Міністерство освіти і науки України
Чорноморський національний університет імені Петра Могили
Факультет комп'ютерних наук
Кафедра комп'ютерної інженерії

ДОПУЩЕНО ДО ЗАХИСТУ
Завідувач кафедри, канд. техн. наук, доцент
_____ Я. М. Крайник
« __ » _____ 2022 р.

КВАЛІФІКАЦІЙНА РОБОТА

Галузь знань: 12 Інформаційні технології
Спеціальність: 123 Комп'ютерна інженерія
Тема: **Система моделювання роботи декодування LDPC-кодів з використанням бібліотек JavaScript**

Шифр: 123 – КР.1 – 405.21810524

Виконав:
студент 4 курсу, групи 405, спеціальності
123 Комп'ютерна інженерія
_____ К. М. Степанов

Керівник:
Завідувач кафедри, канд. техн. наук, доцент
_____ Я. М. Крайник

Миколаїв 2022

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Чорноморський національний університет імені Петра Могили
Факультет комп'ютерних наук
Кафедра комп'ютерної інженерії

ЗАТВЕРДЖУЮ
Завідувач кафедри,
канд. техн. наук, доцент
_____ Я. М. Крайник
« __ » _____ 2022 р.

ЗАВДАННЯ

на виконання кваліфікаційної роботи

Видано студенту групи 405 факультету комп'ютерних наук

Степанов Кирило Миколайович

(прізвище, ім'я, по батькові студента)

1. Тема кваліфікаційної роботи: Система моделювання роботи декодування LDPC-кодів з використанням бібліотек JavaScript .

Затверджена наказом по ЧНУ від «__» _____ 2022 р. No _____

2. Строк представлення кваліфікаційної роботи «____» _____ 2022 р.

3. Очікуваний результат роботи та початкові дані, якщо такі потрібні

Застосунок має моделювати та наочно зображати процес роботи декодування LDPC-кодів. _____

_____

_____

4. Перелік питань, що підлягають розробці

Виконати аналіз предметної області завадостійких кодів для передачі інформації та засобів моделювання, програмних засобів для реалізації системи моделювання. Обрати алгоритм декодування LDPC-кодів, що буду використовуватись у системі. Розробити систему з допомогою бібліотек JavaScript. Протестування систему на обраних наборах вхідних даних. Зробити аналіз результатів тестування з метою порівняння роботи різних кодів LDPC у різних ситуаціях.

5. Перелік графічних матеріалів

Матриці,

демонстрація інтерфейсів,

лістінги програмного коду

6. Завдання до спеціальної частини

Розглянути основні державні будівельні норми України, щодо вентиляції та кондиціонування, забирання зовнішнього та викид витяжного повітря, витрати припливного повітря та організація повітрообміну. Оволодіти навиками розрахунку систем повітрообміну при загально обмінній вентиляції виробничих приміщень.

7. Консультанти:

Консультант	Кафедра (організація)	Частина роботи
ст. викладач А. О. Алексєєва	кафедра екології Медичного інституту ЧНУ ім. Петра Могили	спеціальна частина з охорони праці

Керівник роботи ____ канд. тех. наук, доцент, Я. М. Крайник ___ _____

(посада, прізвище, ім'я, по батькові) (підпис)

Завдання прийнято до виконання __Степанов Кирило Миколайович _ _____

(прізвище, ім'я, по батькові студента) (підпис)

Дата видачі завдання «____» _____ 20_____ р.

КАЛЕНДАРНИЙ ПЛАН

виконання кваліфікаційної роботи

Тема: Система моделювання роботи декодування LDPC-кодів з використанням бібліотек JavaScript

№	Найменування роботи	Початок	Закінчення	Примітки
1.	Розробка та затвердження завдання на виконання КР	21.10.2021	02.11.2021	Виконано
2	Огляд літератури за темою роботи	03.11.2021	30.11.2021	Виконано
3	Складання календарного плану КР	01.12.2022	15.12.2021	Виконано
4	Аналіз предметної області	10.01.2022	23.01.1021	Виконано
5	Розробка проектних рішень	24.01.2022	10.03.2022	Виконано
6	Моделювання та конструювання додатку	10.03.2022	21.04.2022	Виконано
7	Перевірка працездатності, тестування розробленого додатку, аналіз результатів тестування	22.04.2022	22.05.2022	Виконано
8	Відгук керівника КР	23.05.2022	01.06.2022	Виконано
9	Оформлення КР та презентації	02.06.2022	09.06.2022	Виконано
10	Попередній захист	10.06.2022	11.06.2022	Виконано
11	Рецензування	14.06.2022	18.06.2022	Виконано
12	Завершення оформлення КР та презентації	18.06.2022	24.06.2022	Виконано
13	Захист кваліфікаційної роботи	27.06.2022	27.06.2022	Виконано

Розробив студент_____Степанов Кирило Миколайович_____ _____

(прізвище, ім'я, по батькові) (підпис)

«__» _____ 2022 р.

Керівник роботи ___канд. тех. наук, доцент, Я. М. Крайник____ _____

(посада, прізвище, ім'я, по батькові) (підпис)

«__» _____ 2022 р.

АНОТАЦІЯ

до кваліфікаційної роботи

«Система моделювання роботи декодування LDPC-кодів з використанням бібліотек JavaScript»

Студент: Степанов Кирило Миколайович

Керівник: канд. тех. наук, доцент Крайник Я. М.

Кваліфікаційна робота присвячена розробці системи моделювання роботи декодування LDPC-кодів. Актуальність цієї роботи обумовлена малою кількістю на ринку веб систем моделювання роботи декодування LDPC-кодів. Метою кваліфікаційної роботи є створення системи для моделювання роботи декодування LDPC-кодів.

Практичним значенням результатів дослідження та розробки БР є можливість створеним сервісом отримувати дані щодо декодування LDPC-кодів.

Пояснювальна записка кваліфікаційної роботи складається зі вступу, трьох розділів, висновків та трьох додатків.

У вступі визначається актуальність теми, сформульовані мета, об'єкт, предмет та завдання роботи.

У першому розділі знаходиться аналітичний огляд літератури та патерної інформації з декодування LDPC-кодів.

В другому розділі відбувається огляд методів та алгоритмів декодування LDPC. Схеми декодування та оцінка ефективності LDPC–кодів.

В третьому розділі відбувається реалізація системи декодування та оцінка ефективності LDPC–кодів.. Також проводиться тестування та демонстрація робочого процессу.

В додатку А наводиться код для react складової.

В додатку Б наводиться код для рендеру клієнтського застосунку.

В додатку В наводиться результат роботи додатку.

Кваліфікаційна робота складається з 94 сторінок, 35 рисунки, 3 додатків та 39 використаних джерел посилання.

Ключові слова: декодування, бібліотеки, JavaScript, фреймворк, Next.js, React.js

ABSTRACT

of the Bachelor's Thesis

«LDPC code decoding simulation system using JavaScript libraries»

Student: Stepanov Kyrylo Mykolaevich

Scientific Advisor: Ph.D. in Computer Systems and Components, Associate Professor Krainyk Y. M.

