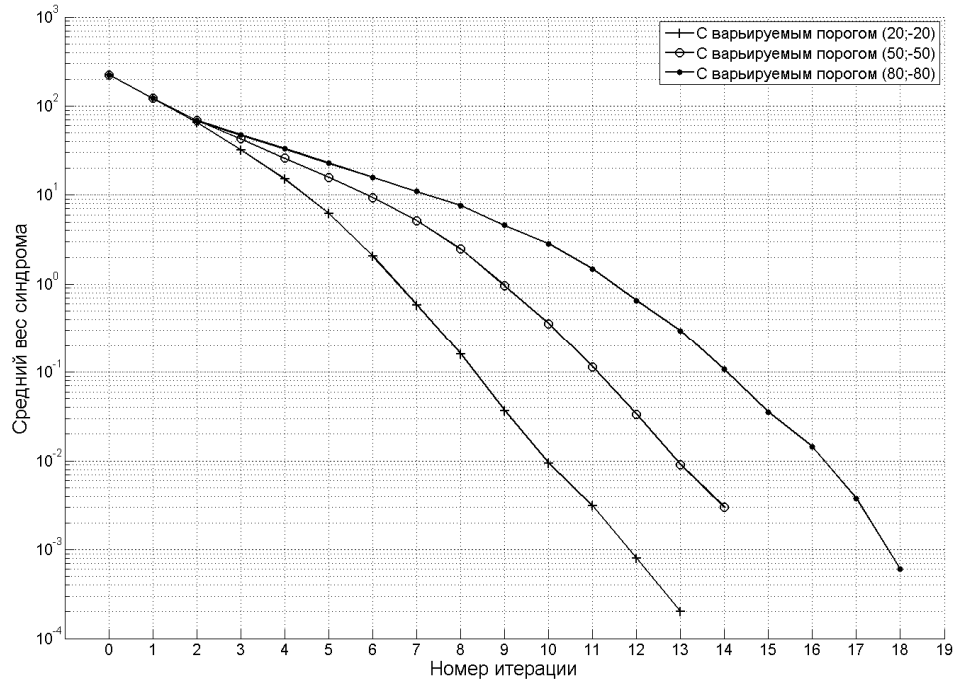

Рисунок 3.17 – Сходимость среднего веса синдрома при различных порогах

3.4.4 Сравнение характеристик декодирования субоптимальных алгоритмов

Для описанных выше алгоритмов декодирования были получены кривые помехоустойчивости BER (рисунок 3.18), зависимости среднего числа использованных декодером итераций от отношения сигнал/шум (рисунок 3.19) и характеристики сходимости среднего веса синдрома от номера итерации (рисунок 3.20 и рисунок 3.21) при битовом отношении сигнал/шум 3 дБ.

Алгоритм «Belief propagation» демонстрирует наилучшие характеристики помехоустойчивости, наименьшее среднее число использованных итераций и наибыстрейшее схождение среднего веса синдрома к нулю.

Версии алгоритмов с нормировкой (normalized) и сдвигом (offset) из семейства «Min-sum» (или, то же самое с точки зрения помехоустойчивости, из семейства «UMP BP») и мажоритарные алгоритмы с варьируемым порогом и нормировкой и сдвигом проигрывают алгоритму «Belief propagation» не более ~0.1 дБ. Причем во всех случаях алгоритмы со сдвигом демонстрируют лучшие на ~0.05 дБ характеристики помехоустойчивости, чем алгоритмы с нормировкой. Более того, в среднем версии алгоритмов со сдвигом требуют меньшее число итераций на декодирование, а их средние веса синдромов сходятся быстрее средних весов алгоритмов с нормировкой.

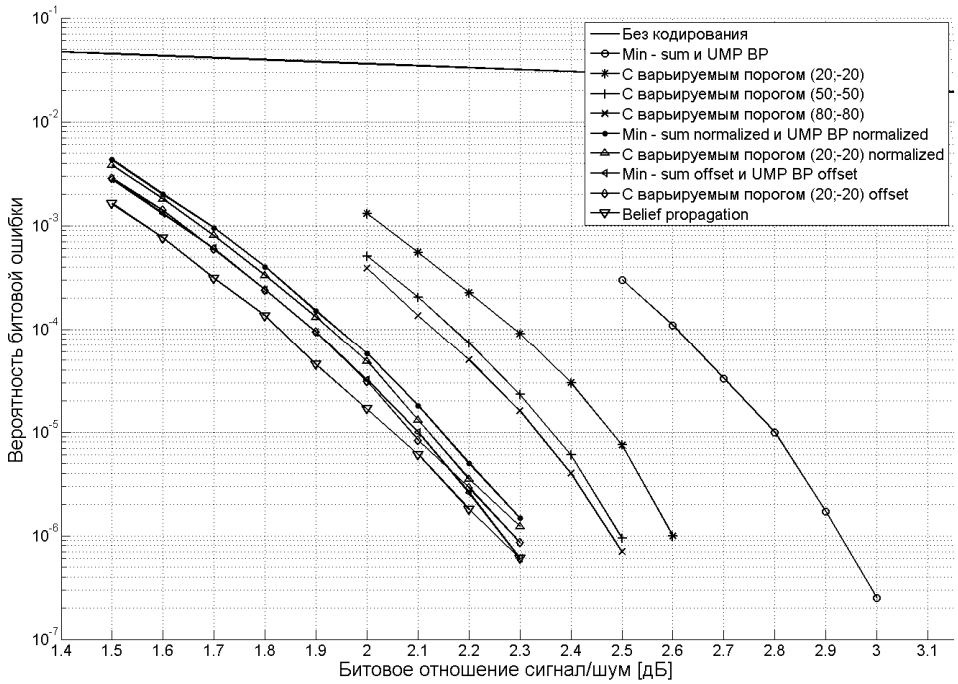

Рисунок 3.18 – BER для различных алгоритмов декодирования

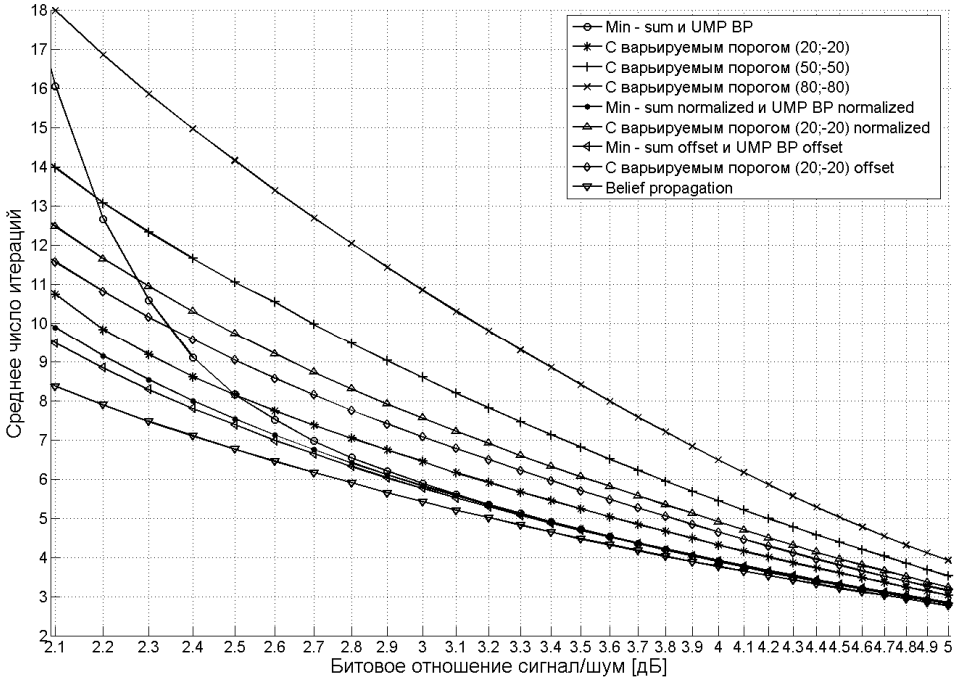

Рисунок 3.19 – Зависимость среднего числа итераций от битового отношения сигнал/шум

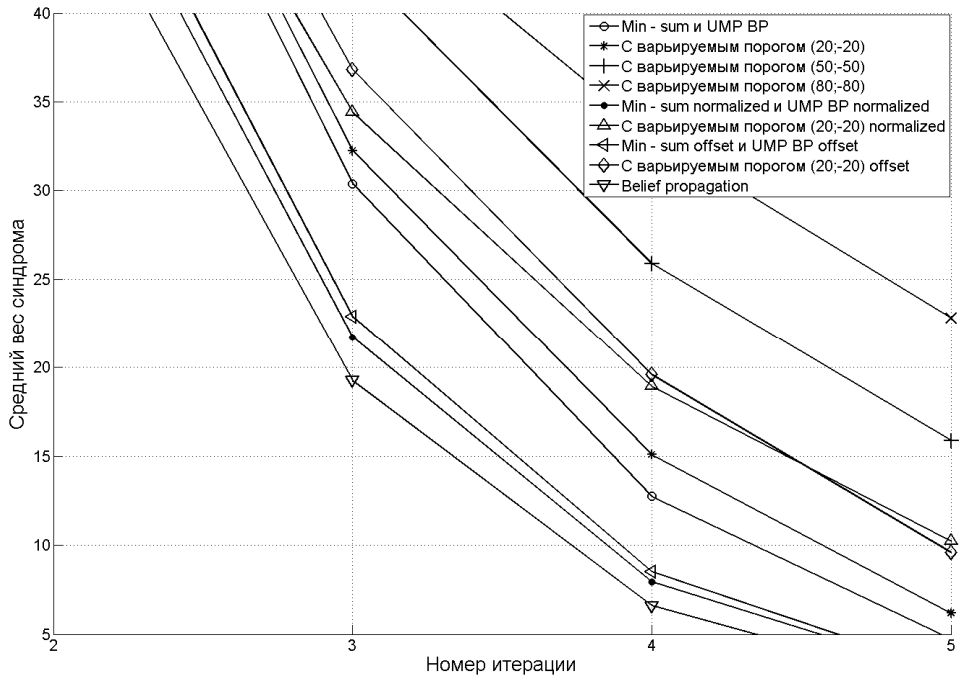

Рисунок 3.20 – Сходимость среднего веса синдрома (начальный участок)

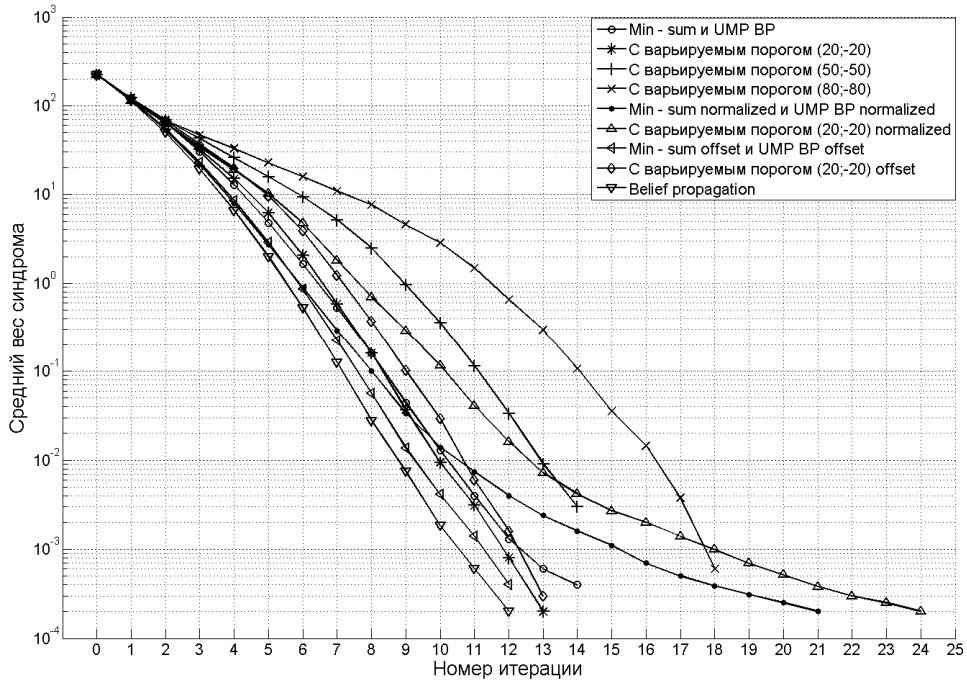

Рисунок 3.21 – Сходимость среднего веса синдрома

Когда ошибок достаточно много имеет место эффект негативного влияния поправок (уже упомянут в п. 3.4.1), сформированных с участием символов локализованной группы, в которую входят ошибочно принятые символы. Алгоритмы с нормировкой рассчитывают такие поправки с весом $(\frac{1}{\alpha})$, в то время как алгоритмы со сдвигом лишь вычитают значение константы *c* из абсолютного значения рассчитанной поправки. В свете этого в условиях обилия ошибок алгоритмы с нормировкой оказываются предпочтительнее. Об этом свидетельствует и тот факт, что средний вес синдрома у алгоритмов с нормировкой на начальных итерациях (рисунок 3.20), когда ошибок достаточно много, сходится быстрее, чем у алгоритмов со сдвигом. Однако с ходом итеративного декодирования количество ошибок уменьшается, что сводит к минимуму эффект, обуславливающий формирование поправок с участием ошибочно принятых символов. В этой ситуации алгоритмы со сдвигом смотрятся выигрышнее за счет того, что значения поправок, сформированные на основании локализованных групп, уже не содержащих ошибок, лишь корректируются константой, а не грубо «режутся» нормирующим коэффициентом. Как следствие, медленное схождение среднего веса синдрома на начальных итерациях у алгоритмов со сдвигом компенсируется его быстрым схождением на завершающей стадии декодирования (рисунок 3.21).

Мажоритарные алгоритмы с варьируемым порогом демонстрируют энергетический выигрыш ~0.3-0.4 дБ относительно декодирования с нулевым порогом (стандартный «UMP BP»), но проигрывают кластеру из алгоритма «Belief propagation» и алгоритмов с нормировкой

и сдвигом порядка ~0.3-0.4 дБ, тем самым занимая промежуточный сегмент. Следует отметить, что в силу своей специфики эти алгоритмы отличаются медленным схождением среднего веса синдрома к нулю и большим средним числом используемых итераций для декодирования по отношению к алгоритмам, не использующим варьирование порога.

3.4.5 Варианты кусочной аппроксимации гиперболических функций

Наряду с алгоритмами, использующими приближение (1.21), существуют другие пути упрощения процесса декодирования, позволяющие избежать аналитического расчета гиперболических функций. Одним из таких путей является использование аппроксимаций функций $\tanh(x)$ и $\operatorname{atanh}(x)$. В данной работе предлагаются различные варианты аппроксимаций гиперболических функций $\tanh(x)$ и $\operatorname{atanh}(x)$ с учетом свойств нечетности данных функций.

Функция $\tanh(x)$, используемая при декодировании по алгоритму «Belief propagation», представлена на рисунке 3.22.

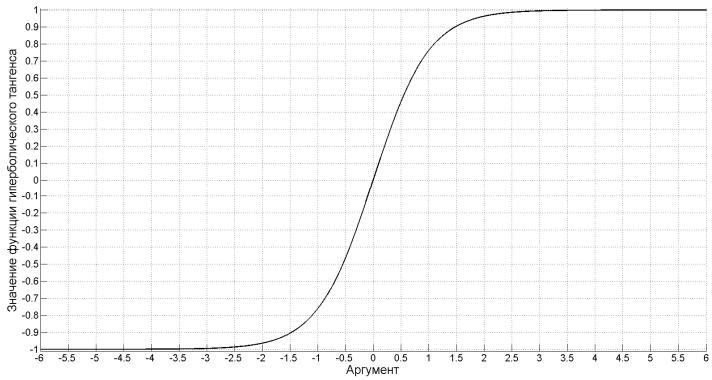

Рисунок 3.22 – Функция гиперболического тангенса tanh(x)

Учитывая факт нечетности данной функции можно аппроксимировать только половину функции, например, в области положительных аргументов. В этом случае, если аргумент отрицательный, необходимо взять модуль аргумента, рассчитать его по положительной части аппроксимированной функции и умножить полученное значение на -1. Такой подход позволяет сократить участок аппроксимируемой функции вдвое, что приводит к упрощению процесса аппроксимации.

В данной работе рассмотрено 4 варианта линейной аппроксимации функции гиперболического тангенса. Аппроксимации изображены на рисунке 3.23.

Общий вид уравнения прямой на каждом из отрезков аппроксимации имеет вид:

$$y(x) = kx + b\,, \tag{3.4}$$

где k и b это некоторые константы.

В таблице 3.2 приведены наборы констант k и b для предложенных вариантов аппроксимации и соответствующие им ошибки аппроксимации, рассчитанные следующим образом:

$$\sigma^2 = \int_0^{\infty} \left(y(x) - \tanh(x)\right)^2 dx\,. \tag{3.5}$$

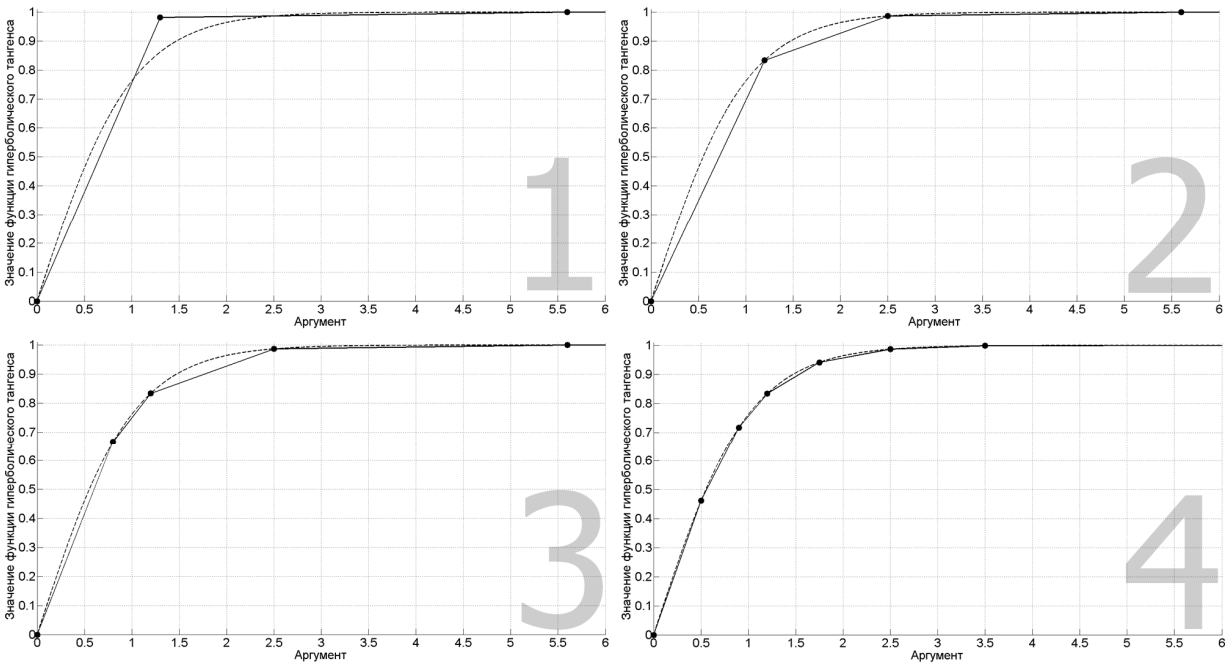

Рисунок 3.23 – Варианты аппроксимации tanh(x)

В зависимости от выбора варианта аппроксимации функции гиперболического тангенса, с учетом идеального расчета функции гиперболического арктангенса, помехоустойчивость изменяется так, как показано на рисунке 3.24.

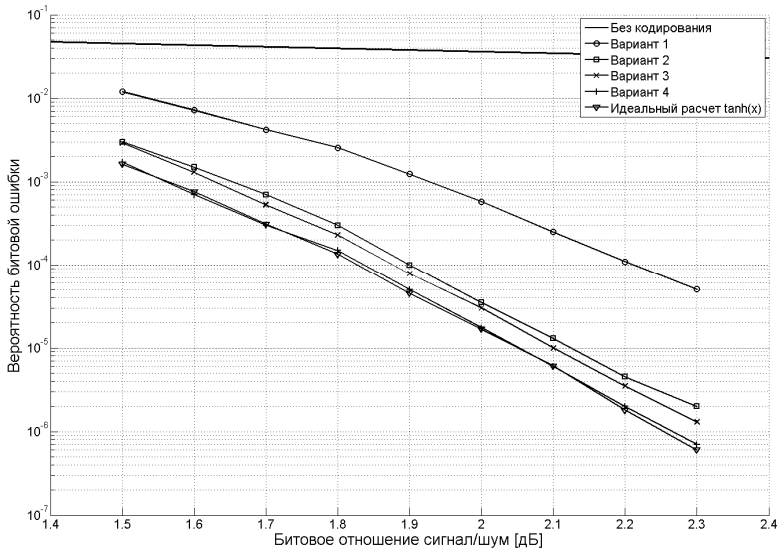

Рисунок 3.24 – BER для различных вариантов аппроксимации tanh(x)

В ходе моделирования выявлена связь между ошибкой аппроксимации σ^2 и помехоустойчивостью. Чем точнее выполнена кусочная аппроксимация, тем лучшие характеристики помехоустойчивости демонстрирует алгоритм. Наилучшие характеристики помехоустойчивости продемонстрировал вариант аппроксимации номер 4 со значением $\sigma^2 = 0.0002$.

Таблица 3.2 – Коэффициенты уравнений аппроксимации tanh(x)

Вариант аппроксимации	Участок	Аргумент x	k	b	σ^2
1	1	[0; 1.3)	0.7548	0	0.01
	2	[1.3; 5.6)	0.0043	0.9756	
	3	[5.6; +∞)	$y(x)=1$		
2	1	[0; 1.2)	0.6947	0	0.008
	2	[1.2; 2.5)	0.1177	0.6925	
	3	[2.5; 5.6)	0.0043	0.9756	
	4	[5.6; +∞)	$y(x)=1$		
3	1	[0; 0.8)	0.8333	0	0.002
	2	[0.8; 1.2)	0.424	0.3248	
	3	[1.2; 2.5)	0.1177	0.6925	
	4	[2.5; 5.6)	0.0043	0.9756	
	5	[5.6; +∞)	$y(x)=1$		
4	1	[0; 0.5)	0.9242	0	0.0002
	2	[0.5; 0.9)	0.6355	0.1444	
	3	[0.9;1.2)	0.3912	0.3642	
	4	[1.2;1.75)	0.1958	0.5986	
	5	[1.75;2.5)	0.0603	0.8358	
	6	[2.5;3.5)	0.0115	0.9577	
	7	[3.5;8)	0.0004	0.9967	
	8	[8; +∞)	$y(x)=1$		

Функция atanh(x), используемая при декодировании по алгоритму «Belief propagation», представлена на рисунке 3.25.

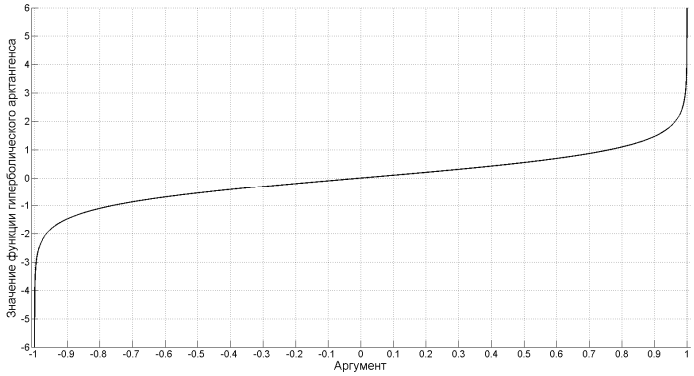

Рисунок 3.25 – Функция гиперболического арктангенса atanh(x)

В данной работе рассмотрено 5 вариантов линейной аппроксимации функции гиперболического арктангенса в области положительных аргументов. Варианты аппроксимаций изображены на рисунке 3.26.

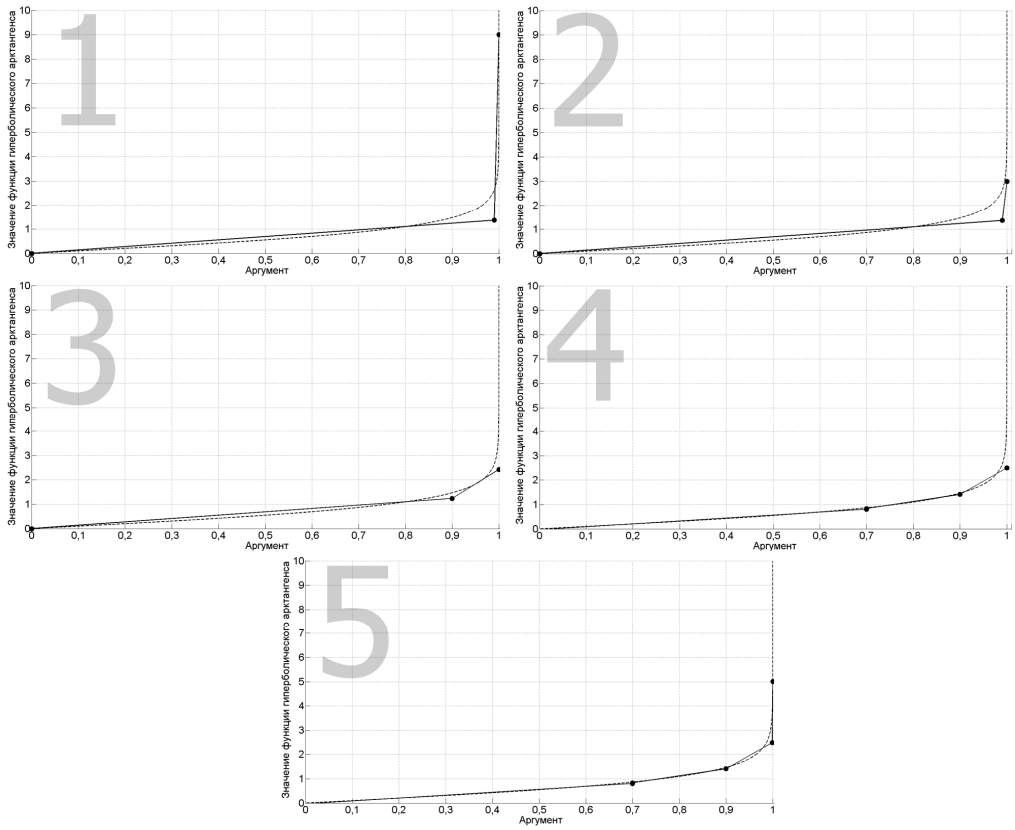

Рисунок 3.26 – Варианты аппроксимации atanh(x)

В таблице 3.3 приведены наборы констант k и b для предложенных вариантов аппроксимации и соответствующие им ошибки аппроксимации. В зависимости от выбора варианта аппроксимации функции гиперболического арктангенса, с учетом идеального расчета функции гиперболического тангенса, помехоустойчивость изменяется так, как показано на рисунке 3.27.

Наилучшую помехоустойчивость обеспечивает аппроксимация вариантом номер 5 ($\sigma^2 = 0.0052$). Здесь, как и в случае аппроксимации функции гиперболического тангенса, имеет место связь между среднеквадратичным значением ошибки аппроксимации σ^2 и помехоустойчивостью.

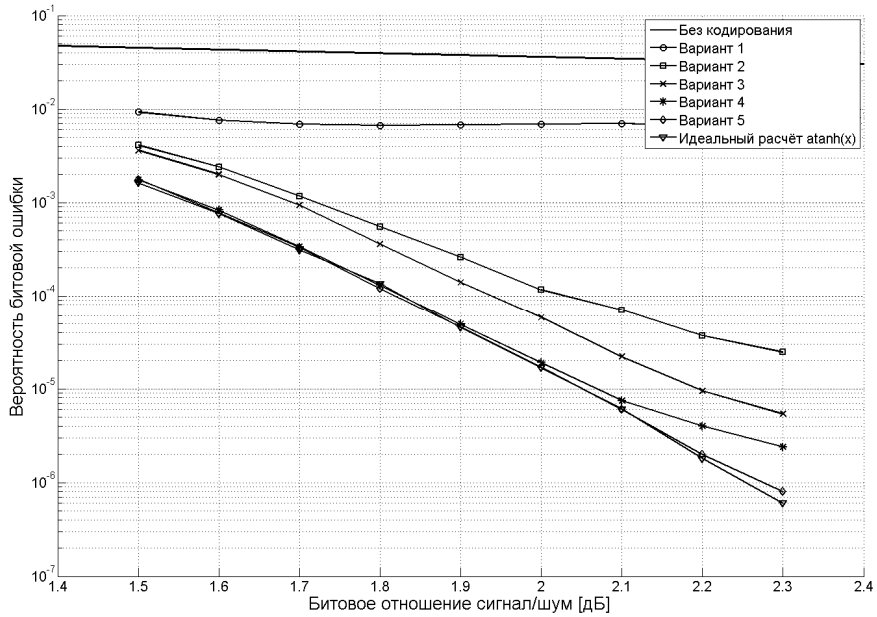

Рисунок 3.27 – BER для различных вариантов аппроксимации atanh(x)

Таблица 3.3 - Коэффициенты уравнений аппроксимации atanh(x)

Вариант аппроксимации	Участок	Аргумент x	k	b	σ^2
1	1	[0; 0.99)	1.3733	0	0.117
	2	[0.99; 1]	764.0433	-755.0433	
2	1	[0; 0.99)	1.3733	0	0.053
	2	[0.99;1]	164.0433	-161.0433	
3	1	[0; 0.9)	1.3733	0	0.018
	2	[0.9;1]	11.9162	-9.4886	
4	1	[0; 0.7)	1.196	-0.0323	0.0083
	2	[0.7;0.9)	2.9187	-1.214	
	3	[0.9;1]	10.8717	-8.3717	
5	1	[0; 0.7)	1.196	-0.0323	0.0052
	2	[0.7;0.9)	2.9187	-1.214	
	3	[0.9;0.999)	10.8717	-8.3717	
	4	[0.999; 1]	2510.9	-2505.9	

Сочетание вариантов аппроксимаций функций гиперболического тангенса и гиперболического арктангенса приводит к следующим результатам на рисунке 3.28.

Моделирование показало, что энергетический проигрыш от использования аппроксимаций по сравнению с идеальным «Belief propagation» варьируется в диапазоне от 0.02 до 0.25 дБ по уровню 10^{-5} в зависимости от точности аппроксимации.

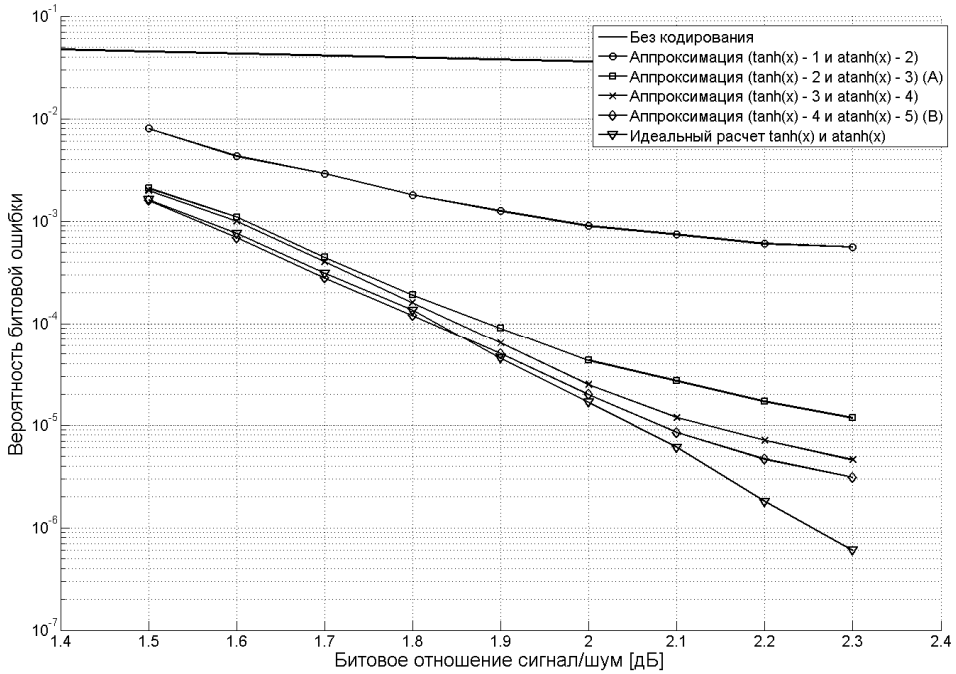

Рисунок 3.28 – BER для различных вариантов аппроксимации tanh(x) и atanh(x).

На рисунке 3.29 представлена зависимость математического ожидания числа итераций от битового отношения сигнал/шум для различных вариантов аппроксимаций. Алгоритмы с грубой аппроксимацией в среднем требуют большее число итераций на декодирование, чем алгоритмы с более точной аппроксимацией (то есть с меньшим значением σ^2).

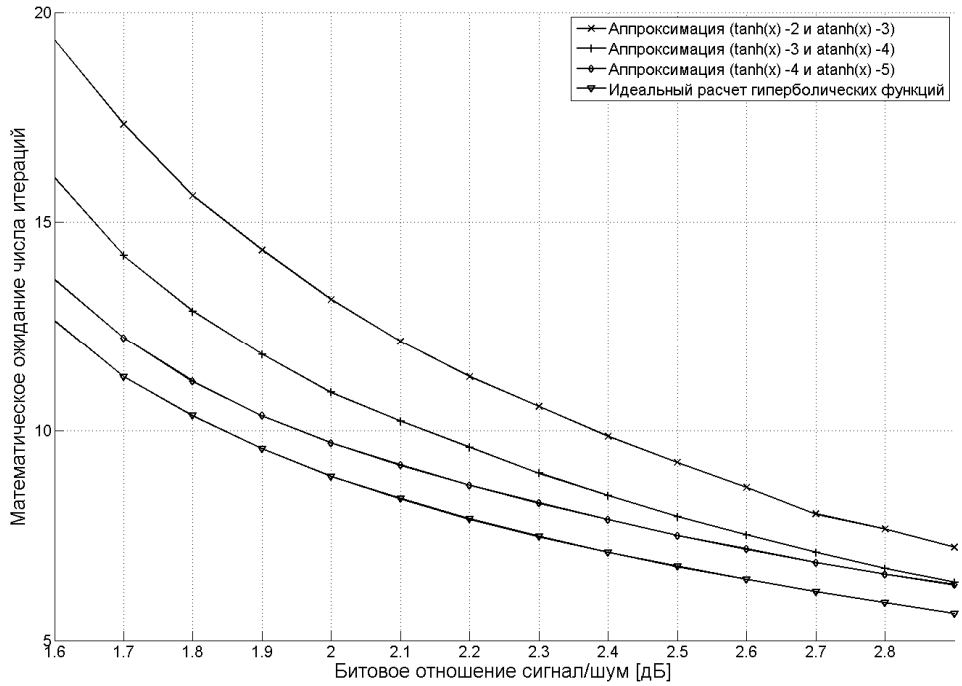

Рисунок 3.29 – Среднее число итераций для различных вариантов аппроксимации

3.5 Выводы по главе

1. Предложенная методика представления матрицы проверки на четность позволяет экономить ресурсы памяти, предназначенные для её хранения.

2. Разработанная имитационная модель служит для получения ряда статистических характеристик декодирования и отладки программной реализации, предназначенной для последующего внедрения.

3. Рекомендованные значения коэффициента α и корректирующей константы c, обеспечивающие наилучшую помехоустойчивость для соответствующих алгоритмов декодирования в рамках рассматриваемого кода с малой плотностью проверок на четность, составили 1.33 и 0.6, соответственно.

4. Зависимости числа итераций от величины порога при многопороговом декодировании, полученные на имитационной модели, показали, что с уменьшением величины порога растет среднее число итераций, выполняемое декодером.

5. На имитационной модели выявлена связь между точностью аппроксимации гиперболических функций и помехоустойчивостью алгоритма «Belief propagation». Рассмотрены и рекомендованы варианты аппроксимаций гиперболических функций, обеспечивающие наилучшую помехоустойчивость.

ГЛАВА 4. Исследование характеристик декодирования БЧХ и LDPC кодов при обработке сигнала L1C

В данной главе приведен пример применения предлагаемой методики выбора алгоритма декодирования LDPC кодов, а также проанализированы характеристики декодирования LDPC кодов на основе результатов обработки сигнала L1C. Кроме того, предложен способ идентификации инверсии битового потока сигнала L1C за счет внутренних ресурсов LDPC декодера.

4.1 Пример применения методики выбора алгоритма декодирования LDPC

В качестве примера задается требование обеспечения вероятности ошибки на выходе декодера 10^{-5} при битовом отношении сигнал/шум 2.2 дБ. Согласно предлагаемой методике, вначале происходит локализация алгоритмов, которые могут обеспечить заданную вероятность ошибки при указанном отношении сигнал/шум. В конкретном случае на рисунке 3.18 таких алгоритмов 7: «Min-sum normalized», «UMP BP normalized», «С варьируемым порогом (20;-20) normalized», «Min-sum offset», «UMP BP offset», «С варьируемым порогом (20;-20) offset», «Belief propagation».