Qualification work is devoted to the development of a system for modeling the work of decoding LDPC-codes. The relevance of this work is due to the small number on the market of web systems for modeling the work of decoding LDPC-codes. The purpose of the qualification work is to create a system for modeling the decoding of LDPC-codes.

The practical significance of the results of research and development of BR is the ability of the created service to obtain data on the decoding of LDPC-codes.

The explanatory note of the qualification work consists of an introduction, three sections, conclusions and three appendices.

The introduction determines the relevance of the topic, formulates the purpose, object, subject and objectives of the work.

The first section provides an analytical review of the literature and pattern information on decoding LDPC codes.

The second section provides an overview of LDPC decoding methods and algorithms. Decoding schemes and evaluation of LDPC codes efficiency.

The third section implements the decoding system and evaluates the effectiveness of LDPC codes. Testing and demonstration of the workflow are also performed.

Annex A provides the code for the react component.

Appendix B provides the code for rendering the client application.

Annex B provides the result of the application.The qualification work consists of 94 pages, 35 figures, 3 appendices and 39 used reference sources.

Keywords: decoding, libraries, JavaScript, framework, Next.js, React.js

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

FEC – Forward Error Correction

IEEE – Institute of Electrical and Electronics Engineers

LDPC – Low-density parity-check

LLR – Log-likelihood ratio

OFDM – Orthogonal Frequency-Division Multiplexing

МОН – Міністерство освіти і науки

ПЛІС – Програмована логічна інтегральна схема

ЧНУ – Чорноморський національний університет

FEC – Forward Error Correction

IEEE – Institute of Electrical and Electronics Engineers

LDPC – Low-density parity-check

LLR – Log-likelihood ratio

OFDM – Orthogonal Frequency-Division Multiplexing

ВСТУП

У сучасному світі набули широкого поширення і продовжують швидко розвиватися області, пов'язані з обробкою та передачею даних - локальні дротяні мережі, мобільний зв'язок, бездротові мережі, пристрої зберігання даних. Важливим завданням є підвищення ефективності існуючих методів передачі, у тому числі, розробка алгоритмів, що дозволяють підвищити надійність інформації, що передається. Обробка інформації за допомогою процедур перешкодостійкого кодування дозволяє забезпечити необхідну ймовірність помилки, проте використання кодування вимагає пристроїв кодера, декодера, інтерлівера, а отже, додаткових витрат на обробку. В умовах, коли потрібно зберегти високу швидкість передачі при забезпеченні заданої завадостійкості, необхідно наявність кодів, що дозволяють ефективно боротися з помилками, що відбуваються, і володіють швидкими процедурами кодування/декодування.

Завадостійкі коди знайшли широкі можливості для застосування у системах передачі інформації. Серед них можна виділити Low-Density Parity Check (LDPC) коди, які стали частиною багатьох стандартів комунікацій. Для розробки нових засобів, що використовують LDPC-коди актуальною є проблема дослідження та моделювання характеристик кодів при їх роботі в різних умовах, з різними даними та ін. Тому пропонується розробити систему, яка б дозволила проводити моделювання для певного класу кодів та досліджувати їх характеристики. Використання засобів і бібліотек JavaScript передбачає, що буде забезпечено зручний користувацький інтерфейс та забезпечена універсальність платформи для моделювання.

Мета: розробка системи моделювання роботи LDPC-кодів для дослідження характеристик таких кодів, що надасть можливість підвищити завадостійкість систем, які використовуватимуть такі рішення.

Об’єкт: процес декодування LDPC-кодів у системах передачі інформації та засоби моделювання цього процесу.

Предмет: програмна система моделювання процесу декодування LDPC-кодів з використанням бібліотек JavaScript.

Для досягнення поставленої мети необхідно вирішити такі **завдання**:

1. Виконати аналіз предметної області завадостійких кодів для передачі інформації та засобів моделювання, програмних засобів для реалізації системи моделювання. Виконати аналіз та обрати програмні засоби, які необхідні для реалізації системи моделювання.
2. Вибір алгоритму декодування LDPC-кодів, що буде використовуватись у системі.
3. Реалізація системи засобами JavaScript.
4. Тестування системи на обраних наборах вхідних даних.
5. Аналіз результатів тестування з метою порівняння роботи різних кодів LDPC у різних ситуаціях.

Практичне значення: розроблена система надасть можливість наглядно подивитись процесь декодування LDPC-кодів.

Апробація: результати дипломної роботи були представлені у форматі тез на Ольвійському форумі, який проводився з 23.06.2022 по 26.06.2022

РОЗДІЛ 1
АНАЛІТИЧНИЙ ОГЛЯД ЛІТЕРАТУРИ ТА ПАТЕНТНОЇ ІНФОРМАЦІЇ З ДЕКОДУВАННЯ LDPC-КОДІВ

У теорії інформації код перевірки парності з низькою щільністю (LDPC) — це лінійний код виправлення помилок, метод передачі повідомлення через шумовий канал передачі.[1][2] LDPC будується за допомогою розрідженого графа Таннера (підклас дводольного графа).[3] Коди LDPC є кодами, що наближаються до ємності, що означає, що існують практичні конструкції, які дозволяють встановити поріг шуму дуже близько до теоретичного максимуму (межі Шеннона) для симетричного каналу без пам'яті. Поріг шуму визначає верхню межу для шуму каналу, до якої ймовірність втрати інформації може бути мінімальною за бажанням. Використовуючи методику ітераційного поширення віри, коди LDPC можна декодувати в часі, лінійно довжині їх блоку.

Коди LDPC знаходять все більше застосування в програмах, які вимагають надійної та високоефективної передачі інформації через канали з обмеженою пропускною здатністю або каналами зворотного зв'язку за наявності шкідливих шумів. Реалізація кодів LDPC відставала від інших кодів, зокрема турбо-кодів. Термін дії фундаментального патенту на турбо-коди закінчився 29 серпня 2013 року.[4][5]

Коди LDPC також відомі як коди Галлагера на честь Роберта Г. Галлагера, який розробив концепцію LDPC у своїй докторській дисертації в Массачусетському технологічному інституті в 1960 році [6][7]. Також було показано, що коди LDPC мають ідеальні комбінаторні властивості. У своїй дисертації Галладжер показав, що коди LDPC з високою ймовірністю досягають межі Гілберта–Варшамова для лінійних кодів над двійковими полями. У 2020 році було показано, що коди LDPC Галлагера досягають можливості декодування списку, а також досягають межі Гілберта–Варшамова для лінійних кодів над загальними полями. [8]

1.1 Історія

Коли вперше було розроблено Галлагером у 1963 році [9], коди LDPC були забуті, поки його робота не була знову відкрита в 1996 році.[10] Турбо-коди, інший клас кодів, що наближаються до ємності, виявлені в 1993 році, стали вибраною схемою кодування наприкінці 1990-х років і використовувалися для таких додатків, як мережа глибокого космосу та супутниковий зв'язок. Однак завдяки прогресу в кодах з перевіркою парності з низькою щільністю вони перевершують турбо-коди з точки зору мінімальної кількості помилок і продуктивності у вищому діапазоні кодової швидкості, в результаті чого турбо-коди краще підходять лише для нижчих кодовів.[11]

1.2 Застосування

У 2003 році код LDPC в стилі нерегулярного повторення (IRA) перевершив шість турбо-кодів і став кодом для виправлення помилок у новому стандарті DVB-S2 для цифрового телебачення.[12] Відбірковий комітет DVB-S2 зробив оцінку складності декодера для пропозицій турбо-коду, використовуючи набагато менш ефективну архітектуру послідовного декодера, ніж архітектуру паралельного декодера. Це змусило пропозиції турбо-коду використовувати розміри кадрів близько половини розміру кадру пропозицій LDPC.