Как отмечалось выше, алгоритм с распространением доверия «Belief propagation» редко реализуется на практике в чистом виде и будет вычеркнут из рассмотрения. Для остальных алгоритмов необходимо рассчитать комплексный показатель сложности, учитывающий суммарную сложность декодирования на итерацию (таблица 2.12) и среднее число итераций (рис. 3.19), которое используется алгоритмами при фиксированном отношении сигнал/шум. При учете суммарной сложности алгоритма на итерацию декодирования, как уже отмечалось в подразделе 2.4, следует учитывать «веса» операций различного типа при их суммировании. В данном подразделе, в качестве примера, «веса» приятны равными 1. Разработчики, используя полученные во второй главе значения сложности по каждому типу операции, могут провести суммирование с соответствующими для их аппаратуры «весами».

В таблице 4.1 и на рисунке 4.1 показаны значения комплексных показателей сложности для локализованных алгоритмов. Согласно полученным значениям, для удовлетворения сформулированного выше требования с наименьшими «затратами» следует отдать предпочтение алгоритмам «UMP BP offset» и «UMP BP normalized».

Таблица 4.1 – Значения комплексных показателей сложности (2.2 дБ)

Алгоритм	Среднее число итераций	Суммарная сложность на итерацию	Комплексный показатель сложности*
UMP BP offset	8.84	164579	1454878
UMP BP normalized	9.14	159761	1460216
Min-sum offset	8.84	182813	1616067
Min-sum normalized	9.14	177995	1626874
С варьируемым порогом (20;-20) offset	10.81	174215	1883264
С варьируемым порогом (20;-20) normalized	11.64	169397	1971781

* - *значения округлены до целых*

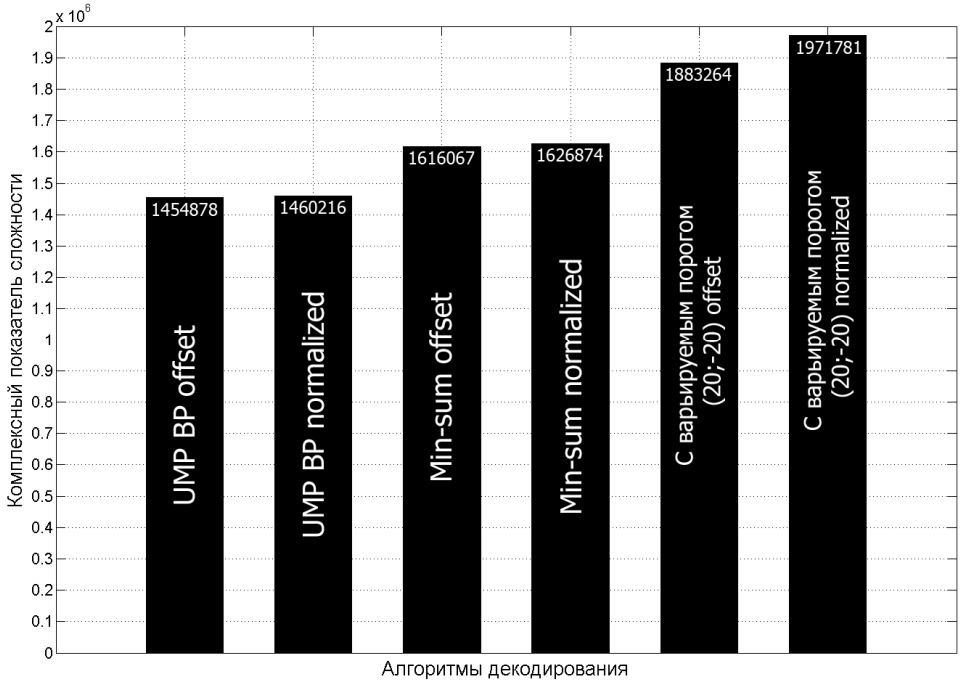

Рисунок 4.1 – Гистограмма значений комплексного показателя сложности

4.2 Исходные данные для декодирования

В качестве «мягких» решений на входах декодеров с малой плотностью проверок на четность и БЧХ используется синфазная (**I**) составляющая радиосигнала L1C. Выборка представлена на рисунке 4.2.

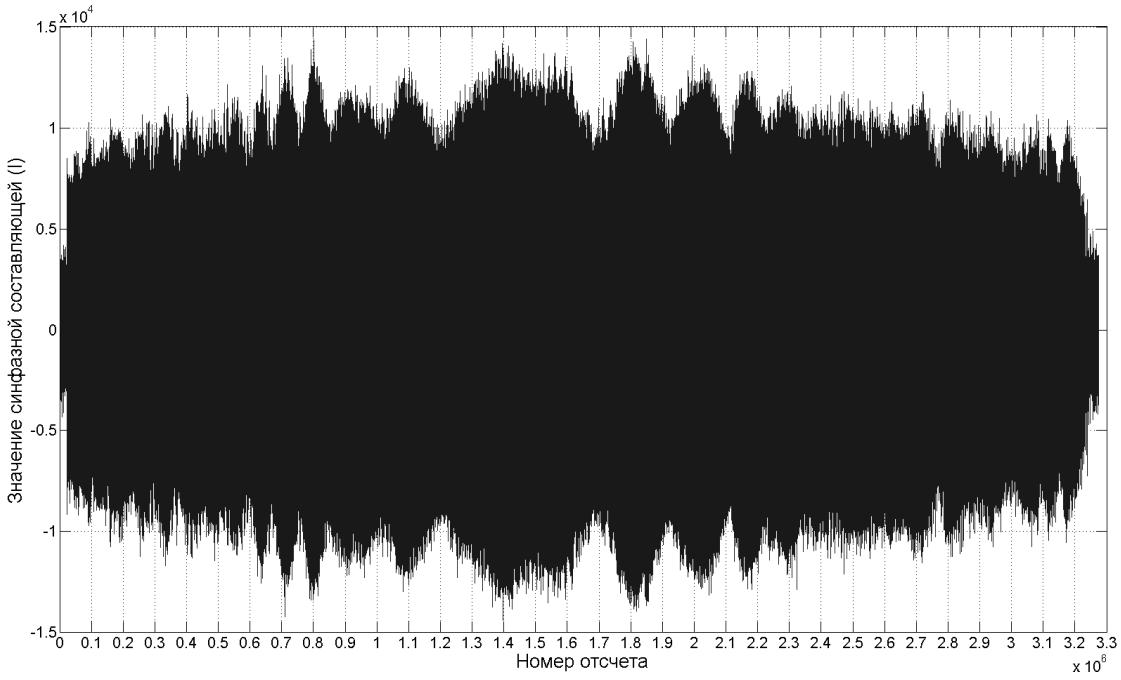

Рисунок 4.2 – Выборка отсчетов сигнала L1C для декодирования

Решающее правило для «жестких» решений принятых символов *C(i)* относительно «мягких» значений синфазной составляющей *I(i)* выглядит следующим образом:

$$C(i) = \begin{cases} 0, \ если \quad I(i) > 0, \\ 1, \ если \quad I(i) \le 0. \end{cases} \tag{4.1}$$

4.3 Исследование БЧХ кодека

Каждый 18-ти секундный кадр сопровождается номером эпохи, который изменяется в диапазоне от 0 (000000000) до 399 (110001111) и представляется 9-ю битами. В качестве корректирующего кода, обеспечивающего целостность номера эпохи, используется БЧХ код (52,9).

Процедура кодирования номера эпохи описывается следующим образом. 8 младших разрядов номера эпохи загружаются в сдвиговый регистр, изображенный на рисунке 4.3. Обратные связи регистра задаются полиномом $x^8 + x^7 + x^6 + x^5 + x^4 + x + 1$. Загрузку необходимо осуществлять от младших разрядов к старшим. После этого происходит 51 сдвиг вправо. В результате этой операции, на выходе образуется кодовое слово длиной 51 символ. Девятый (самый старший разряд) номера эпохи прибавляется по модулю 2 к каждому из 51 сгенерированных символов, после чего встает в начало этих 51 символов, образуя 52-битный код. Таким образом, если самый старший разряд в номере эпохи «1», то произойдет полная

инверсия сгенерированной регистром последовательности из 51 символов и вперед них, первым разрядом в 52-битном слове, встанет «1». В противном случае, если самый старший разряд в номере эпохи «0», сгенерированная регистром последовательность не претерпит изменений, и вперед нее встанет «0», так же образуя 52-битный код.

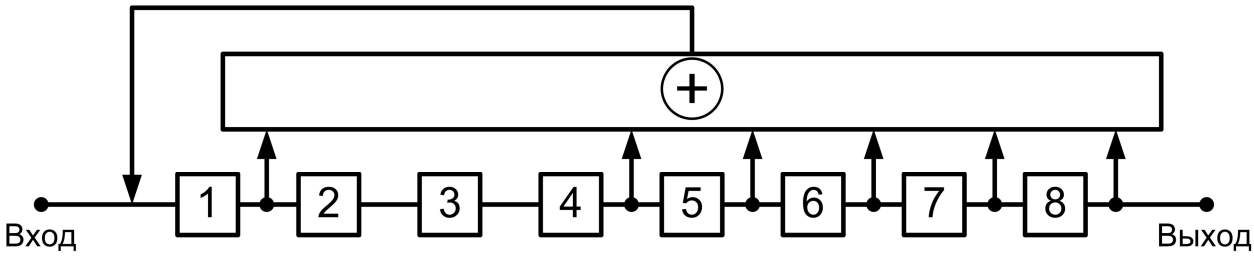

Рисунок 4.3 – Регистр сдвига

Перед декодированием БЧХ кода необходимо сгенерировать регистром сдвига все 256 возможных кодовых слов длиной 51 символ. Вперед каждого из этих слов необходимо поставить «0». В результате имеется 256 гипотез по 52 символа, с каждой из которых необходимо сравнить пришедшие 52 символа БЧХ кода следующим образом: если соответствующие разряды гипотетического и пришедшего кодового слова совпадают, то значение корреляции пришедшего слова с этим гипотетическим словом увеличивается на абсолютную величину «мягкого» решения пришедшего символа. В противном случае величина вычитается. Процедура сравнения повторяется для всех 256 гипотез. Далее происходит поиск максимального абсолютного значения корреляции и соответствующей этому значению гипотезы. Первые 9 разрядов этой гипотезы децимируются. Если значение корреляции, соответствующее максимальному абсолютному значению, положительно, полученные 9 разрядов уже представляют собой номер эпохи в двоичной системе. В противном случае происходит инвертирование первого старшего разряда с «0» на «1».

Моделирование работы БЧХ кодека проводилось на основе разработанной модели, структура которой приведена на рисунке 3.4. В этой модели низкоплотностный кодек был заменен на кодек БЧХ. Согласно [57], минимальное расстояние Хэмминга у применяемого БЧХ кода равно 20. Отсюда следует, что гарантированное число ошибок t, которое исправляет декодер при работе с «жесткими» решениями демодулятора, равно 9.

$$t = \left\lfloor \frac{d_{\min} - 1}{2} \right\rfloor = \left\lfloor \frac{20 - 1}{2} \right\rfloor = \lfloor 9.5 \rfloor = 9 . \qquad (4.2)$$

В ходе имитационного моделирования на выборке в 100000 слов при различных отношениях сигнал/шум не было выявлено вектора ошибок с весом меньше 10, который бы привел к ошибкам на выходе декодера при работе с «жесткими» решениями демодулятора. Это подтверждает гарантию числа исправляемых ошибок по формуле (4.2). Характеристика помехоустойчивости BER для БЧХ кода (52,9) для «мягких» и «жестких» решений приведена на рисунке 4.4. При переходе от «жестких» решений к «мягким» уже нельзя гарантировать число исправляемых ошибок *t*. Однако моделирование показывает, что переход к «мягким» решениям позволяет повысить исправляющую способность кода. Выигрыш от применения «мягких» решений по сравнению с «жесткими» решениями составил 2 дБ по уровню 10^{-5}. Аналогичные значения энергетического выигрыша для БЧХ кодов получены в [40,67].

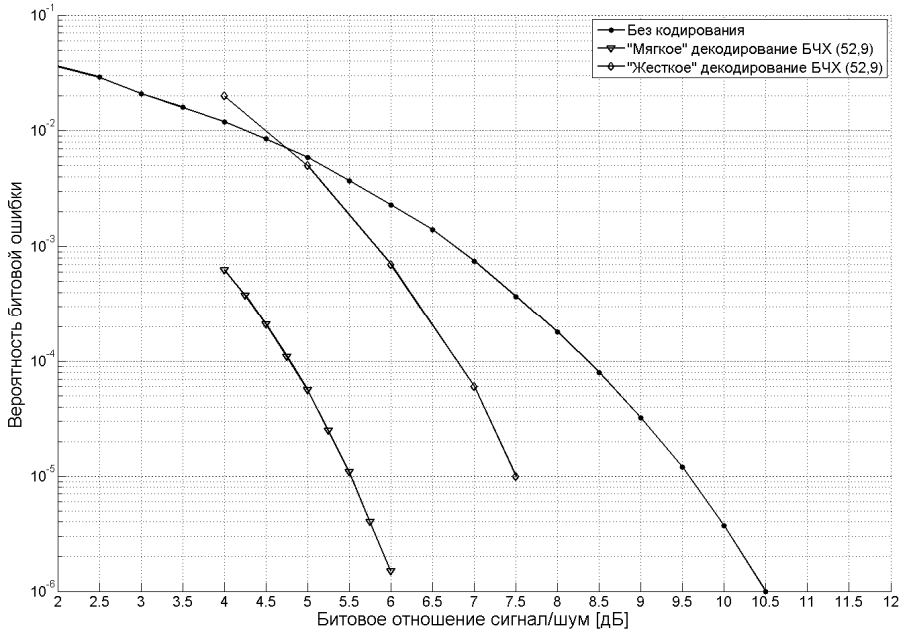

Рисунок 4.4 – BER для БЧХ кода (52,9) с «мягкими» и «жесткими» решениями.

4.4 Результаты декодирования выборки сигнала L1C

Массив, состоящий из Subframe2 и Subframe3, подвергается блоковому перемежению на передающей стороне. Характер процедуры представлен на рисунке 4.5. Перемежение заключается в записи информационной последовательности в матрицу по строкам с последующим считыванием последовательности по столбцам. Как следствие, перед декодированием Subframe2 и Subframe3 на приемной стороне необходимо выполнить процедуру блокового деперемежения.

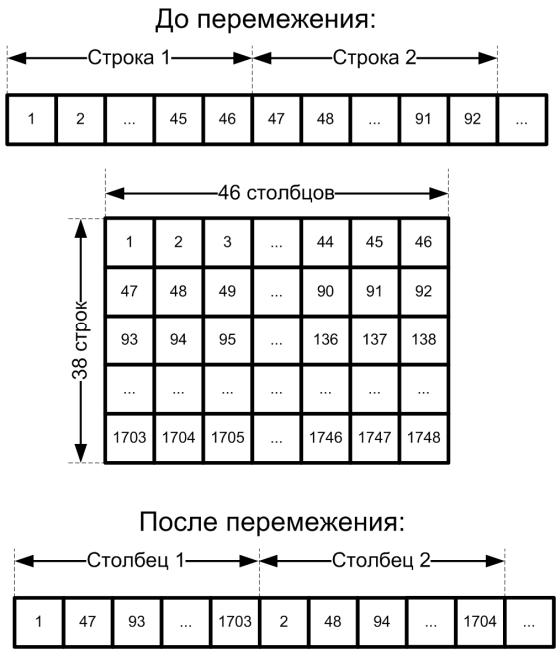

Рисунок 4.5 – Блоковое перемежение

На выборке было декодировано 1784 кадра. Первые 22000 отсчетов не несут полезной информации. Кадровая синхронизация происходит на 22618 отсчете, где берет начало первый декодированный кадр. По выходной информации БЧХ декодера кадр имеет номер эпохи 4, а по декодированным битам 14-21 из Subframe2, обозначающим номер двухчасового отрезка ITOW, данный кадр относится к 35-ому двухчасовому отрезку из 84-ёх возможных (84 двухчасовых отрезка составляют одну неделю). Далее декодированные кадры идут друг за другом с возрастанием номера эпохи на 1 и с фиксированным номером двухчасового отрезка ITOW 35 вплоть до 396 декодированного кадра с номером эпохи 399. После кадра с номером эпохи 399 следует кадр с номером эпохи 0, являющийся последним кадром в 35-ом двухчасовом отрезке ITOW. Следующий декодированный кадр с номером эпохи 1 относится, судя по разрядам 14-21 из Subframe2, к 36-ому двухчасовому отрезку. Затем следуют три полных двухчасовых отрезка с номерами 36,37,38 и неполный двухчасовой отрезок 39, содержащий кадры с номерами эпох 1-187. Первый и последний двухчасовой цикл «обрезаны» за счет того, что начало и окончание записи отсчетов началось в произвольный момент времени. Изменение номера эпохи в зависимости от номера декодированного кадра показано на рисунке 4.6.

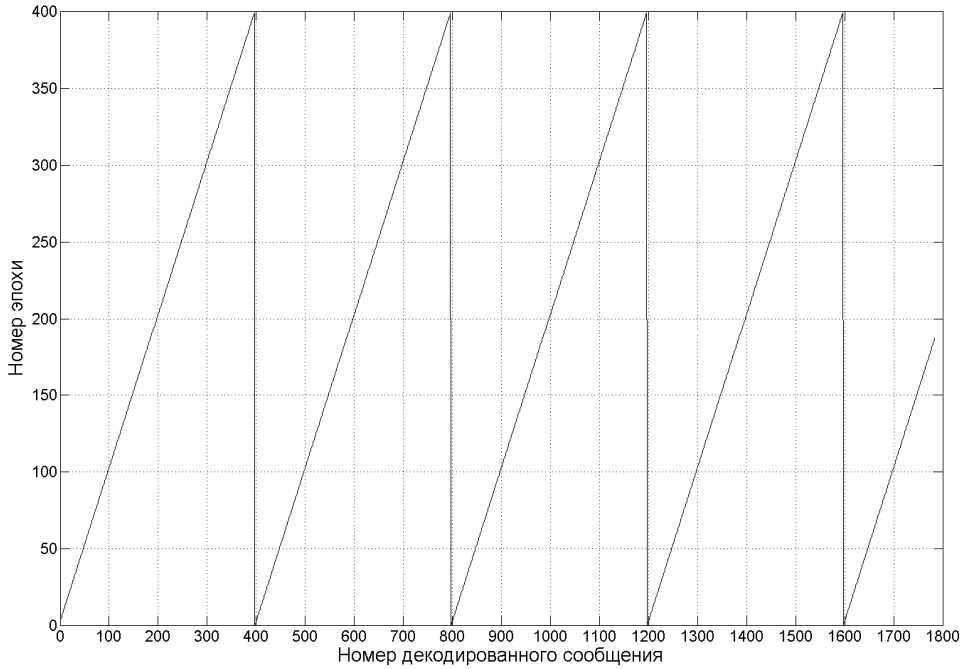

Рисунок 4.6 – Изменение номеров эпох

В таблице 4.2 приводится соответствие номеров эпох и двухчасовых отрезков ITOW. Каждый новый двухчасовой отрезок ITOW, кроме 35-ого и 39-ого, содержит 400 номеров эпох начиная с 1-ого номера и заканчивая 0-ым номером. Кроме того, у каждого из декодированных кадров сходится контрольная сумма CRC-24. Из этого можно сделать вывод, что декодирование произведено корректно.

Таблица 4.2. Расшифровка значений ITOW и номеров эпох.

Номер ITOW	Номера декодированных кадров	Номера эпох
35	1 – 397	4, 5, 6 – 398, 399, 0
36	398 – 797	1, 2, 3 – 398, 399, 0
37	798 – 1197	1, 2, 3 – 398, 399, 0
38	1198 – 1597	1, 2, 3 – 398, 399, 0
39	1598 – 1784	1, 2, 3 – 187

Работу LDPC декодера характеризует статистика кратности исправленных декодером ошибок и числа итераций, которое потребовалось для их исправления. Статистика приведена на рисунке 4.7.

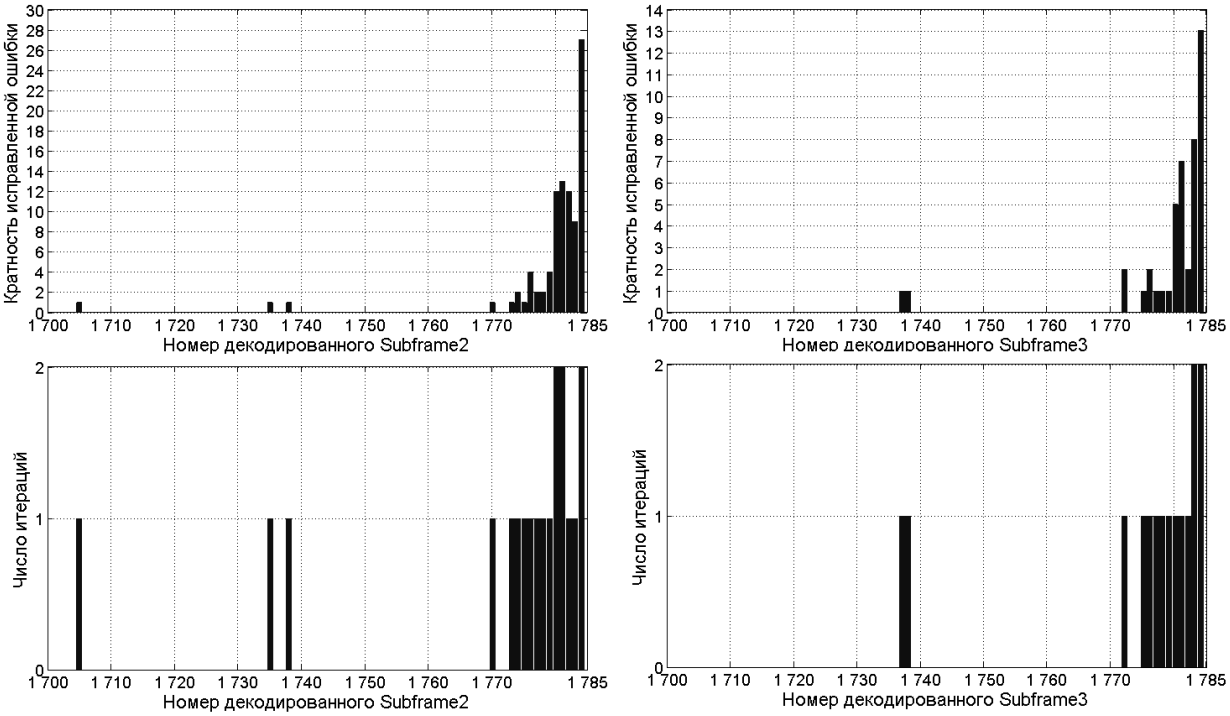

Рисунок 4.7 – Статистика по декодированию Subframe2 и Subframe3

Статистика показана для последних 84 кадров со спутника. Все предыдущие кадры не содержали ошибок. В ходе своей работы декодер Subframe2 столкнулся с ошибками 1-27 кратности, а декодер Subframe3 с ошибками 1-13 кратности. Ошибки в Subframe2 и в Subframe3 были исправлены за 1-2 итерации декодирования.

Статистика декодирования БЧХ представлена на рисунке 4.8. Как и LDPC декодер, декодер БЧХ столкнулся с ошибками при декодировании лишь на завершающей части рассматриваемой выборки.

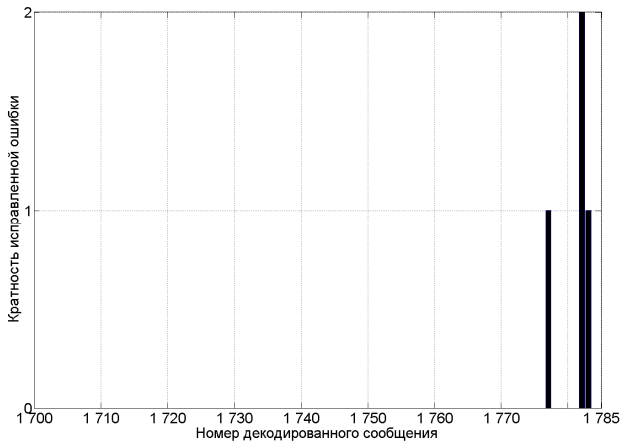

Рисунок 4.8 – Статистика по декодированию БЧХ кода

Из анализа статистики по декодированию составных частей сигнала L1C, представленной на рисунке 4.7 и 4.8, видно, что обрабатываемый сигнал получен при относительно высоком отношении сигнал/шум, так как содержит малое число ошибок в принятых кадрах. Этот факт затрудняет сбор статистики по числу использованных итераций декодера LDPC и сопоставлению полученных результатов при обработке реального сигнала с результатами, полученными на имитационной модели в главе 3. Для анализа работы декодера требуется выборка, полученная при низком энергетическом потенциале и содержащая большое число ошибок в принимаемых кадрах.

4.5 Идентификация инверсии битового потока сигнала L1C

Выборка, полученная при низком энергетическом потенциале (битовое отношение сигнал/шум ~3-5 дБ) и содержащая 3646800 отсчетов (ровно 2026 кадров), представлена на рисунке 4.9.

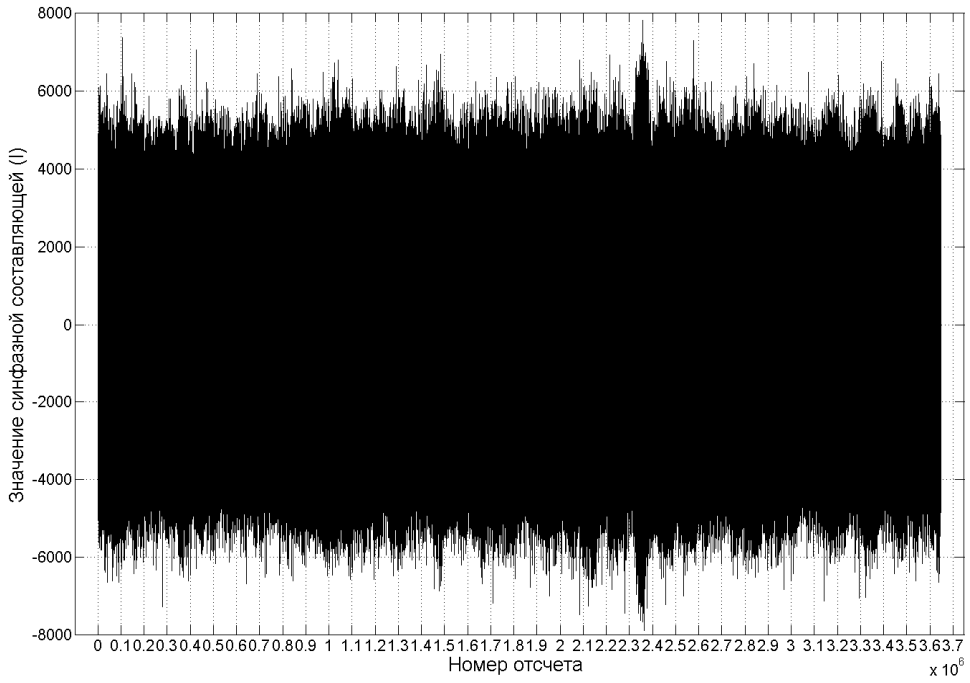

Рисунок 4.9 – Выборка отсчетов сигнала L1C

На выборке было декодировано 1012 кадров из 2026 (~50%). Потери обусловлены тем, что при обработке выборки сигнала с низким энергетическим потенциалом в тракте приемника есть вероятность срыва слежения за фазой в схеме Костаса. Это может приводить к инверсии «мягких» решений на входе декодера, что равносильно внесению ошибок в принимаемый кадр сигнала L1C. Если под этот негативный эффект попадает лишь малая часть кадра, есть

вероятность нивелировать проблему за счет мощных техник коррекции ошибок LDPC. Однако если под инверсию попал весь кадр, или большая его часть, необходимо идентифицировать факт произошедшей инверсии и принимать общее системное решение.

На рисунке 4.10 показано изменение полярности кадров в составе обрабатываемой выборки. Прямую полярность имеют 1012 кадров (декодированы успешно), а обратную – 988 (потеряны при декодировании). Еще 26 кадров находятся на «стыках» между сменой полярности. Такие кадры инвертированы частично и чаще всего не поддаются декодированию. В отличие от кадров, находящихся на «стыках», полностью инвертированные кадры можно декодировать, умножив «мягкие» решения по этим кадрам на -1. Для этого необходимо идентифицировать факт произошедшей инверсии.

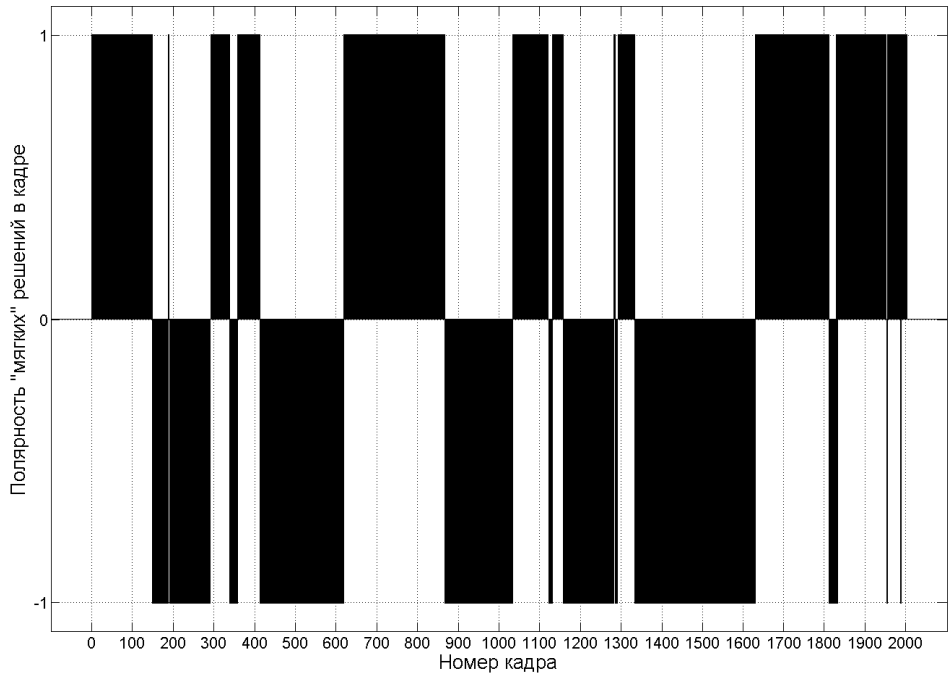

Рисунок 4.10 – Полярность кадров в составе обрабатываемой выборки

В структуре сигнала L1C не предусмотрено фиксированной преамбулы, по которой можно было бы принимать решение об инверсии приятного кадра. Как следствие, появилась необходимость разработки способа идентификации инверсии битового потока сигнала L1C. В данной работе идентифицировать факт инверсии предлагается за счет внутренних ресурсов LDPC декодера, обрабатывающего Subframe2.

Идея идентификации инверсии заключается в подаче синхронизированного кадра сигнала L1C на вход декодера и последующем декодировании его составной части Subframe2 в течение большого, но ограниченного числа итераций (до 100 итераций в данной работе). Если в

течение 100 итераций не удалось декодировать Subframe2, то принимается решение о полной или частичной инверсии всего кадра. С первого взгляда может показаться некорректным вынесение решения по всему кадру на основе анализа только одной его части. Однако следует учитывать, что после внесения инверсии, имеющей характер пакета ошибок, и до декодирования происходит блоковое деперемежение содержимого в кадре. Это означает, что в какой бы части принятого кадра не произошла инверсия, после деперемежения она будет распространена по всему кадру.

Недостатком данного способа является долгое время для принятия решения об инверсии кадра.

4.5.1 Идентификация инверсии по ограниченному числу итераций

Чтобы сократить время на принятие решения о полной или частичной инверсии кадра можно выставить меньшее ограничение на число итераций для декодера Subframe2. Выставленное ограничение будет являться тем числом итераций, за которое будет идентифицироваться факт инверсии, однако в этом случае есть опасность не декодировать кодовые слова, требующие больше итераций, чем выставленное ограничение. На рисунке 4.11 демонстрируются зависимости числа потерянных кадров от ограничения на число итераций для декодера Subframe2, полученные на имитационной модели для различных битовых отношений сигнал/шум и при обработке выборки реального сигнала без учета кадров, находящих на «стыках». К примеру, при выставленном на имитационной модели отношении сигнал/шум 3 дБ и ограничении в 7 проводимых итераций на выборке в 2000 кадров произошла потеря 300 кадров. Это означает, что 300 Subframe2 потребовали больше чем 7 итераций декодирования. При увеличении ограничения до 8 итераций теряется лишь ~170 кадров. Увеличение отношения сигнал/шум при фиксированном ограничении числа итераций приводит к уменьшению потерь кадров.

Кривая для реального сигнала вобулирует в области кривых для 3 и 5 дБ, снятых на имитационной модели, так как именно в таком диапазоне битового отношения сигнал/шум получена обрабатываемая выборка реального сигнала.

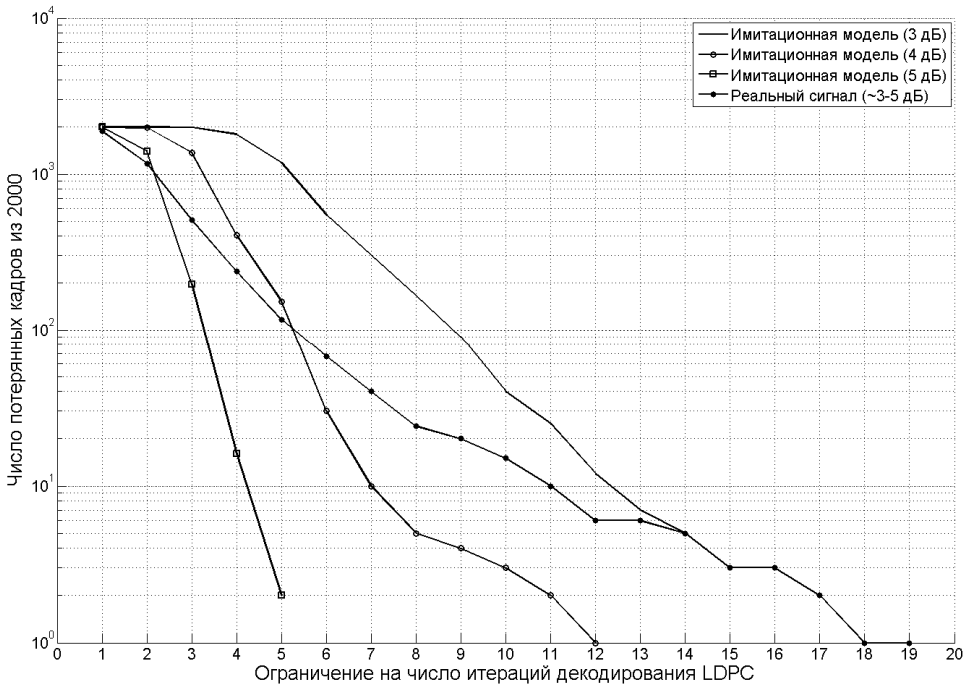

Рисунок 4.11 – Потери при ограничении декодера Subframe2

Чем сильнее декодер ограничен в возможностях по проведению итеративного декодирования, тем быстрее он будет принимать решение об инверсии поданных на вход данных. Платой за быстрое принятие решения служит потеря кадров, требующих большого числа итераций на декодирование. Потери инициируют поиск других путей идентификации инверсии битового потока.

4.5.2 Идентификация инверсии по сходимости синдрома

В данной работе предлагается не ограничивать декодер в проведении итеративного декодирования, а идентифицировать факт инверсии по анализу сходимости синдрома. Перед декодированием и после каждой новой проделанной итерации декодер рассчитывает уравнения на четность (синдром), в которых участвуют полученные в результате проведения итерации декодирования «жесткие» символы кодового слова. Моделирование показало (рис. 3.21), что при нормальном ходе декодирования средний вес синдрома (число несошедшихся уравнений на четность) уменьшается от одной итерации к другой. Это связано с тем, что, как правило, с каждой новой проделанной итерацией в принятом кодовом слове становится все меньше ошибок, число которых инициирует вес синдрома.

На рисунке 4.12 показана сходимость синдрома по ходу декодирования одного из Subframe2, изначально содержащего 127 ошибок и успешно декодированного за 17 итераций.

Динамика такова, что на каждой следующей итерации вес синдрома становится меньше, чем на предыдущей.