У 2008 році LDPC перевершив згорткові турбо–коди як систему прямого виправлення помилок (FEC) для стандарту ITU-T G.hn.[13] G.hn вибрав коди LDPC замість турбо-кодів через їх меншу складність декодування (особливо при роботі зі швидкістю передачі даних близько 1,0 Гбіт/с) і тому, що запропоновані турбо-коди демонстрували значну мінімальну похибку в бажаному діапазоні роботи.[14]

Коди LDPC також використовуються для 10GBASE-T Ethernet, який передає дані зі швидкістю 10 гігабіт на секунду по кабелю витої пари. З 2009 року коди LDPC також є частиною стандарту Wi-Fi 802.11 як додаткова

частина 802.11n і 802.11ac у специфікації High Throughput (HT) PHY.[15] LDPC є обов'язковою частиною стандарту 802.11ax (Wi–Fi 6).[16]

Деякі системи OFDM додають додаткову зовнішню корекцію помилок, яка виправляє випадкові помилки («мінімум помилок»), які виходять за межі внутрішнього коду корекції LDPC навіть при низьких частотах бітових помилок. Наприклад: код Ріда-Соломона з кодовою модуляцією LDPC (RS-LCM) використовує зовнішній код Ріда-Соломона.[17] Стандарти DVB-S2, DVB-T2 та DVB-C2 використовують зовнішній код BCH для видалення залишкових помилок після декодування LDPC.[18]

1.3 Оперативне використання

Коди LDPC функціонально визначаються за допомогою розрідженої матриці перевірки парності. Ця розріджена матриця часто генерується випадковим чином з урахуванням обмежень розрідженості — побудова коду LDPC обговорюється пізніше. Ці коди вперше були розроблені Робертом Галлагером у 1960 році.[7]

Нижче наведено фрагмент графіка прикладу коду LDPC з використанням позначення факторного графа Форні. На цьому графіку n змінних вузлів у верхній частині графіка з’єднані з (n–k) вузлами обмеження в нижній частині графіка.

Це популярний спосіб графічного представлення (n, k) коду LDPC. Біти дійсного повідомлення, розміщені на T у верхній частині графіка, задовольняють графічним обмеженням. Зокрема, всі рядки, що з’єднуються з вузлом змінної (поле зі знаком '='), мають однакове значення, а всі значення, що підключаються до факторного вузла (поле зі знаком '+'), повинні сумуватися за модулем два до нуля (у іншими словами, вони повинні складатися до парного числа або має бути парна кількість непарних значень).

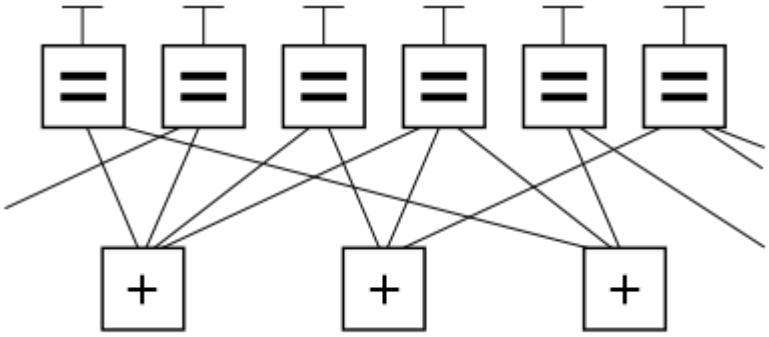

Рисунок 1.1 – Графік LDPC коду

Ігноруючи будь-які рядки, що виходять із зображення, існує вісім можливих шести бітових рядків, які відповідають дійсним кодовим словам: (тобто 000000, 011001, 110010, 101011, 111100, 100101, 001111, 101). Цей фрагмент коду LDPC являє собою трирозрядне повідомлення, закодоване як шість біт. Резервування використовується тут, щоб збільшити шанс відновлення після помилок каналу. Це (6, 3) лінійний код з n = 6 і k = 3.

Знову ігноруючи рядки, що виходять із зображення, матриця перевірки парності, що представляє цей фрагмент графіка, є:

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

Рисунок 1.2 – Матриця перевірки парності

У цій матриці кожен рядок представляє одне з трьох обмежень перевірки парності, тоді як кожен стовпець представляє один із шести біт в отриманому кодовому слові.

У цьому прикладі вісім кодових слів можна отримати, поставивши матрицю перевірки парності H у такий вигляд

$$[-P^T \mid I_{n-k}] \tag{1.1}$$

через основні операції з рядками в GF(2):

$$\mathbf{H} = \begin{pmatrix} 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 \end{pmatrix}_1 \sim \begin{pmatrix} 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 & 0 \end{pmatrix}_2 \sim \begin{pmatrix} 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 \end{pmatrix}_3 \sim \begin{pmatrix} 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 \end{pmatrix}_4.$$

Рисунок 1.3 – Матриця перевірки парності

Крок 1: H.

Крок 2: Ряд 1 додається до рядка ь3.

Крок 3: рядки 2 і 3 міняються місцями.

Крок 4: Ряд 1 додається до рядка 3.

З цього генераторну матрицю G можна отримати як

$$[I_k|P] \tag{1.2}$$

зазначаючи, що в окремому випадку це двійковий код

$$[P = -P] \tag{1.3}$$

або конкретно:

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

Рисунок 1.4 – Генераторна матриця

Нарешті, шляхом множення всіх восьми можливих 3-бітових рядків на G, отримують всі вісім дійсних кодових слів. Наприклад, кодове слово для бітового рядка '101' отримується за допомогою:

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

Рисунок 1.5 – Прилад генерації кодового

де $\odot$ — символ множення по моду 2.

Як перевірку, простір рядків G ортогональний до H, так що

$$G \odot H^T = 0 \tag{1.4}$$

Рядок бітів '101' міститься як перші 3 біти кодового слова '101011'.