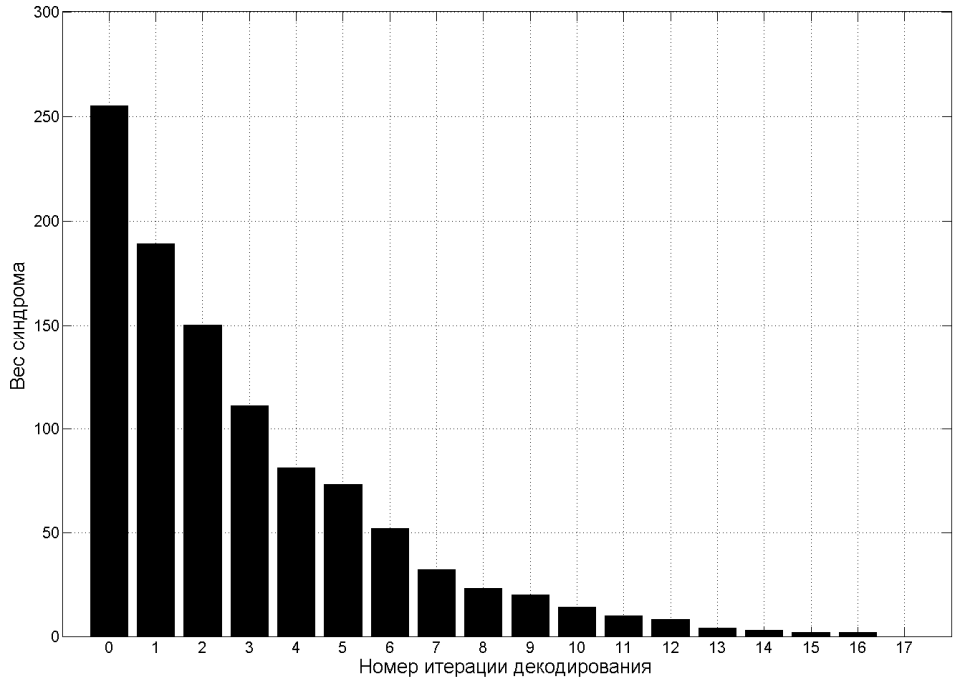

Рисунок 4.12 – Пример сходимости синдрома при декодировании Subframe2

В случае полной инверсии кадра вес синдрома при декодировании такого Subframe2 не сойдется к нулю. Пример сходимости синдрома для такого случая показан на рисунке 4.13.

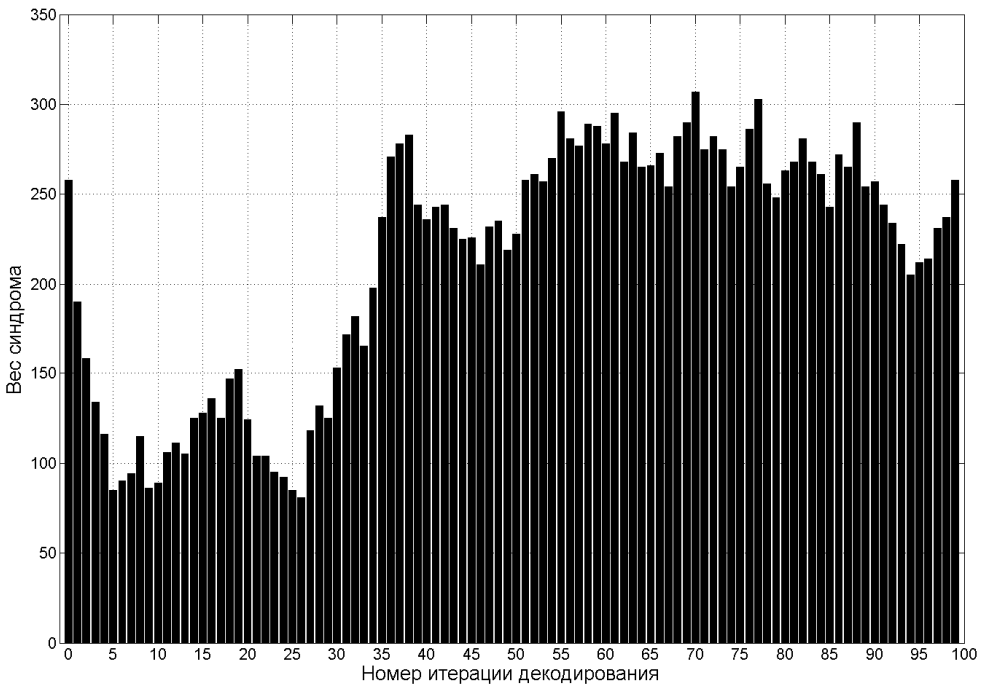

Рисунок 4.13 – Пример сходимости синдрома при декодировании Subframe2 с инверсией

В данном случае значение веса синдрома от итерации к итерации ведет себя случайным образом. Это связано с зависимостью веса синдрома не только от количества ошибок, но и от места, где эти ошибки произошли. При таком обилии ошибок во время декодирования случайным образом исправляются и вносятся ошибки, причем на каждой новой итерации это происходит в разных местах, что приводит динамику изменения веса синдрома к непредсказуемому характеру.

В первом приближении можно сформулировать критерий, позволяющий отличить динамику на рисунке 4.12 от динамики на рисунке 4.13: если значение веса синдрома после проведения текущей итерации больше, чем после проведения предыдущей – выносится решение о том, что дальнейшее декодирование не имеет смысла и процесс останавливается. Таким образом, в конкретном случае уже после 6 итераций декодирования можно сделать вывод о том, что принятый кадр является инверсным.

Сформулированный критерий требует уточнения. Дело в том, что проверка на четность не может обнаружить ошибки четной кратности. Это означает, что если проверка содержит сразу две ошибки (или любое четное число), она будет сходиться (элемент синдрома 0). После проведения одной итерации декодирования одна из этих ошибок может быть исправлена. Число ошибок станет нечетным, что будет зафиксировано проверкой на четность (элемент синдрома 1). Таким образом, число ошибок в результате проведения итерации декодирования уменьшится, а число несошедшихся уравнений на четность (вес синдрома) увеличится.

На рисунке 4.14 показан пример «нестандартной» сходимости синдрома по ходу декодирования одного из Subframe2, изначально содержащего 142 ошибки и успешно декодированного за 15 итераций. Вес синдрома при декодировании сошелся к нулю, что эквивалентно выполнению всех проверок на четность, однако по ходу декодирования трижды был нарушен сформулированный выше критерий остановки декодирования. Выходом из данной ситуации является разрешение превышения значения веса синдрома после текущей итерации над значением после предыдущей итерации, но только фиксированное число раз N. Как только число превышений веса синдрома по сравнению с предыдущим значением превышает N – выносится решение об инверсии битового потока.

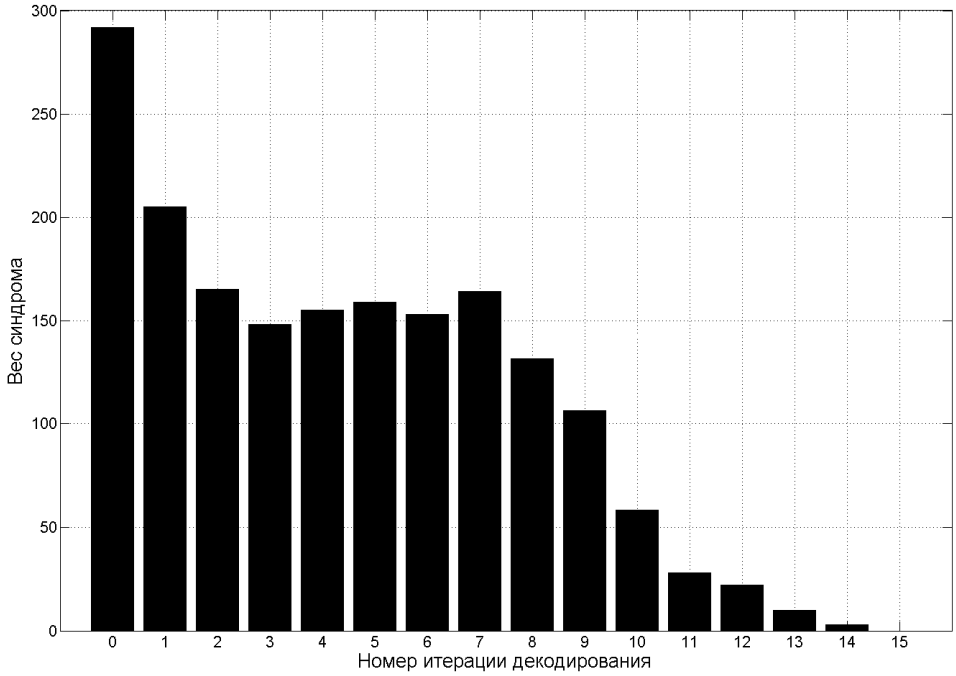

Рисунок 4.14 – Пример «нестандартной» сходимости синдрома

С уменьшением N растет риск потери кадров, имеющих «нестандартную» сходимость синдрома. На рисунке 4.15 демонстрируются зависимости числа потерянных кадров от выставленного значения N, полученные на имитационной модели для различных битовых отношений сигнал/шум и при обработке выборки реального сигнала без учета кадров, находящихся на «стыках». К примеру, при выставленном на имитационной модели отношении сигнал/шум 3 дБ и значении N=0 на выборке в 2000 кадров произошла потеря 26 кадров. Это означает, что 26 Subframe2 имеют хотя бы одно превышение текущего значения веса синдрома над полученным весом на предыдущей итерации. Увеличение значения N до 1 приводит к потере лишь 5 кадров. Это означает, что 5 Subframe2 имеют два и больше превышений. Увеличение отношения сигнал/шум приводит к уменьшению числа «нестандартных» всплесков в сходимости синдромов.

Как и в примере выше, кривая для реального сигнала вобулирует в области кривых для 3 и 5 дБ. Из рисунка становится ясно, что при обработке реального сигнала лишь 14 Subframe2 из 2000 имеют «нестандартную» сходимость синдрома, причем 11 из них имеют одно превышение веса синдрома над предыдущим значением и еще 3 имеют три превышения.

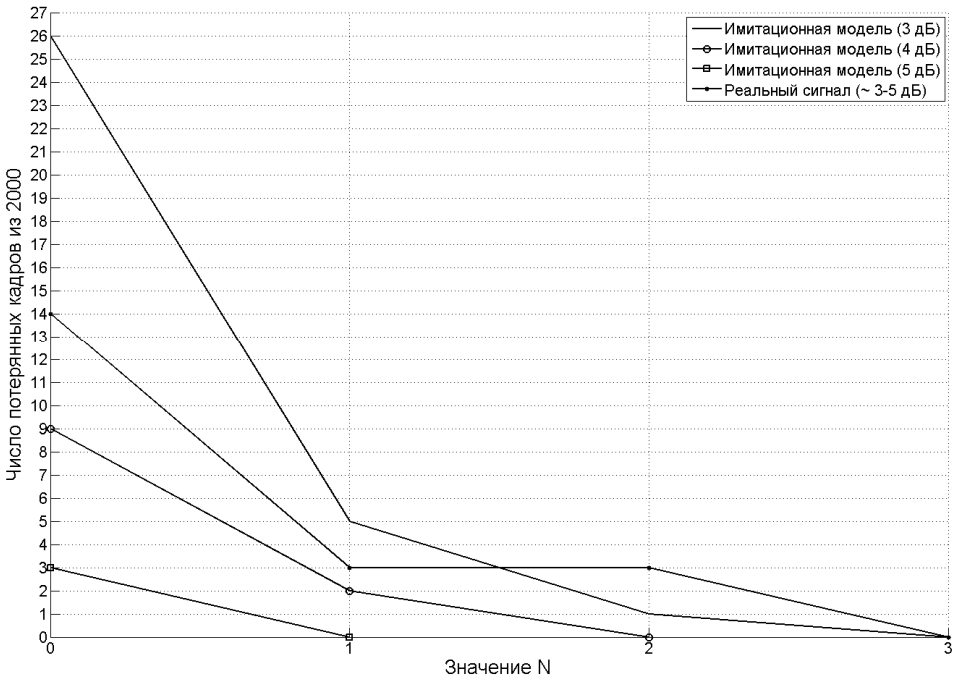

Рисунок 4.15 – Потери кадров в зависимости от разрешенного числа превышений N

На рисунке 4.16 демонстрируются зависимости среднего (а также минимального и максимального на рассмотренной выборке) числа итераций, необходимых для вынесения решения об инверсии в зависимости от выставленного значения N, полученные на имитационной модели и при обработке реального сигнала без учета кадров, находящихся на «стыках». При обработке выборки реального сигнала и выставленном значении N=0 не разрешается превышений веса синдрома после текущей итерации по сравнению с предыдущей. При этом в среднем инверсия битового потока будет определяться за 5.4 итерации, однако при обработке на выборке будет потеряно 14 кадров. Увеличивая значение N до 1, число потерянных кадров уменьшается до 3, однако среднее число итераций, необходимых для определения инверсии, увеличивается до 6.9. Аналогично интерпретируются остальные значения на рисунке 4.16. Результаты, полученные на имитационной модели, соответствуют результатам, полученным при обработке реального сигнала.

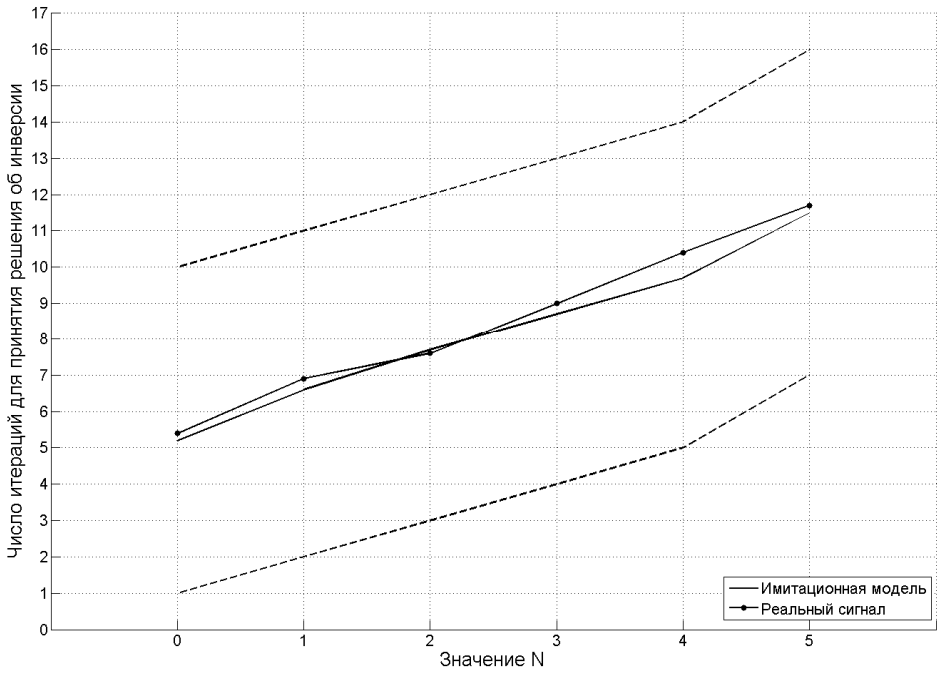

Рисунок 4.16 – Используемое число итераций для вынесения решения

Из представленных зависимостей следует очевидный вывод – чем больше значение N, тем дольше происходит его «накопление», то есть тем дольше необходимо проделывать итерации перед фиксацией необходимого количества превышений веса синдрома над его предыдущим значением.

4.5.3 Сравнение способов идентификации инверсии

Сравниваются два способа идентификации инверсии: по ограниченному числу итераций и по анализу сходимости синдрома. Критериями сравнения являются число итераций, за которое удается идентифицировать инверсию и потери кадров в процессе идентификации.

На рисунке 4.17 показаны зависимости числа итераций, необходимых для идентификации инверсии, от потерь на рассматриваемой выборке. В случае использования способа идентификации по ограниченному числу итераций потери складываются из потерь кадров, находящихся на «стыках» при смене полярности и потерь кадров, которые требуют большее число итераций на декодирование, чем выставленное ограничение. В случае использования способа идентификации по анализу сходимости синдрома потери складываются из потерь кадров, находящихся на «стыках» при смене полярности и потерь кадров, которые имеют «нестандартную» сходимость синдрома, превышающую значение N.

Без идентификации инверсии на обрабатываемой выборке было потеряно ~50% принятых кадров. Применение предложенных способов идентификации позволило сократить потери до ~1-2%, в зависимости от параметров алгоритма идентификации.

При равных потерях на выборке способ с анализом сходимости синдрома в среднем быстрее определяет факт инверсии, чем способ с ограничением числа итераций. К примеру, при потерях в 1.43 % (29 кадров из 2026) для обоих случаев способ с ограничением числа итераций будет идентифицировать инверсию за 15 итераций, а способ с анализом сходимости синдрома в среднем за 6.9.

С минимизацией потерь в обоих случаях растет время (число проделываемых итераций) на идентификацию инверсии. В случае идентификации по ограниченному числу итераций декодер вынужден проделывать все отведенные ему на декодирование итерации. В случае идентификации за счет анализа сходимости синдрома декодер имеет критерий, позволяющий принять решение об инверсии на промежуточном этапе декодирования. Критерий срабатывает в недетерминированный момент времени, так как на сходимость синдрома оказывают влияние случайные факторы, в числе которых место в кодовом слове, где произошла инверсия, значения априорных «мягких» решений на входе декодера и прочее. В связи с этим в рамках анализа сходимости синдрома следует говорить о средней величине числа итераций для идентификации инверсии.

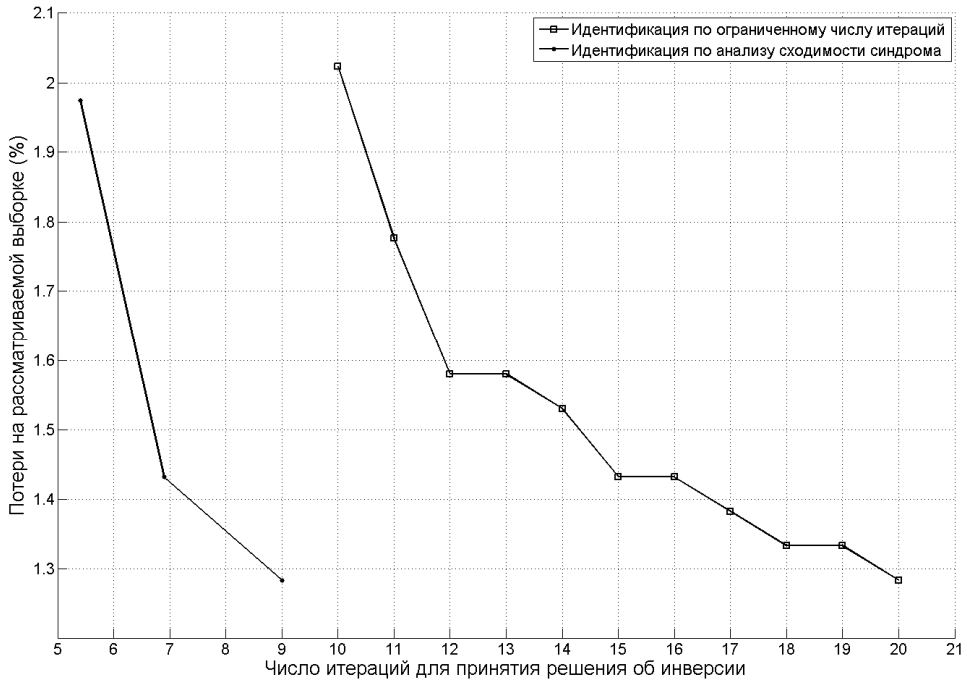

Рисунок 4.17 – Зависимость числа итераций для идентификации от потерь

На основании вышеизложенного рекомендовано использование способа идентификации инверсии по анализу сходимости синдрома.