1.4 Приклад кодера

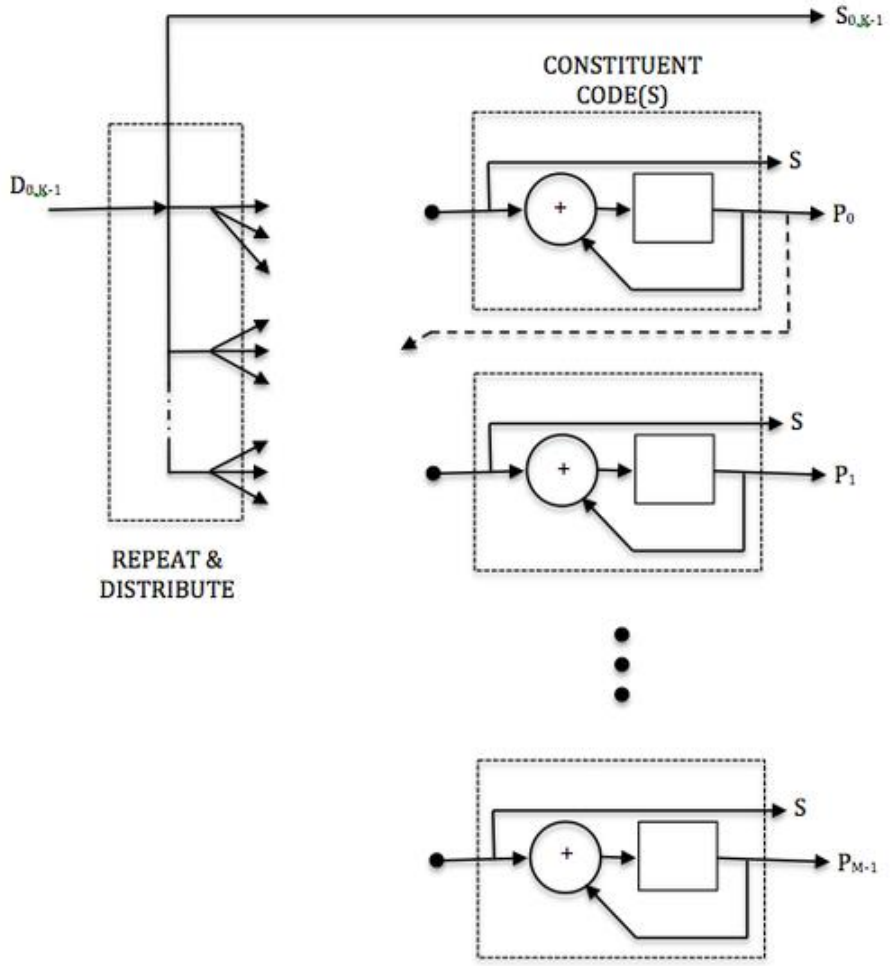

Рисунок 1.6 – Функціональні компоненти більшості кодерів LDPC.

Під час кодування кадру біти вхідних даних (D) повторюються і розподіляються на набір складових кодерів. Складовими кодерами, як правило, є акумулятори, і кожен акумулятор використовується для створення символу парності. Одна копія вихідних даних (S0,K-1) передається з бітами парності (P) для створення кодових символів. S-біти кожного складового кодера відкидаються. Біт парності може використовуватися в іншому складовому коді.

У прикладі використання коду зі швидкістю 2/3 DVB-S2 розмір закодованого блоку становить 64800 символів (N=64800) з 43200 бітами даних (K=43200) і 21600 бітами парності (M=21600). Кожен складовий код (контрольний вузол) кодує 16 біт даних, за винятком першого біта парності,

який кодує 8 біт даних. Перші 4680 бітів даних повторюються 13 разів (використовуються в 13 кодах парності), тоді як інші біти даних використовуються в 3 кодах парності (нерегулярний код LDPC).

Для порівняння, класичні турбо-коди зазвичай використовують два складових коди, налаштовані паралельно, кожен з яких кодує весь вхідний блок (K) бітів даних. Ці складові кодери є рекурсивними згортковими кодами (RSC) помірної глибини (8 або 16 станів), які розділені обмежувачем коду, який розмежовує одну копію кадру.

Код LDPC, навпаки, використовує багато складових кодів (акумуляторів) малої глибини паралельно, кожен з яких кодує лише невелику частину вхідного кадру. Багато складових кодів можна розглядати як багато "згорткових кодів" малої глибини (2 стани), які з'єднані за допомогою операцій повторення та розподілу. Операції повтору та розподілу виконують функцію обмежувача в турбо-коді.

Здатність більш точно керувати з’єднаннями різних складових кодів і рівень надмірності для кожного вхідного біта надають більше гнучкості в розробці кодів LDPC, що в деяких випадках може призвести до кращої продуктивності, ніж турбо коди. Здається, що турбо-коди все ще працюють краще, ніж LDPC при низьких швидкостях коду, або, принаймні, дизайн високопродуктивних кодів з низькою швидкістю легше для турбо-кодів.

На практиці апаратне забезпечення, яке формує акумулятори, використовується повторно під час процесу кодування. Тобто, як тільки генерується перший набір бітів парності та зберігаються біти парності, те саме обладнання акумулятора використовується для створення наступного набору бітів парності.

1.5 Декодування

Як і в інших кодах, декодування з максимальною ймовірністю коду LDPC на двійковому симетричному каналі є NP-повною проблемою.

Виконання оптимального декодування для NP-повного коду будь-якого корисного розміру непрактично.

Однак не оптимальні методи, засновані на ітеративному декодуванні поширення віри, дають чудові результати і можуть бути практично реалізовані. Не оптимальні методи декодування розглядають кожну перевірку на парність, яка складає LDPC, як незалежний код єдиної перевірки парності (SPC). Кожен код SPC розшифровується окремо за допомогою методів м'якого введення-м'якості (SISO), таких як SOVA, BCJR, MAP та інших їх похідних. Інформація м'якого рішення від кожного декодування SISO перевіряється та оновлюється іншими надлишковими декодуваннями SPC того ж інформаційного біта. Кожен код SPC потім знову декодується з використанням оновленої інформації м'якого рішення. Цей процес повторюється до тих пір, поки не буде досягнуто дійсне кодове слово або не буде вичерпано декодування. Цей тип декодування часто називають декодуванням суми добутку.

Декодування кодів SPC часто називають обробкою «перевірочного вузла», а перехресну перевірку змінних часто називають обробкою «змінного вузла».

У практичній реалізації декодера LDPC набори кодів SPC декодуються паралельно для збільшення пропускної здатності.

На відміну від цього, поширення переконань на каналі двійкового стирання є особливо простим, коли воно складається з ітераційного виконання обмежень.

Наприклад, врахуйте, що дійсне кодове слово 101011 з наведеного вище прикладу передається через двійковий канал стирання та приймається зі стиранням першого та четвертого бітів, щоб отримати ?01?11. Оскільки передане повідомлення повинно задовольняти кодові обмеження, повідомлення можна представити записом отриманого повідомлення у верхній частині графіка факторів.

У цьому прикладі перший біт ще не може бути відновлений, оскільки всі пов’язані з ним обмеження мають більше одного невідомого біта. Щоб продовжити декодування повідомлення, повинні бути ідентифіковані обмеження, що підключаються лише до одного зі стертих бітів. У цьому прикладі достатньо лише другого обмеження. Розглядаючи друге обмеження, четвертий біт повинен бути нульовим, оскільки лише нуль у цій позиції задовольняє обмеження.

Потім цю процедуру повторюють. Нове значення для четвертого біта тепер можна використовувати разом з першим обмеженням для відновлення першого біта, як показано нижче. Це означає, що перший біт повинен бути одиницею, щоб задовольнити крайнє ліве обмеження.

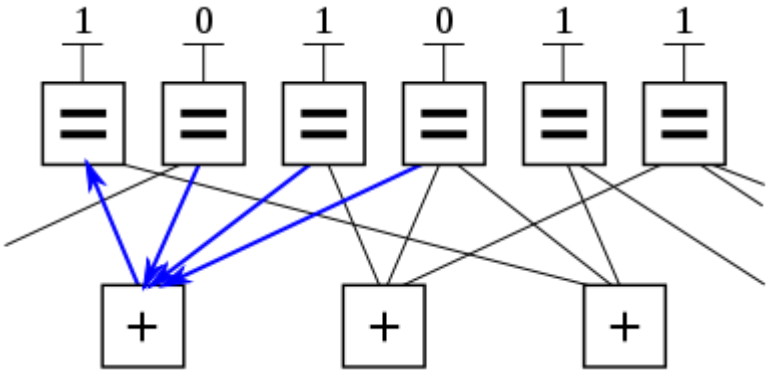

Рисунок 1.6 – Відновлення першого біта

Таким чином, повідомлення можна декодувати ітераційно. Для інших моделей каналів повідомлення, що передаються між вузлами змінних і перевіреними вузлами, є реальними числами, які виражають ймовірність і вірогідність.

Цей результат можна перевірити, помноживши виправлене кодове слово r на матрицю перевірки парності H:

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

Рисунок 1.7 – Процес декодування

Оскільки результатом z (синдромом) цієї операції є вектор три × один нуль, отримане кодове слово r успішно перевіряється.

Після завершення декодування вихідні біти повідомлення «101» можна витягти, подивившись на перші 3 біти кодового слова.

Хоча ілюстративний, цей приклад стирання не показує використання декодування м'якого рішення або передачі повідомлень м'якого рішення, яке використовується практично у всіх комерційних декодерах LDPC.

В останні роки також було проведено велику роботу, витрачену на вивчення впливу альтернативних розкладів для оновлення вузлів змінних і обмежених вузлів. Початкова техніка, яка використовувалася для декодування кодів LDPC, була відома як flooding. Цей тип оновлення вимагає, щоб перед оновленням вузла змінної необхідно було оновити всі вузли обмежень і навпаки. У пізнішій роботі Віла Касадо та ін. [21] були вивчені альтернативні методи оновлення, за яких змінні вузли оновлюються найновішою доступною інформацією про контрольний вузол.

Інтуїція за цими алгоритмами полягає в тому, що змінні вузли, значення яких змінюються найбільше, є тими, які потрібно оновити першими. Високонадійні вузли, чия величина коефіцієнта логарифмічної правдоподібності (LLR) велика і не змінюється суттєво від одного оновлення до наступного, не потребують оновлення з тією ж частотою, що й інші вузли, знак і величина яких коливаються ширше. Ці алгоритми планування показують більшу швидкість конвергенції та нижчі мінімальні похибки, ніж ті, що використовують flooding. Ці нижчі межі помилок досягаються завдяки

здатності алгоритму інформованого динамічного планування (IDS)[21] долати перехоплення наборів близьких кодових слів.[22]

Коли використовуються алгоритми планування без затоплення, використовується альтернативне визначення ітерації. Для (n, k) коду LDPC зі швидкістю k/n повна ітерація відбувається, коли оновлюються n змінних і n − k вузлів обмеження, незалежно від порядку, в якому вони були оновлені.

1.6 Побудова коду

Для великих розмірів блоків коди LDPC зазвичай конструюються, спочатку вивчаючи поведінку декодерів. Оскільки розмір блоку прагне до нескінченності, можна продемонструвати, що декодери LDPC мають поріг шуму, нижче якого надійно досягається декодування, і вище якого декодування не досягається [23], що в просторіччі називають ефектом обриву. Цей поріг можна оптимізувати шляхом знаходження найкращої пропорції дуг із контрольних вузлів і дуг із змінних вузлів. Приблизним графічним підходом до візуалізації цього порогу є діаграма EXIT.

Побудова конкретного коду LDPC після такої оптимізації поділяється на два основних типи методів:

1) Псевдовипадкові підходи
2) Комбінаторні підходи

Побудова за допомогою псевдовипадкового підходу базується на теоретичних результатах, що для великого розміру блоку випадкова конструкція дає хорошу продуктивність декодування.[10] Загалом, псевдовипадкові коди мають складні кодери, але псевдовипадкові коди з найкращими декодерами можуть мати прості кодери.[24] Різні обмеження часто застосовуються, щоб гарантувати, що бажані властивості, очікувані на теоретичній межі нескінченного розміру блоку, відбуваються при кінцевому розмірі блоку.

Комбінаторні підходи можна використовувати для оптимізації властивостей кодів LDPC невеликого розміру блоку або для створення кодів за допомогою простих кодерів.

Деякі коди LDPC засновані на кодах Ріда-Соломона, наприклад, код RS-LDPC, що використовується в стандарті 10 Gigabit Ethernet.[25] Порівняно з випадково згенерованими кодами LDPC, структуровані коди LDPC — наприклад, код LDPC, що використовується в стандарті DVB-S2 — можуть мати простіший і, отже, більш дешевий апаратне забезпечення — зокрема, коди, сконструйовані таким чином, що матриця H є матрицею циркуляції. [26]

Ще одним способом побудови кодів LDPC є використання кінцевої геометрії. Цей метод був запропонований Y. Kou. у 2001 році.[27]

1.7 Переваги LDPC-кодів, їх застосування та перспективи розвитку

LDPC-коди в сучасних системах передачі інформації займають нішу, аналогічну турбо кодів. Обидва ці класи кодів використовуються в системах, де потрібні підвищені швидкості передачі при обмеженій смузі пропускання каналу. До таких систем можна віднести, наприклад, супутниковий зв'язок, цифрове телебачення (зокрема високої чіткості), а також канали передачі в електронно-обчислювальних машинах та їх мережах. LDPC-кодери можуть забезпечувати колосальну швидкість передачі даних (до 40 Гб/с), що обумовлено простотою їхньої реалізації. Найшвидшими декодерами розумно було б вважати багатопорогові декодери (МПД), що декодують за однойменним алгоритмом. В МПД можуть легко декодуватися довгі коди, в широкому діапазоні кодових швидкостей під час використання як жорсткого, і м'якого модемів. При цьому МПД виконує лише найпростіші операції складання та порівняння невеликих цілих чисел, що обумовлює його крайню простоту за всіх варіантів програмної чи апаратної реалізації. Наприклад, МПД може бути реалізований з використанням лінійних зсувних регістрів найшвидших апаратних елементів.

Слід зазначити, що LDPC—кодування не є суто теоретичною розробкою, а вже активно використовується та введено в деякі стандарти. наприклад, у 2003 р. LDPC—код замість турбокода став частиною стандарту DVB-S2 супутникової передачі для цифрового телебачення. Аналогічна заміна відбулася і в стандарті DVB-T2 для цифрового телевізійного наземного мовлення. також LDPC-коди увійшли до стандарту IEEE 802, мережі Ethernet 10G та інші.

За результатами дослідження серед кодів для включення до стандарту DVB-S2 були відзначені такі переваги:

1) відставання від кордону Шеннона лише на 0,6–0,8 дБ;

2) перевага на 0,3 дБ порівняно з найкращим із представлених турбокодів;

3) перевага на 2,5–3,0 дБ, тобто 30–відсотковий приріст у потужності, порівняно із стандартом DVB–S.