Применяя способ идентификации инверсии посредствам анализа сходимости синдрома, на обрабатываемой выборке при N=3 было декодировано 2000 из 2026 кадров. Еще 26 кадров, как было отмечено выше, было потеряно на «стыках» при смене полярности «мягких» решений на входе декодера. Правильность декодирования подтверждается расшифровкой декодированной информации и расчетом контрольной суммы для каждого кадра. Фрагмент статистики по обработке данной выборки представлен на рисунке 4.18.

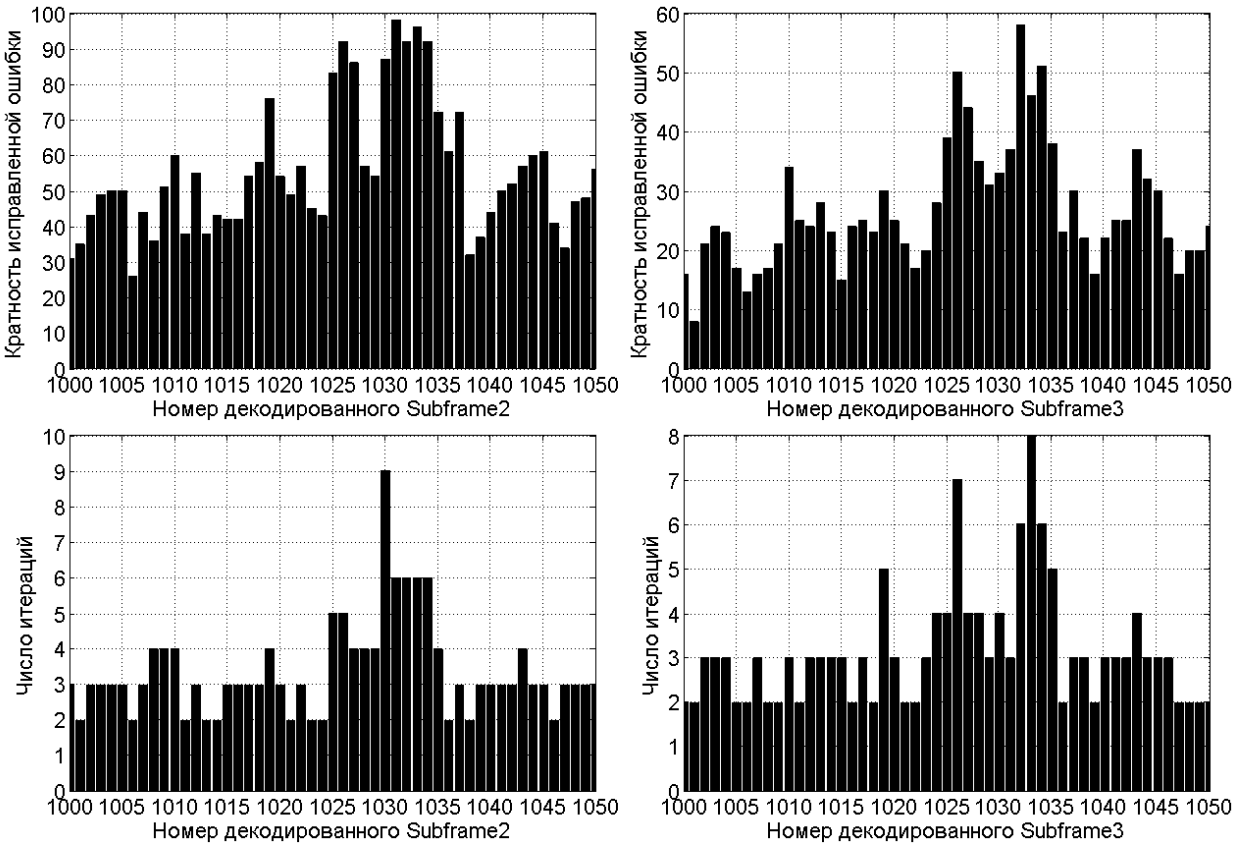

Рисунок 4.18 – Статистика по декодированию Subframe2 и Subframe3

Средняя кратность ошибки на входе декодера Subframe2 составила 46.5 (разброс от 1 до 142 ошибок). Ошибки были исправлены в среднем за 3.05 итераций. Средняя кратность ошибки на входе декодера Subframe3 составила 21.5 (разброс от 1 до 63 ошибок). Ошибки были исправлены в среднем за 2.66 итерации. Соответствующие законы распределения числа итераций представлены на рисунке 4.19.

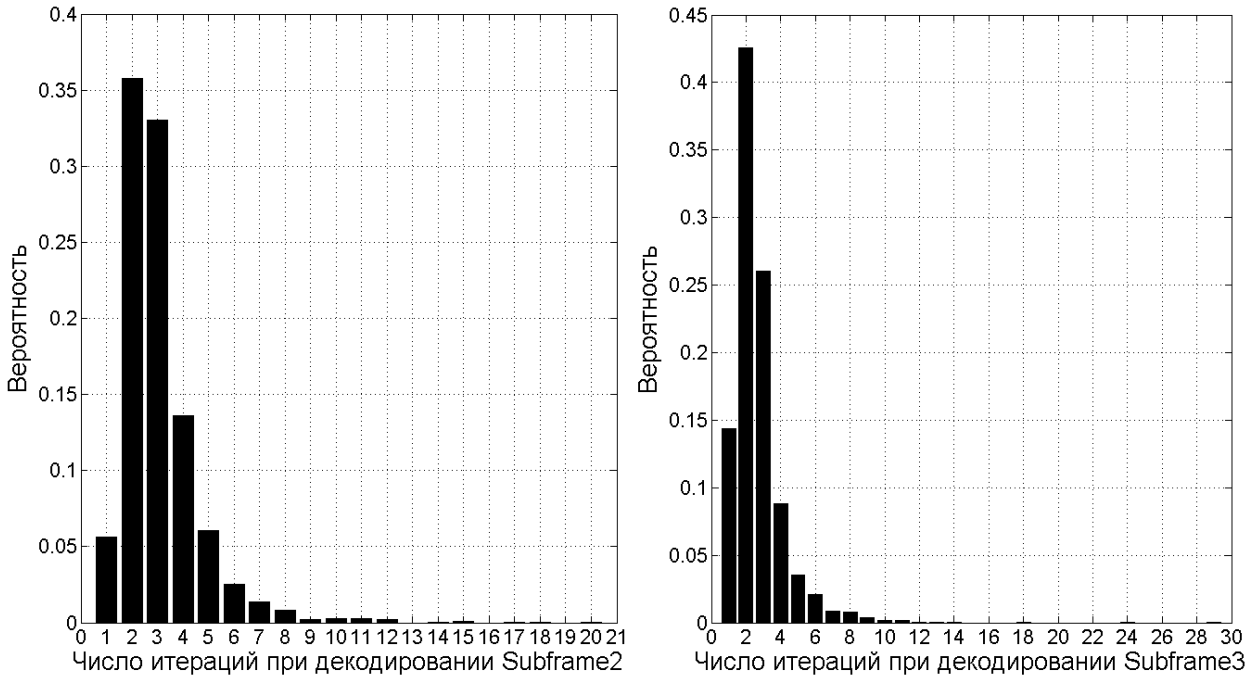

Рисунок 4.19 – Законы распределения числа итераций

Собранная статистика по числу используемых итераций при декодировании Subframe2 соответствует статистике, полученной на имитационной модели и представленной на рисунке 3.19.

4.6 Выводы по главе

1. Апробированная в данной главе методика позволяет выбрать алгоритм декодирования LDPC кода под конкретную телекоммуникационную систему.

2. Переход от использования «жестких» решений к «мягким» при декодировании рассматриваемого БЧХ кода позволяет повысить его корректирующую способность.

3. Предложенный способ идентификации инверсии позволяет определять инверсию битового потока на входе декодера.

ГЛАВА 5. Сравнение LDPC кодов и турбо кодов

В данной главе приводится краткая классификация турбо кодов и описание выбранного для сравнения турбо кодека. Анализируются результаты турбо декодирования, полученные на имитационной модели. Оценивается вычислительная сложность одной итерации турбо декодирования и проводится сравнение турбо кода и рассматриваемого кода с малой плотностью проверок на четность по выбранным критериям.

Основные результаты, полученные в рамках данной главы, опубликованы автором в [23].

5.1 Классификация турбо кодов

При создании турбо кодов использовалась парадигма каскадных кодов. Сущность турбо кодирования заключается в каскадировании двух и более составляющих кодов [8]. Соединение может быть как параллельным, так и последовательным, но в обоих случаях оно происходит через перемежитель.

Турбо коды, использующие в качестве составляющих кодов сверточные коды, называются Turbo Convolution Codes (TCC) или сверточными турбо кодами. Декодируют такие коды с помощью алгоритмов Soft Output Viterby Algorithm (SOVA), Log-MAP и его модификации Max-Log-MAP.

В отличие от сверточных турбо кодов, блоковые турбо коды произведения (Turbo Product Codes (TPC)) используют в качестве составляющих кодов линейные блоковые коды БЧХ или Хэмминга. Перемежение в этом случае не требуется, а декодирование таких кодов осуществляется по алгоритму Чейза.

Для сравнения с рассмотренным выше LDPC кодом выбран двумерный блоковый турбо код (32,26)x(32,26) со скоростью кодирования 0.66. В качестве составляющих кодов используется расширенный код Хэмминга (32,26).

5.2 Турбо кодек

Информационная последовательность длиной 676 бит кодируется выбранным турбо кодом следующим образом. Последовательность записывается в матрицу размером 26 на 26 слева направо и сверху вниз как это показано на рисунке 5.1. Каждая строка матрицы на рисунке 5.1 кодируется расширенным кодом Хэмминга (32,26). Кодирование осуществляется перемножением строк на порождающую матрицу кода Хэмминга (32,26). В результате кодирования для каждой строки вычисляются 6 проверочных символов кода Хэмминга. Структура закодированной по строкам информации представлена на рисунке 5.2. Затемненные символы – проверочные.

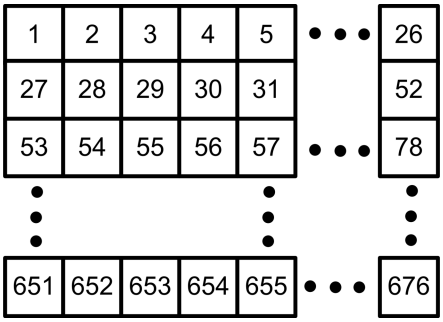

Рисунок 5.1 – Структура информационной последовательности перед кодированием

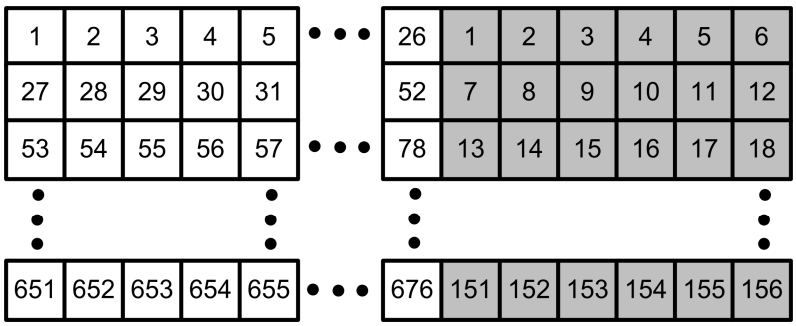

Рисунок 5.2 – Структура информационной последовательности после кодирования по строкам

Полученная на рисунке 5.2 структура кодируется по столбцам аналогичным образом. Результирующая, закодированная турбо кодом структура представлена на рисунке 5.3. Структура передается в эфир слева направо и сверху вниз.

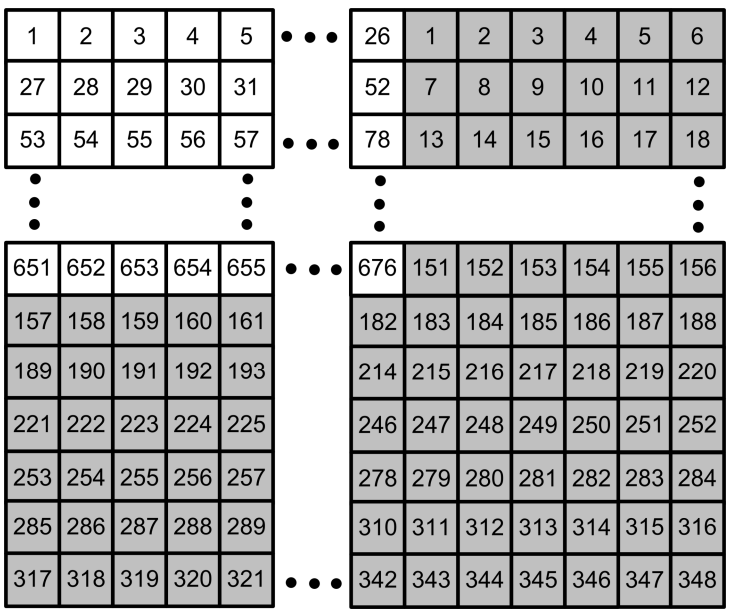

Рисунок 5.3 – Структура данных, закодированная блоковым турбо кодом

На приемной стороне турбо декодер выстраивает из пришедшего «мягкого» кодового слова структуру данных, аналогичную представленной на рисунке 5.3. Далее турбо декодирование осуществляется с использованием алгоритма Чейза [1,26,30,81] в несколько итераций, каждая из которых состоит из двух фаз.

Первая фаза итерации турбо декодирования заключается в обработке каждой строки структуры. Результатом обработки являются поправки к «мягким» априорным решениям демодулятора по каждому символу, входящему в обрабатываемую строку. На входе алгоритма обработки используются апостериорные надежности символов, рассчитанные как суммы «мягких» априорных решений демодулятора и соответствующих поправок, полученных на второй фазе предыдущей итерации турбо декодирования.

Вторая фаза итерации турбо декодирования заключается в обработке каждого столбца структуры. Результатом обработки являются поправки к «мягким» априорным решениям демодулятора по каждому символу, входящему в обрабатываемый столбец. На входе алгоритма обработки используются апостериорные надежности символов, рассчитанные как суммы «мягких» априорных решений демодулятора и соответствующих поправок, полученных на первой фазе текущей итерации турбо декодирования.

Выходом алгоритма турбо декодирования на текущей итерации являются апостериорные надежности символов, рассчитанные как суммы «мягких» априорных решений демодулятора и соответствующих поправок, рассчитанных на второй фазе текущей итерации турбо декодирования.

Процедура обработки строки (для столбца аналогично) описывается ниже. Для описания введены следующие обозначения из [81]:

$R = (r_1, r_2, \dots r_n)$ - априорные «мягкие» решения демодулятора по каждому символу, входящему в рассматриваемую строку, где n – длина строки;

$R' = (r_1', r_2', \dots r_n')$ - «мягкий» вход алгоритма обработки строки;

$$P = \begin{pmatrix} p_{11}, p_{12}, \dots p_{1n} \\ p_{21}, p_{22}, \dots p_{2n} \\ \dots \\ p_{2^T 1}, p_{2^T 2}, \dots p_{2^T n} \end{pmatrix}$$ - ансамбль порожденных гипотез;

$D = (d_1, d_2, \dots d_n)$ - гипотеза из ансамбля P, имеющая наименьшую с R' метрику;

$K_i = (k_1, k_2, \dots k_n)$ - гипотеза «конкурент» из ансамбля P для i-ого символа, имеющая наименьшую с R' метрику, причем $d_i \neq k_i$.

На первой фазе первой итерации принимается $R' = R$.

Шаг 1. Определить **T** позиций (длину списка гипотез), на которых абсолютные значения апостериорных надежностей символов $R^{'}$ наименьшие. В [26] число позиций **T** определяется исходя из кодового расстояния используемого кода. В [81] **T** является константой и не зависит от кодового расстояния. В общем случае подбор числа **T** влияет на помехоустойчивость и осуществляется исходя из компромисса между корректирующей способностью турбо кода и сложностью декодирования.

Шаг 2. На всех возможных комбинациях позиций, определенных на шаге 1, проинвертировать «жесткие» символы, полученные по «мягкому» входу $R^{'}$ алгоритма обработки строки. Такая процедура порождает ансамбль гипотез *P*, число которых равно 2^T.

Шаг 3. Декодировать каждую из 2^T гипотез «жестким» декодером. В данном случае декодером Хэмминга.

Шаг 4. Рассчитать для каждой из декодированных гипотез метрику *M* следующим образом:

$$M_j = \sum_{i \in 1\ldots n} \left(r_i^{'} - p_{ji} \right)^2, \tag{5.1}$$

где $j \in 1\ldots 2^T$ - номер гипотезы.

Перед расчетом метрик необходимо заменить в ансамбле *P* значения «0» и «1» на «1» и «-1», соответственно.

Шаг 5. Выбрать из ансамбля *P* гипотезу *D*, имеющую наименьшее значение M.

Шаг 6. Для каждого символа *i* выбрать из ансамбля *P* гипотезу «конкурента» K_i, имеющую наименьшее значение M при условии $d_i \neq k_i$. Возможен случай, при котором для символа *i* гипотеза «конкурент» может не существовать.

Шаг 7. Рассчитать поправки к априорным «мягким» решениям демодулятора.

При наличии гипотезы «конкурента» поправка к соответствующему символу с номером *i* рассчитывается следующим образом:

$$w_i = \frac{\left(R^{'} - K_i\right)^2 - \left(R^{'} - D\right)^2}{4} \cdot d_i . \tag{5.2}$$

При отсутствии гипотезы «конкурента» поправка рассчитывается на основе некоторого функционала от рассматриваемой на входе алгоритма обработки строки надежности $r_i^{'}$:

$$w_i = \left| r_i^{'} \right| \cdot d_i \cdot \beta \,, \tag{5.3}$$

где β - некоторый коэффициент, значение которого зависит от номера текущей итерации.

Шаг 8. Сформировать «мягкий» вход для следующей фазы турбо декодирования по каждому символу *i* следующим образом:

$$r_i^{'} = r_i + w_i \cdot \alpha \,, \tag{5.4}$$

где w_i - поправка к *i*-ому априорному «мягкому» решению демодулятора;

α - некоторый коэффициент, значение которого зависит от номера текущей итерации.

В случае если текущая фаза декодирования вторая, величина, рассчитанная по (5.4), является «мягким» выходом алгоритма турбо декодирования на текущей итерации. При выполнении всех проверок на четность после выполнения второй фазы итеративный процесс турбо декодирования завершается. В противном случае начинается новая итерация турбо декодирования.