1.8 Висновки до розділу 1

Було охарактеризовано та розглянуто методі побудову та декодування LDPC-кодів. Розглянуто як новітні так і застарілі методи та алгоритми.

Також було розглянуто варіанти застосування та оперативного використання LDPC-кодів. Порівняння класичних турбо-кодів та LDPC, які використовують багато складових кодів (акумуляторів) малої глибини паралельно, кожен з яких кодує лише невелику частину вхідного кадру.

Як і в інших кодах, декодування з максимальною ймовірністю коду LDPC на двійковому симетричному каналі є NP-повною проблемою. Виконання оптимального декодування для NP-повного коду будь-якого корисного розміру непрактично. Однак не оптимальні методи, засновані на ітеративному декодуванні, дають чудові результати і можуть бути практично реалізовані. Не оптимальні методи декодування розглядають кожну перевірку на парність, яка складає LDPC, як незалежний код єдиної перевірки парності.

РОЗДІЛ 2
МЕТОДИ ТА АЛГОРИТМИ ДЕКОДУВАННЯ LDPC. СХЕМИ ДЕКОДУВАННЯ ТА ОЦІНКА ЕФЕКТИВНОСТІ LDPC–КОДІВ.

Коди з малою щільністю перевірок на парність (LDPC–код від англ. Low–density parity–check code, LDPC–code, низькощільний код) були вперше запропоновані р. Галлагером і пізніше досліджувалися у багатьох наукових працях. незважаючи на те, що протягом тривалого часу LDPC були практично виключені з розгляду, в останні роки спостерігається збільшення кількості досліджень у цій галузі. це пов'язано з тим, що, володіючи поганим мінімальним відстанню, коди з малою щільністю, тим не менш, забезпечують високий рівень виправлення помилок за дуже малої складності їх декодування.

Було показано, що зі зростанням довжини деякі LDPC–коди можуть перевершувати турбокоди та наближатися до пропускної спроможності каналу з адитивним білим гауссівським шумом (АБГШ). разом з тим багато запропонованих конструкції LDPC-кодів є циклічними або квазіциклічними, що дозволяє робити не тільки швидке декодування, а й ефективні процедури кодування. Крім того, навіть для LDPC-кодів, що не володіють властивістю циклічності, були запропоновані ефективні процедури кодування

Поява нових ефективних алгоритмів декодування стимулювала та підвищення інтерес до методів побудови недвійкових LDPC–кодів. Спочатку побудова недвійкових LDPC кодів здійснювалося шляхом заміни ненульових елементів у перевірочній матриці двійкових. LDPC–коди на випадкові елементи кінцевого поля. Пізніше Ліїн (Lin) запропонував методи алгебраїчної побудови квазіциклічних недвійкових LDPC-кодів (QC NB-LDPC)

LDPC-коди стають потрібними в системах передачі інформації, що вимагають максимальну швидкість передачі при обмеженій смузі частот. Основним конкурентом LDPC-кодів на даний момент є турбокоди, які знайшли своє застосування в системах супутникового зв'язку, ряду стандартів

цифрового телебачення та мобільних систем зв'язку третього покоління. проте LDPC-коди проти турбокодами мають ряд переваг:

1) LDPC-коди обганяють турбокоди за швидкістю декодування;
2) LDPC-коди кращі в каналах з меншими ймовірностями помилок. З розвитком методів передачі інформації канали передачі покращуються, що дає хорошу перспективу для розвитку LDPC-кодів.

Має місце також і правовий аспект застосування LDPC-кодів та турбокодів. компанії France Telecom та Telediffusion de France запатентували широкий клас турбокодів, що обмежує можливість їх вільного застосування і водночас стимулює розвиток та використання інших методів кодування, таких як LDPC.

2.1 Стиснуті коди зондування та лінійні блочні коди

У стисненому зондуванні важливим процесом є визначення розташування ненульових компонентів під час реконструкції, і це відомо як відновлення підтримки або відновлення шаблону розрідженості, яке має застосування в обробці сигналів, шумозаглушенні, цифровому зв'язку тощо. Аналогічно, у лінійному блоковому коді важливою частиною є визначення розташування бітів помилки під час декодування, і цей процес називається відновленням шаблону помилки.

2.1.1 Відновлення шаблону розрідженості та алгоритм MP

Поширеною проблемою в стиснутому зондуванні є оцінка невідомого розрідженого вектора $x \in R^n$ з лінійних вимірювань наступного вигляду [10, 11].

$$y = Hx + N \tag{2.1}$$

де matrix $H \in R^{m \times n}$ – це задана матриця вимірювань або надлишковий словник і є вектором адитивного шуму. Кажуть, що вектор $x \in R^n$ є розрідженим, оскільки відомо, що він має відносно невелику кількість відмінних від нуля компонентів, але розташування та значення цих компонентів невідомі і

повинні бути виявлені як частина оцінки сигналу. Визначення розташування ненульових компонентів відоме як відновлення шаблону розрідженості або відновлення підтримки, як згадувалося вище. Що стосується розрідженого сигналу, то максимальна кількість ненульових компонент становить.

2.1.2 Лінійне вимірювання в лінійних блочних кодах

Враховуючи (n, k) лінійний блоковий код C, визначений як нульовий простір $m \times n$ матриці перевірки парності, $H = H_1, H_2, \cdots, H_n$ рівняння $Hc^T = 0$ виконується для будь-якого кодового слова $c \in C$.

2.2 Декодування двійкових кодів LDPC за допомогою алгоритму MP

Властивість обмеженої ізометрії (RIP) може гарантувати точність реконструкції в стиснутому зондуванні. Однак для більшості лінійних блочних кодів дуже важко визначити, чи відповідає матриця перевірки парності умові RIP. На щастя, існує певна відповідна література, яка може допомогти нам вирішити цю проблему.

2.3 Ази блочного кодування

LDPC коди - це лінійні блокові коди, а значить перевірочні біти в даній схемі кодування додаються до кінця інформаційного повідомлення - блоком.

Відповідно, процедура кодування (encoding) - є ніщо інше, як перемноження вектора інформаційного повідомлення довжиною K на деяку матрицю, що породжує G:

$$a = u \otimes G \tag{2.2}$$

де символ $\otimes$ - це множення за модулем. Для двійкових кодів це $modulo\ 2$, для недвійкових $modulo\ q$, з полів Галуа $GF(q = 2^p)$.

Відповідно, і кодова швидкість теж задається через матрицю, що породжує:

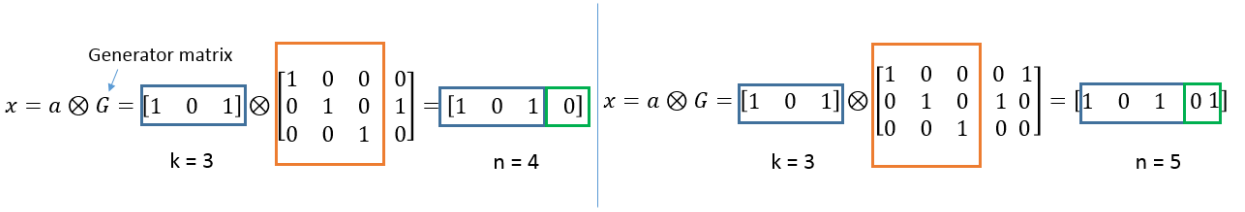

Рисунок 2.1 – Породжувальна матриця

Породжувальна матриця складається з двох конкатенованих (з'єднаних) частин:

$$G = K \times N \tag{2.3}$$

$$I = K \times K \tag{2.4}$$

$$P = K \times (N - K) \tag{2.5}$$

$$G = [I \ -P] \tag{2.6}$$

де P - це, так звана, парна частина, а I - одинична (identity) матриця.