Ниже, в соответствии с [30,81], приводятся значения коэффициентов α и β для четырех итераций (всего восемь величин для обработки по строкам и столбцам):

$$\alpha = \begin{pmatrix} 0 & 0.2 & 0.3 & 0.5 & 0.7 & 0.9 & 1 & 1 \end{pmatrix}, \tag{5.5}$$

$$\beta = \begin{pmatrix} 0.2 & 0.4 & 0.6 & 0.8 & 1 & 1 & 1 & 1 \end{pmatrix}. \tag{5.6}$$

5.3 Сравнение характеристик декодирования

Для моделирования работы турбо кодека использовалась модель, описанная в главе 3. На рисунке 5.4 демонстрируются зависимости BER для блокового турбо кода при различном значении длины списка гипотез **T**. Увеличение длины списка гипотез **T** приводит к повышению помехоустойчивости. Результаты моделирования BER, полученные в данной работе для рассматриваемого турбо кода при **T**=4, соответствуют приведенным в [81].

Следует подчеркнуть, что повышение корректирующей способности стандартного итеративного алгоритма декодирования с использованием алгоритма Чейза может быть достигнуто подбором коэффициентов α и β для конкретной структуры кода и конкретного номера итерации. К примеру, кодек фирмы AHA демонстрирует [8] вероятность битвой ошибки

10^{-5} при битовом отношении сигнал/шум ~2.95 дБ, что в среднем на ~0.25 дБ лучше, чем при декодировании с использованием коэффициентов (5.5) и (5.6).

Помимо этого, на рисунке 5.4 приводится сравнение помехоустойчивости LDPC кода, декодируемого по алгоритму минимума суммы «Min-sum normalized», и различных реализаций турбо кодека. LDPC код демонстрирует лучшую исправляющую способность, чем блоковый турбо код, опережая реализацию турбо кодека фирмы AHA на ~0.8 дБ по уровню вероятности битовой ошибки 10^{-5}.

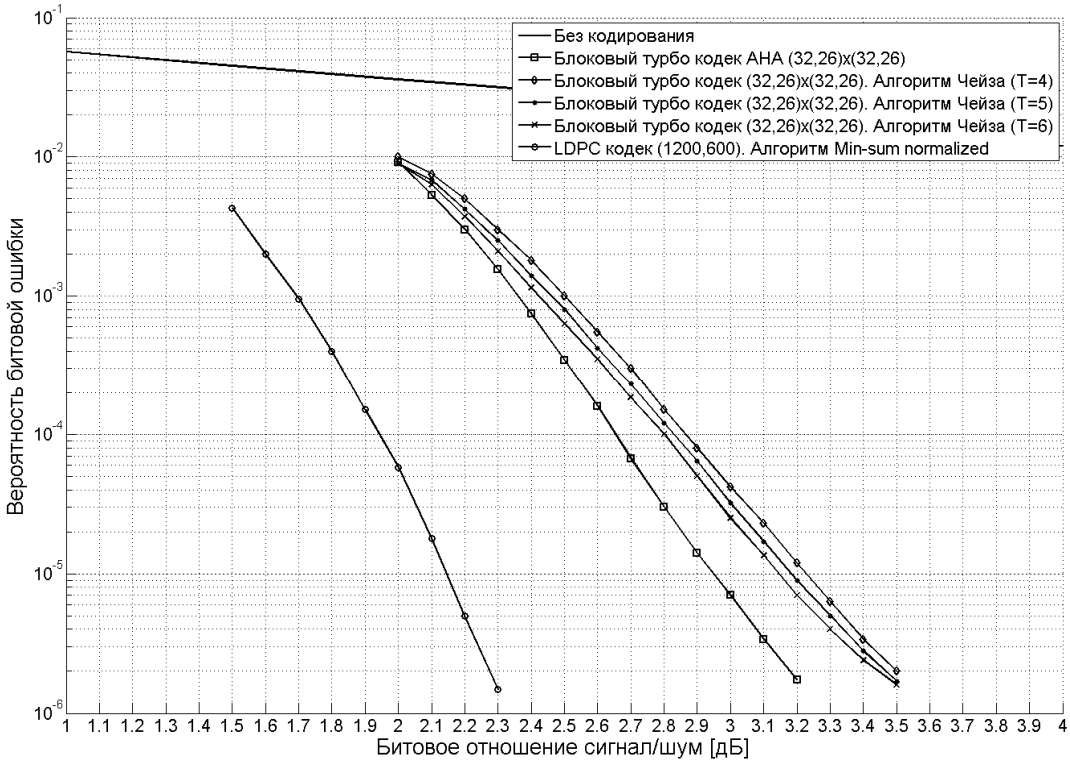

Рисунок 5.4 – Сравнение BER турбо кодека и LDPC кодека

Моделирование показывает, что в среднем при декодировании кодов с малой плотностью проверок на четность используется больше итераций, чем при декодировании турбо кодов. Зависимости среднего числа итераций приведены на рисунке 5.5. Обе характеристики получены для полной достоверности на выборке в 10000 слов. Например, оба кода обеспечивают на заданной выборке нулевую вероятность ошибки при битовом отношении сигнал/шум 3.6 дБ, однако LDPC декодер использует в среднем на 1.5 итерации больше, чем турбо декодер.

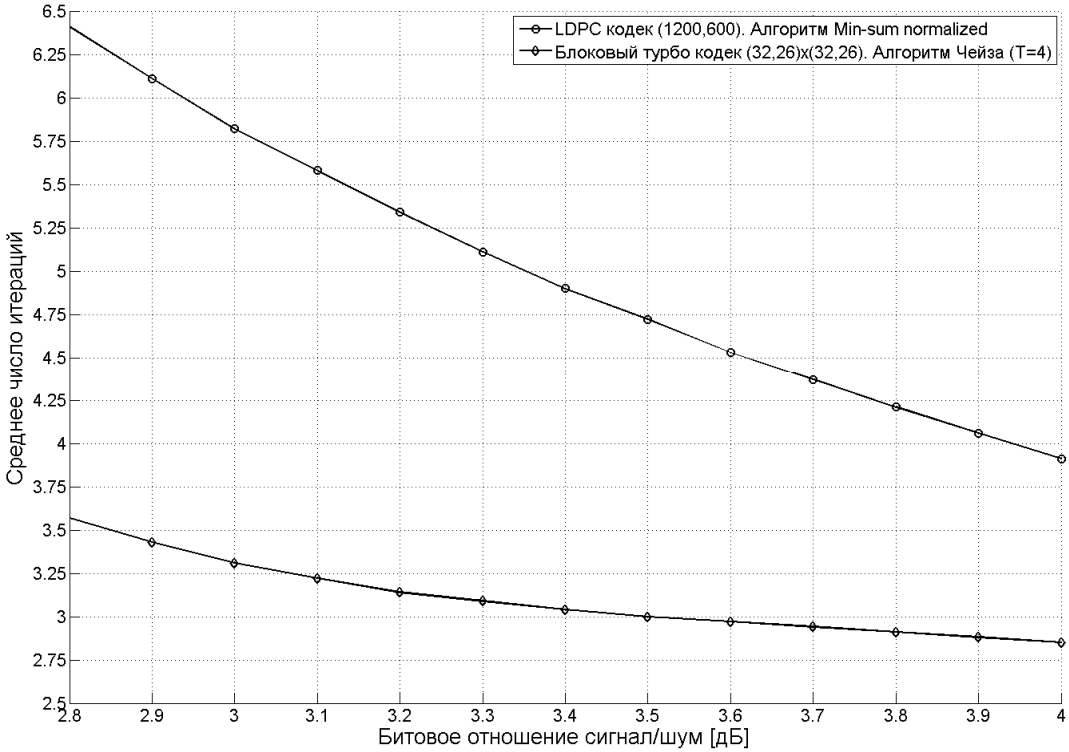

Рисунок 5.5 – Сравнение среднего числа итераций при декодировании турбо кода и LDPC кода

5.4 Сравнение вычислительной сложности декодирования

В данном подразделе проведена оценка вычислительной сложности одной итерации декодирования блокового турбо кода (32,26)x(32,26), декодируемого по алгоритму Чейза, и сравнение полученной вычислительной сложности с вычислительной сложностью итерации декодирования LDPC кода, декодируемого по алгоритму минимума суммы «Min-sum normalized».

Работу итеративного алгоритма декодирования блокового турбо кода можно разделить на несколько этапов, представленных на рисунке 5.6. Подсчет элементарных операций будет вестись для этапов 1-4, выполняемых итеративно.

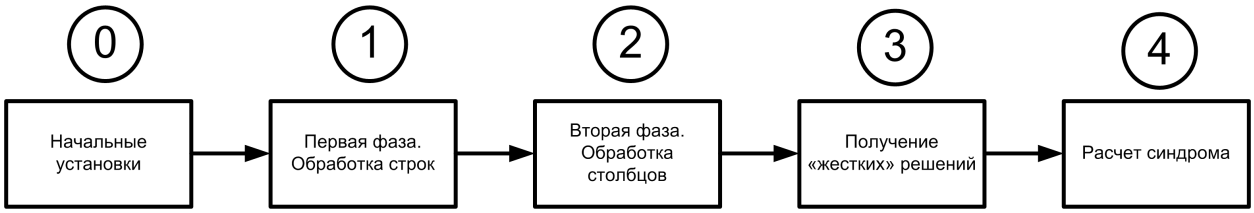

Рисунок 5.6 – Этапы декодирования блокового турбо кода

Сложность этапов 1 и 2 одинакова и в случае рассматриваемого турбо кода сводится к 32-кратному повторению процедуры обработки строки, описанной в пункте 5.2. Расчет сложности ведется для длины списка гипотез при **T=4.**

Шаг 1. Перед поиском четырех наименее надежных символов в рассматриваемой строке ко всем значениям надежностей символов применяется операция взятия модуля числа. Всего $N_{|a|} = 32$ операции взятия модуля числа.

Затем происходит поиск 4 наименее надежных символов. Вначале посредствам $N_{/} = 31$ сравнения определяется наименее надежный символ. Затем посредствам $N_{/} = 62$ сравнений определяется второй с конца по надежности символ (31 сравнение надежностей между собой и 31 проверка на то, что найденный символ не является наименьшим по надежности). Для поиска третьего и четвертого символов с конца потребуется $N_{/} = 93$ и $N_{/} = 124$ операции сравнения, соответственно.

Всего на данном шаге обработки строки используется $N_{|a|} = 32$ операции взятия модуля числа и $N_{/} = 310$ операций сравнения.

Шаг 2. На данном шаге создаётся 16 гипотез. Для их создания изначально нужно преобразовать «мягкие» решения на входе алгоритма обработки строки в «жесткие» решения путем $N_{/} = 32$ сравнений надежностей с нулём. Далее создаются 16 гипотез путем $N_{\oplus} = 64$ прибавлений по модулю 2 определенных значений ко всем возможным комбинациям позиций, содержащих наименее надежные символы.

Всего на данном шаге обработки строки используется $N_{/} = 32$ операции сравнения и $N_{\oplus} = 64$ операции сложения по модулю 2.

Шаг 3. На данном шаге происходит декодирование сформированных гипотез. 31 символ перемножается по правилам матричного умножения на транспонированную проверочную матрицу кода Хэмминга из 5 строк и 31 столбца. Для этого используется $N_{\times} = 155$ умножений и $N_{\oplus} = 150$ сложений по модулю 2. Получившийся синдром преобразуется в число в десятичной системе счисления путем $N_{\times} = 5$ умножений и $N_{+} = 4$ сложений. По полученному значению происходит поиск номера разряда, в котором произошла ошибка. Для этого будет использовано не более $N_{/} = 31$ операции сравнения. По найденному номеру разряда инвертируется жесткий символ путем $N_{\oplus} = 1$ сложения по модулю 2. Далее на основе декодированных 31 символов путем $N_{\oplus} = 30$ сложений по модулю 2 вычисляется последний символ гипотезы. Процедура повторяется для всех 16 гипотез.

Всего на данном шаге обработки строки используется $N_{\times} = 2560$ операций умножения, $N_{\oplus} = 2896$ операций сложения по модулю 2, $N_{+} = 64$ сложения, $N_{/} = 496$ сравнений.

Шаг 4. Перед расчетом одной метрики значения элементов гипотезы необходимо заменить по правилу «0» на «1», а «1» на «-1». Для этого используется $N_{/} = 32$ сравнения. Для расчета метрики используется $N_{+} = 32$ вычитания и $N_{+} = 31$ сложение, а также $N_{\times} = 32$ умножения полученного результата на самого себя (возведение в квадрат).

Всего на данном шаге обработки строки используется $N_{\times} = 512$ операций умножения, $N_{+} = 1008$ сложений, $N_{/} = 512$ сравнений.

Шаг 5. Гипотеза с наименьшим значением метрики находится путем $N_{/} = 15$ сравнений метрик гипотез друг с другом.

Всего на данном шаге обработки строки используется $N_{/} = 15$ сравнений.

Шаг 6. Гипотеза «конкурент» для *i*-ого символа находится путем $N_{/} = 15$ сравнений метрик гипотез друг с другом и $N_{/} = 15$ проверок условия $d_i \neq k_i$.

Всего на данном шаге обработки строки используется $N_{/} = 960$ операций сравнения для поиска 32 гипотез «конкурентов».

Шаг 7. Оценка сложности ведется для расчета поправок по (5.2). Значения в скобках уже рассчитаны на шаге 4. Для расчета одной поправки используется $N_{+} = 1$ вычитание и $N_{\times} = 2$ умножения (операция деления на 4 заменена на умножение на 0.25).

Всего на данном шаге обработки строки используется $N_{+} = 32$ вычитания и $N_{\times} = 64$ умножения.

Шаг 8. Для формирования одного элемента «мягкого» входа используется $N_{\times} = 1$ умножение и $N_{+} = 1$ сложение.

Всего на данном шаге обработки строки используется $N_{+} = 32$ сложения и $N_{\times} = 32$ умножения.

После обработки всех строк и столбцов необходимо получить 64 синдрома, каждый из которых рассчитывается путем перемножения 32 символов на проверочную матрицу кода Хэмминга из 6 строк и 32 столбцов. Всего для расчета синдромов необходимо $N_{\times} = 12288$ умножений и $N_{\oplus} = 11904$ сложений по модулю 2. Перед расчетом синдрома необходимо получить «жесткие» апостериорные решения посредствам $N_{/} = 1024$ сравнений. $N_{/} = 1$ сравнение требуется для проверки выхода за максимальное число итераций.

Сравнение сложности итерации декодирования LDPC кода и турбо кода приводится в таблице 5.1. Моделирование на ПК показало, что проведение итерации LDPC декодирования требует в 3 раза меньше времени, чем проведение итерации блокового турбо декодирования.

Таблица 5.1 – Сравнение сложности декодирования

Операция	Количество на итерацию декодирования	
	LDPC («Min-sum normalized»)	Блоковый турбо код (Алгоритм Чейза)
Сложение	37024	72704
Умножение	38706	215040
Сравнение	64159	149825
Взятие модуля	33888	2048
Сложение по модулю 2	4218	201344

Следует подчеркнуть, что процедура декодирования блокового турбо кода может быть оптимизирована с точки зрения вычислений, как это показано в [1]. Однако в случае оптимизированного алгоритма декодирования турбо кода его сложность следует сравнивать со сложностью оптимизированного алгоритма декодирования LDPC. Соответствующая оптимизация проведена в пункте 2.5.

Анализ соотношений между сложностью сравниваемых кодов на итерацию декодирования и их средним используемым числом итераций показывает, что в конкретном случае выгоднее проделывать больше итераций декодирования LDPC с меньшей сложностью, чем осуществлять меньшее число итераций турбо декодирования, но с большей сложностью.

5.5 Выводы по главе

1. Моделирование работы турбо кодека на имитационной модели выявило зависимость вероятности ошибки на выходе декодера от числа генерируемых алгоритмом Чейза гипотез.

2. По результатам сравнения для конкретно взятых LDPC кода и турбо кода по критериям сложности реализации и эффективности коррекции ошибок предпочтение отдано LDPC коду.

Основные результаты и выводы по работе

Проведенный в данной работе анализ существующих алгоритмов декодирования LDPC показал их тесную взаимосвязь друг с другом в рамках отдельных семейств, а также принципиальные отличия семейств друг от друга.

Полученные в данной работе аналитические соотношения позволяют оценивать сложность итерации декодирования LDPC для различных алгоритмов.

Выявленные характерные особенности в работе алгоритмов декодирования в части расчета поправок позволили разработать модификации с целью повышения вычислительной эффективности декодирования.

Разработанная имитационная модель позволила проводить исследование вероятностных характеристик алгоритмов декодирования и осуществлять отладку программных реализаций декодера для последующего внедрения.

Разработанный способ идентификации инверсии битового потока на входе декодера позволяет определять инверсию за счет внутренних ресурсов LDPC декодера.

Основные результаты можно сформулировать следующим образом:

1. Разработана методика выбора алгоритма декодирования LDPC кодов.
2. Предложена методика представления разряженной проверочной матрицы кода с малой плотностью проверок на четность.
3. Разработаны модификации алгоритмов декодирования, позволяющие без потери в качестве декодирования уменьшить число операций, выполняемых декодером на одну итерацию декодирования.
4. Предложен способ идентификации инверсии битового потока на входе LDPC декодера.

Список используемой литературы

1. Архипкин А.В., Разработка алгоритмов кодирования и декодирования для телекоммуникационных систем радиосвязи с ортогональными поднесущими, диссертация на соискание ученой степени кандидата технических наук, Москва, 2008.

2. Башкиров А.В., Науменко Ю.С. Современные методы декодирования недвоичных кодов с малой плотностью проверок на четность: краткий обзор и сравнение // Современные проблемы радиоэлектроники: труды всероссийской научно-технической конференции, Красноярск, 2013, с.414-416.

3. Воробьев К. А., Методы построения и декодирования недвоичных низкоплотностных кодов // Теория и практика системного анализа. 2010. Т. II. С. 96–102.

4. Дворкович А.В., Лихобабин Е.А., Использование квазиоптимальных алгоритмов декодирования LDPC кодов в системе цифрового телевизионного вещания стандарта DVB-T2// Цифровая обработка сигналов и ее применение, 12-ая международная конференция – М.:2010, с.25-27.

5. Зигангиров Д.К., Зигангиров К.Ш., мл. Костелло Д., Partially-regular LDPC codes with linear encoding complexity and improved thresholds, IEEE International Symposium on Information Theory - Proceedings Сер. 2011 IEEE International Symposium on Information Theory Proceedings, 2011, pp. 528-532.

6. Зигангиров К. Ш. , А. Е. Пусане, Д. К. Зигангиров, Д. Дж. Костелло, О корректирующей способности кодов с малой плотностью проверок на четность, Пробл. передачи информ.,2008, № 44, с. 50-62.

7. Золотарев В.В., Реальный энергетический выигрыш кодирования для спутниковых каналов // 4-я Международная конференция «Спутниковая связь - 2000», Т.2, с. 20-25.

8. Золотарев В.В., Овечкин Г.В., Обзор исследований и разработок методов помехоустойчивого кодирования, 2005 г.

9. Кирьянов И.А., Моделирование работы LDPC-декодера по алгоритму с распространением доверия по надежностям, Информационные технологии в проектировании и производстве, изд. ФГУП ВИМИ, №4, 2012 г., стр. 57-60.

10. Кирьянов И.А., Исследование статистических характеристик декодирования низкоплотностных кодов, Информационно-измерительные и управляющие системы, изд. Радиотехника, №10, 2012 г., стр. 20-25.

11. Кирьянов И.А., Важенин Н.А., Оценка статистических характеристик функционирования LDPC-декодера на имитационной модели, Электронный журнал «Труды МАИ», изд. МАИ, №59, 2012 г.

12. Кирьянов И.А., Субоптимальное декодирование кодов с малой плотностью проверок на четность, Электромагнитные волны и электронные системы, изд. Радиотехника, № 5, 2014 г., стр.47-51.

13. Кирьянов И.А., Важенин Н.А., Особенности программной реализации и характеристики декодера низкоплотностных кодов в среде MATLAB/SIMULINK, Вестник Московского авиационного института, изд. МАИ, №2, 2014 г., стр. 104-113.

14. Кирьянов И.А., Декодирование кодов с малой плотностью проверок на четность по алгоритму «Belief propagation» с аппроксимацией, Вестник воздушно-космической обороны, изд. ОАО «ГСКБ «Алмаз-Антей», №2, М.: - 2014 г., стр.40-47.

15. Кирьянов И.А., Программа оценки вычислительной сложности декодирования кодов с малой плотностью проверок на четность, Свидетельство о государственной регистрации программы для ЭВМ №2014615590 от 29.05.14 г.

16. Кирьянов И.А., Исследование статистических характеристик декодирования низкоплотностных кодов с использованием алгоритма с распространением доверия по надёжностям, Сборник тезисов докладов Московской молодежной научно-практической конференции «Инновации в авиации и космонавтике - 2012», изд. ООО «Принт-салон», Санкт-Петербург: -2012 г., стр. 96-97.

17. Кирьянов И.А., Моделирование работы LDPC – декодера по алгоритму с распространением доверия по надёжностям, Сборник докладов 1-ой межвузовской студенческой научно-технической конференции «Современные состояния и перспективы развития сложных радиоэлектронных систем», изд. ОАО «ГСКБ Алмаз-Антей», М.: - 2012 г., стр. 16-22.

18. Кирьянов И.А., Программная реализация алгоритма декодирования «Belief propagation» LDPC кода на языке C++, Сборник тезисов докладов Московской молодежной научно-практической конференции «Инновации в авиации и космонавтике - 2013», изд. ООО «Принт-салон», Санкт-Петербург: -2013 г., стр. 234-235.

19. Кирьянов И.А., Анализ методов повышения эффективности вычислительной реализации алгоритмов декодирования LDPC – кодов, Сборник тезисов докладов 12-ой Международной конференции «Авиация и космонавтика - 2013», изд. «Известия», М.: - 2013 г., стр. 476 – 477.

20. Кирьянов И.А., Мажоритарное декодирование кодов с малой плотностью проверок на четность, Сборник тезисов докладов Московской молодежной научно-практической конференции «Инновации в авиации и космонавтике - 2014», изд. ООО «Принт-салон», Санкт - Петербург: - 2014 г., стр. 160-161.

21. Кирьянов И.А., Мажоритарное декодирование кодов с малой плотностью проверок на четность, Электромагнитные волны и электронные системы, изд. Радиотехника, № 12, 2014 г., стр.9-14.

22. Кирьянов И.А., Повышение вычислительной эффективности декодирования кодов с малой плотностью проверок на чётность, Вестник воздушно-космической обороны, изд. ОАО «ГСКБ «Алмаз-Антей», №4, М.: - 2014 г., стр. 120-124.

23. Кирьянов И.А., Сравнение перспективных техник помехоустойчивого кодирования информации, Электромагнитные волны и электронные системы, изд. Радиотехника, № 1, 2015 г., стр.26-34.

24. Кирьянов И.А., Применение помехоустойчивого кодирования информации в глобальной навигационной спутниковой системе GPS, Вестник воздушно-космической обороны, изд. ОАО «ГСКБ «Алмаз-Антей», №4, М.: - 2014 г., стр. 113-119.

25. Кирьянов И.А., Декодирование низкоплотностных кодов с использованием линейной аппроксимации гиперболических функций, Сборник тезисов докладов 13-ой Международной конференции «Авиация и космонавтика – 2014», изд. ООО «Принт-салон», Санкт-Петербург: - 2014 г., стр. 395.

26. Кларк Дж., Д. Кейн, Кодирование с исправлением ошибок в системах цифровой связи, Москва, изд. Радио и связь, 391 с.

27. Кравченко А.Н., Снижение сложности декодирования низкоплотностного кода // Цифровая обработка сигналов. 2010, № 2, с. 35-41.

28. Крук Е.А., Вопросы защиты и передачи информации. Сборник статей, Санкт-Петербург, 2006.

29. Лихобабин Е.А., Упрощенные алгоритмы декодирования кодов с низкой плотностью проверок на четность, основанные на алгоритме распространения доверия // Цифровая обработка сигналов, 2013, №3, с. 54-60.

30. Морелос-Сарагоса Р. Искусство помехоустойчивого кодирования. Методы, алгоритмы, применение, - М.: ТЕХНОСФЕРА, 2005, 320 с.

31. Овечкин Г.В., Применение Min-sum алгоритма для декодирования блоковых самоортагональных кодов // Межвуз. сб. науч. тр. «Математическое и программное обеспечение вычислительных систем» – Москва, Горячая линия – Телеком, 2010, С. 99–105.

32. Овчинников А.А., Обработка информации при передаче LDPC-кодами по дискретным и полунепрерывным каналам, - М.: РГБ, 2005. – (Из фондов Российской государственной библиотеки)

33. Письменный Д.Т., Конспект лекций по высшей математике, Ч.2. – 4-е изд. – М.: Айрис-пресс, 2006. -256 с.

34. Проскурин А.А., Модифицированный алгоритм итеративного декодирования низкоплотностных кодов, учитывающий свойства и структуру кода и обеспечивающий сокращение временной и вычислительной сложности. URL: http://naukovedenie.ru/PDF/84TVN214.pdf (дата обращения 26.01.15).

35. Скляр Б., Цифровая связь, - М.: Вильямс, 2003.

36. Советов Б.Я., Яковлев С.А., Моделирование систем: Учеб. Для вузов – 3-е изд., перераб. и доп. –М: Высш. Шк., 2001. – 343 с.

37. Солтанов А.Г., Схемы декодирования и оценка эффективности LDPC-кодов. Применение, преимущества и перспективы развития // Безопасность информационных технологий, 2010, М.: №2, с. 61-67.

38. Солтанов А.Г., Защита информации от угроз нарушения целостности в высокоскоростных каналах передачи данных, - М.: РГБ, 2011. – (Из фондов Российской государственной библиотеки)

39. Шридхаран А. , М. Лентмайер, Д. В. Трухачев, Д. Дж. Костелло, К. Ш. Зигангиров, О минимальном расстоянии низкоплотностных кодов с проверочными матрицами, составленными из перестановочных матриц, Пробл. передачи информ.,2005, № 41,с. 39-52.

40. «BCH Codes», URL: http://www.ni.com/white-paper/14765/en/ (дата обращения: 26.01.15).

41. Bhattad Kapil, LDPC Code Design for Min-Sum Based Decoding, URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.94.6066&rep=rep1&type=pdf, (дата обращения 26.01.15).

42. Burshtein D., Krivelevich M., Litsyn S., Miller G., Upper bounds on the rate of LDPC codes, IEEE transactions on information theory, 2002, vol.48, no.9, pp.2473-2449.

43. Burshtein David, On the Error Correction of Regular LDPC Codes Using the Flipping Algorithm, IEEE Trans on Inf Theory, 2008, vol. 54, no. 2, pp.517-530.

44. Campello Jorge, Dharmendra S. Modha, Sridhar Rajagopalan, Designing LDPC Codes Using Bit-Flipping, Proceedings of the IEEE ICC 2001.

45. Chen J., Dholaki A., Elftheriou E., Fossorier M., Near optimal reduced-complexity decoding algorithms for LDPC codes. In IEEE Intern. Symposium on Inf. Theory, 2002, p. 455.

46. Chen J., Marc P. C. Fossorier, Density Evolution for Two Improved BP-Based Decoding Algorithms of LDPC Codes, IEEE communications letters, 2002, vol. 6, no. 5, pp. 208-210.

47. Chen J., Fossorier M., Near Optimum Universal Belief Propagation Based Decoding of Low-Density Parity Check Codes, IEEE transactions on communications, 2002, vol. 50, no.3, pp.406 – 414.

48. Chen J., Tanner R. M., Jones C., Li Y., Improved Min – sum Decoding Algorithms for Irregular LDPC Codes, Proceedings International Symposium on Information Theory, 2005, pp.449-453.

49. Chen Wen-Yao, Chung-Chin Lu, On Error Correction Capability of Bit-Flipping Algorithm for LDPC Codes, IEEE International Symposium on Information Theory Proceedings, 2011, pp. 1288-1291.

50. Conde-Canencia L., Ghouwayel A., Boutillon E., Complexity Comparison of Non-Binary LDPC Decoders, ICT-MobileSummit 2009 Conference Proceedings, 2009, pp.1-8.

51. David J.C. MacKay, Encyclopedia of sparse graph codes. URL: http://www.inference.phy.cam.ac.uk/mackay/codes/data.html (дата обращения 26.01.15).

52. Declercq D., Fossorier M., Extended MinSum Algorithm for Decoding LDPC Codes over GF(q), Proceedings. International Symposium on Information Theory, 2005, pp.464-468.

53. Dong, G., Y. Li, N. Xie, T. Zhang, and H. Liu, Candidate bit based bit-flipping decoding algorithm for LDPC codes, Proceedings of the 2009 IEEE international conference on Information Theory, 2009, pp. 2166–2168.

54. El Alami R., C. B. Gueye, M. Mrabti, Bit Flipping – Sum Product Algorithm for Regular LDPC Codes, IEEE International Symposium on Information Theory Proceedings, 2010, pp. 1-4.

55. Eleftheriou E., T. Mittelholzer and A. Dholakia, Reduced-complexity decoding algorithm for low-density parity-check codes, IEE Electronics Letters, 2001, vol. 37, pp. 102-104.

56. El Hassani, S., Hamon, M. Penard, P., A comparison study of binary and non-binary LDPC codes decoding, International Conference on Software Telecommunications and Computer Networks, 2010, pp. 355-359.

57. «Enhancing the future of Civil GPS»,
URL: http://www.eew.caltech.edu/docs/Betz_etal_L1C-Inside-GNSS_igm_042-049.pdf
(дата обращения: 26.01.15).

58. fec.ldpcdec.
URL: http://www.kxcad.net/cae_MATLAB/toolbox/comm/ug/fec.ldpcdec.html,
(дата обращения 11.01.15).

59. Forney G., Codes on Graphs: Normal realizations, IEEE transactions on information theory, 2001, vol.47, no.2, pp.520-548.

60. Gallager R.G., Low-Density Party-Check Codes, IRE Trans Info Theory, 1962, vol. 8, no. 1, pp. 21-28.

61. Gallager R.G., Low-Density Party-Check Codes, Monograph, M.I.T. Press, 1963.

62. Guo F. and L. Hanzo, Reliability ratio based weighted bit-flipping decoding for low-density parity-check codes, Electron. Lett., 2004, vol. 40, pp. 1356-1358.

63. Han J.H., M.H.Sunwoo, Simplified Sum-product algorithm using piecewise linear function approximation for low complexity LDPC decoding, Proceedings of the 3rd International Conference on Ubiquitous Information Management and Communication, 2009, pp. 302 – 308.

64. Hgenauer J., Offer E., Papke L., Iterative decoding of Binary Block and Convolutional Codes, IEEE Trans on Inf Theory, 1996, vol. 42, no. 2, pp. 429-430.

65. Hu X-Y., Eleftherious E., Arnold D-M. Efficient implementation of the sumproduct algorithm for decoding LDPC codes, Proc. 2001 IEEE GlobeCom Conf., 2001, pp. 1036–1036E.

66. Huangshan, P. R., A Kind of Low Complexity LDPC Decoder, Proceedings of the Second Symposium International Computer Science and Computational Technology, 2009, pp. 102-105.

67. «Information-Dispersion-Entropy-Based Blind Recognition of Binary BCH Codes in Soft Decision Situations», URL: http://www.mdpi.com/1099-4300/15/5/1705 (дата обращения: 19.01.15).

68. Ismail M., I. Ahmed, J. Coon, S. Armour, T. Kocak, J. McGeehan, Low Latency Low Power Bit Flipping Algorithms For LDPC Decoding, IEEE 21st International Symposium on Personal Indoor and Mobile Radio Communications, 2010, pp. 278-282.

69. Jiang M., C. Zhao, Z. Shi, Yu Chen, An Improvement on the Modified Weighted Bit Flipping Decoding Algorithm for LDPC Codes, IEEE communications letters, 2005, vol. 9, no. 9, pp.814-816.

70. Kalaycioglu A., O. Ureten, H.Gokhan, A second order approximation to reduce the complexity of LDPC decoders based on Gallager's approach, Turk J Elec Eng & Comp Sci, 2010, Vol.18, No.6, pp.1053 – 1058.

71. Kim, N., H. Park, Modified UMP-BP decoding algorithm based on mean square error, Electronics Letters, 2004, vol. 40, no. 13, pp. 816- 817.

72. Kou Y., S. Lin, and M. Fossorier, Low-density parity-check codes based on finite geometries: a rediscovery and new results, IEEE Trans. Inf. Theory, 2001, vol. 47, pp. 2711-2736.

73. Lee C.-H. and W. Wolf, Implementation-efficient reliability ratio based weighted bit-fipping decoding for LDPC codes, Electron. Lett., 2005, vol. 41, pp. 755- 757.

74. Marc P.C. Fossorier, Miodrag Mihaljević, Hideki Imai, Reduced Complexity Iterative Decoding of Low-Density Parity Check Codes Based on Belief Propagation, IEEE Transactions on communications, 1999, vol. 47, no. 5, pp. 673-679.

75. Moon T.K., Error correction coding. Mathematical Methods and Algorithms, Wiley-Interscience, 2005, 800p.

76. Ngatched T.M.N., F. Takawira, M. Bossert, A Modified Bit-Flipping Decoding Algorithm for Low-Density Parity-Check Codes, International Conference on Communications, 2007, pp. 653 – 658.

77. Nguyen Dung Viet, Bane Vasic, Michael W. Marcellin, Two-Bit Bit Flipping Decoding of LDPC Codes, Submitted to IEEE International Symposium on Information Theory, 2011.

78. Ohhashi A., T. Ohtsuki, Performance of low-density parity-check (LDPC) code with UMP BP-based algorithm and quantizer on Rayleigh fading channels, The 57th IEEE Semiannual Vehicular Technology Conference, 2003, vol.3, pp. 1881-1885.

79. Paraharalabos S., P. Sweeney, B.G. Evans, Performance evaluation of a modified sum-product decoding algorithm for LDPC codes, 2nd International Symposium on Wireless Communication Systems, 2005, pp.800 – 804.

80. Phromsa-ard T., J. Arpornsiripat, J. Wetcharungsri, P. Sangwongngam, K. Sripimanwat, P. Vanichchanunt, Improved Gradient Descent Bit Flipping algorithms for LDPC decoding, IEEE Second International Conference on Digital Information and Communication Technology and it's Applications, 2012, pp. 324 – 328.

81. Pyndtah R.M., Near-Optimum decoding of product codes block turbo codes, IEEE transactions on communication, 1998, vol.46, no.8, pp. 1003-1010.

82. Qian C., W. Lei, Z. Wang, Low complexity LDPC decoder with modified sum-product algorithm, Tsinghua science and technology, 2013, Vol. 18, No. 1, pp.57-61.

83. Richter G., Schmidt G. and Bossert M., Optimization of a Reduced-Complexity Decoding Algorithm for LDPC Codes by Density Evolution, IEEE International Conference, 2005, vol. 1, pp. 642 – 646.

84. Song Hui-Shi, P. Zhang, Very-low-complexity decoding algorithm for low-density parity-check codes, 14th IEEE Proceedings on Personal, Indoor and Mobile Communications, 2003, vol. 1, pp. 161 – 165.

85. Torres V., A. Perez-Pascual, T. Sansaloni, J. Valls, Fully-parallel LUT-based (2048,1723) LDPC code decoder for FPGA, 19th IEEE International Conference on Electronics, Circuits and Systems, 2012, pp. 408-411.

86. URL: http://neuropro.ru/memo312.shtml, (дата обращения 20.01.15).

87. Vasic B., Kurtas E.M., Coding and Signal Processing for Magnetic Recording Systems, 2004, pp. 627-645.

88. Wadayama T., K. Nakamura, M. Yagita, Gradient Descent Bit Flipping Algorithms for Decoding LDPC Codes, IEEE transactions on communication, 2010, vol. 58, no. 6, pp.1610 – 1614.

89. Wiberg N., Codes and Decoding on General Graphs, Ph.D. dissertation, 1996.

90. Wymeersch, H., Steendam, H., Moeneclaey M., Computational complexity and quantization effects of decoding algorithms for non-binary LDPC codes, IEEE International Conference on Acoustics, Speech, and Signal Processing, 2004, pp. 669 – 672.

91. Zhang F.X., F.X. Yang, M. Dong, A new modified UMP-BP decoding algorithm of quasi-cyclic LDPC codes based on oscillation estimation, Tenth international conference on wireless and optical communications networks, 2003, pp. 1-5.

92. Zhao J., F. Zarkeshvari, A. H. Banihashemi, On implementation of min-sum algorithm and its modifications for decoding low-density parity-check (LDPC) codes, IEEE transactions on communications, 2005, vol. 53, no. 4, pp. 549-554.

93. Zhang Haigang, Dongfeng Yuan, and Cheng-Xiang Wang, A Modified Weighted Bit-Flipping Algorithm for LDPC Codes, Proc. of 2nd International Conference on Wireless, Mobile and Multimedia Networks, 2008.

94. Zhou X. S., B.F. Cockburn, S. Bates, Improved Iterative Bit Flipping Decoding Algorithms for LDPC Convolutional Codes, Conference on Communications, Computers and Signal Processing, 2007, pp. 541-544.

95. Zhong Z., Y.Li, X. Chen, H. Hu, J. Wang, Modified Min-sum Decoding Algorithm for LDPC Codes Based on Classified Correction, Third International Conference on Communications and Networking in China, ChinaCom, 2008, pp. 932-936.