Мінус в останній частині потрібен тому, що при множенні та додаванні по модулю потрібно дотримуватися правил зіставлення негативних і позитивних чисел:

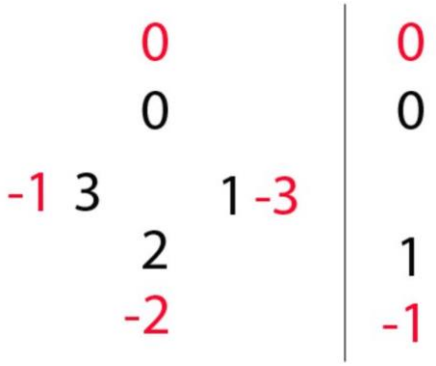

Рисунок 2.2 – Приклад множення та додавання по модулю

Так як ми говоримо про лінійні блокові коди, матриця, що породжує, і повинна забезпечувати цю лінійність. Тобто рядки матриці, що породжує, повинні бути лінійно незалежними.

Треба звернути увагу, що identity-частина потрібна для того, щоб залишати код систематичним: інформаційне повідомлення залишається незмінним, а перевірочні біти додаються до кінця блоком. За такої схеми, правильно відновивши кодове слово, можна відновити початкове повідомлення, просто прибравши перевірочні біти. Зручно, чи не так?

Матриця, що породжує, безпосередньо пов'язана з іншою найважливішою матрицею, що використовується під час процедури декодування: з матрицею перевірки на парність.

Матриця перевірки на парність має $(N - K)$ рядків і N стовпців, де N відповідає довжині кодового слова, а K, повторимо, відповідає довжині повідомлення:

$$H = [P^T \; I] \tag{2.7}$$

де:

$$H = (N - K) \times K \tag{2.8}$$

$$P^T = (N - K) \times K \tag{2.9}$$

$$I = (N - K) \times (N - K) \tag{2.10}$$

Її основну ідею дуже зручно пояснювати за допомогою графа Таннера:

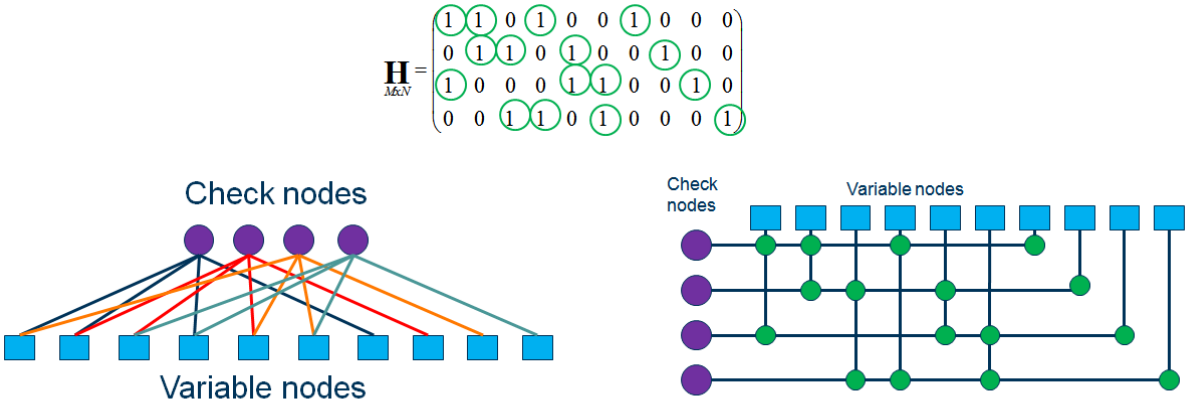

Рисунок 2.3 – Граф Таннера

Тобто існує два види вузлів: так звані, вузли змінних (variable nodes), кількість яких відповідають числу стовпців K, та вузли перевірки (check nodes), що відповідають числу рядків $(N - K)$. Вузли пов'язані між собою, і зв'язок визначається положенням одиниць матриці H. Якщо елемент матриці дорівнює 1, значить зв'язок між вузлами є, якщо дорівнює 0 зв'язку немає.

Для того, щоб вважати процедуру декодування успішною, потрібно, щоб на всіх перевірочних вузлах сформувалися певні значення, як правило, нулі:

$$s = Hx = 0 \tag{2.11}$$

2.4 Ази LDPC кодів

Але все вище описане - це загальні моменти для більшості блокових кодів. Чим тоді LDPC відрізняються від тих самих кодів Хеммінга?

Загалом, тим, що й визначає їх як low-density: їх матриці перевірки на парність мають бути розрядженими (sparce), тобто нулів у них має бути значно більше ніж будь-чого іншого.

Наприклад, у того ж Галлагера дана матриця була такою:

$$H = \left[\begin{array}{cccccccccccc} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ \hline 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ \hline 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \end{array}\right].$$

Рисунок 2.4 – Матриця Галлагера

(3,4)-регулярна матриця перевірки на парність довжиною 12. Пояснення: кодове слово, яке буде закодовано за допомогою такого коду, матиме довжину 12 біт; у кожному стовпці 3 одиниці, а кожному рядку 4,

звідси позначення (3,4); кількість одиниць у рядках і стовпцях — це константи (у разі 3 і 4), отже код — регулярний.

У Маккея та Ніла матриця перевірки на парність була такою:

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

Рисунок 2.5 – Матриця Маккея та Ніла

(3,4)-регулярна матриця перевірки на парність довжиною 12.

Проте нічого не помічаєте? Правильно: ці матриці не підпадають під стандартну форму з формули (3), адже для LDPC кодів ми прагнемо зробити перевірочні матриці розрідженими. А якщо матриці перевірки не потрапляють під стандартну форму, значить не зовсім зрозуміло, як для них сформувати матриці, що породжують.

2.5 Декодування LDPC кодів

Візьмемо за основу один з центральних і найбільш, мабуть, популярних алгоритмів декодування — алгоритму Belief propagation (aka SPA — Sum-product algorithm).

Отже, по-перше, припустимо, що ми маємо якусь систему зв'язку:

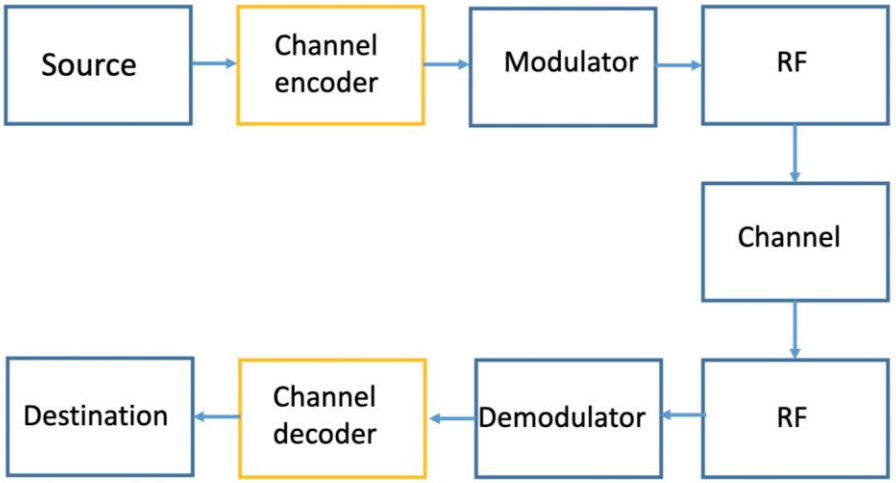

Рисунок 2.6 – Система зв'язку

Система зв'язку складається з:

1) джерела повідомлення (source);
2) передавача;
3) бездротового каналу зв'язку;
4) приймача;
5) та одержувача повідомлення (destination).

Передавач складається з:

1) перешкодостійкого кодера (channel encoder);
2) цифрового модулятора (digital modulator);
3) радіочастоти (RF - radio frequency).

Приймач складається з:

1) радіочастоти;
2) цифрового демодулятора (digital demodulator);
3) завадостійкого декодера (channel decoder).

Договоримся, що під цифровими модемами будемо розглядати в першу чергу найбільш популярні їх різновиди: PSK і QAM.

Чим цікавий дані типи модуляції? По-перше, тим, що саме вони входять у стандарти сучасних бездротових систем (LTE, Wi-Fi, DVB і т.д.).

А, по-друге, тим, що вони вміють представляти зашумлені значення, отримані з зв'язку, у формі, так званих, м'яких значень демодуляції (soft decisions). Або, якщо висловлюватися інакше, у формі логарифмованих коефіцієнтів правдоподібності (LLR — log likelihood ratios):

$$r = ln\left(\frac{Pr(x=0)}{Pr(x=1)}\right) = ln\left(\frac{1-p}{p}\right) \quad (2.12)$$

де p означає ймовірність, а x означає деяку подію.

Частіше трапляється випадок, коли в чисельнику залишають ймовірність того, що біт спочатку був банкрутом, а в знаменнику — що був одиницею. Відповідно, якщо значення LLR на виході демодулятора негативне, ми можемо припустити, що швидше за все біт був нульовим, а якщо позитивне — одиницею. Власне, особливої різниці у положенні ймовірностей немає. Головне, щоб приймач і передавач були однаково обізнані про обрану схему накладання (мапінгу).

2.6 Belief propagation

Belief propagation (далі BP), також відомий як sum-product message passing, є алгоритмом передачі повідомлень для виконання висновку на графічних моделях, таких як байєсівські мережі та випадкові поля Маркова. Він обчислює граничний розподіл для кожного неспостережуваного вузла (або змінної), залежно від будь-яких спостережуваних вузлів (або змінних). BP зазвичай використовується в теорії штучного інтелекту та інформації, і воно продемонструвало емпіричний успіх у численних застосуваннях, включаючи коди перевірки парності з низькою щільністю, турбокоди, наближення вільної енергії та виконання.

Алгоритм працює з ймовірностями. А точніше, із тими натуральними логарифмами від відносин ймовірностей.

Під пересиланням повідомлень між вузлами перевірки та змінних розуміється те, що LLR будуть складатися та перемножуватись за певними формулами.

На етапі ініціалізації алгоритму LLR відповідають апріорним ймовірностям. SPA є одним з алгоритмів максимальної апостеріорної ймовірності (MAP - maximum a posteriori probability), а значить він прагне максимізувати апостеріорну ймовірність, отриману після ітеративної пересилки між вузлами перевірок та змінних.

Алгоритм був вперше запропонований Judea Pearl у 1982 році, який сформулював його як точний алгоритм висновку на деревах, пізніше поширений на полідерева.

2.6.1 Ініціалізація

Початковою точкою нашого алгоритму є матриця значень LLR, повторююча структуру матриці H. Підберемо аналітичний опис:

$$M = (r \cdot 1)^T \odot H \tag{2.13}$$

де 1 є масивом одиниць, а $\odot$ означає похідну Адамара (поелементне множення). На практиці без одиничної матриці можна буде обійтися: замінимо дужку на ітераційне множення Адамара вектора LLR зі стовпцями матриці контролю парності (потрібний буде додатковий цикл). Якщо матриці будуть досить великими, такий підхід може бути ефективнішим з погляду пам'яті.

2.6.2 Повідомлення V2C

Потім, так званий, горизонтальний крок: алгоритм вимагає обробки повідомлення (V2C) в області ймовірності. Для переходу від LLR до ймовірностей скористаємося ставленням між гіперболічним тангенсом та натуральним логарифмом:

$$tanh = \left(\frac{1}{2} ln\left(\frac{1-p}{p}\right)\right) = 1 - 2p \tag{2.14}$$

Власне, процедура передачі V2C повідомлення - це перемноження ненульових ймовірностей у кожному рядку:

$$E_{i,j} = log\left(\frac{1+\Pi_{i' \in B_j i' \neq i} tanh(M_{j,i}/2)}{1-\Pi_{i' \in B_j i' \neq i} tanh(M_{j,i}/2)}\right) = log\left(\frac{1+\Pi_{i' \in B_j i' \neq i} M'_{j,i'}}{1-\Pi_{i' \in B_j i' \neq i} M'_{j,i'}}\right) \quad (2.15)$$

де j — це номер певного рядка, i — це номер певного стовпця, B_j — це безліч ненульових значень в j рядку, а вираз $i' \neq i$ означає, що ми виключаємо i вузол змінних (variable node) з розгляду.

Тобто на цьому етапі нам потрібно:

1) вибрати елемент у матриці M, переведений у ймовірнісну область за формулою (2.14);

2) якщо його позиція відповідає позиції ненульового елемента в матриці H, перемножити всі ненульові ймовірності рядка даного елемента;

3) виключити з перемноження цей елемент (до або після попереднього пункту);

4) повторити всі попередні кроки кожного елемента матриці.

Пункт з винятком вузла з розгляду можна провести двома способами: з'ясовувати потрібне підмножина до перемноження ймовірностей або видаляти значення результату після підрахунків.

2.6.3 Перевірка критерію зупинення декодування

Отже, ми підходимо до кінця першої ітерації, а значить настав час оновити наші апріорні ймовірності — зробити їх апостеріорними:

$$l_i = r_i + \sum_{j \in A_i} \quad E_{j,i} \quad (2.16)$$

де A_i - це безліч елементів, що відповідають ненульовим елементам матриці перевірки на парність в i стовпці.

2.6.4 Повідомлення C2V

На цьому етапі слід перерахувати матрицю \mathbf{M}:

$$M_{j,i} = \sum_{j' \in A_i j' \neq j} \quad E_{j',i+r_i} \tag{2.17}$$

І далі перейти до обчислення матриці E. І так доти, доки не виконається пункт 3 (або не закінчиться кількість доступних ітерацій).

2.7 Графік для лінійних блочних кодів

Граф G складається з множини вершин, позначених $V = \{v_1, v_2, \cdots\}$, і множини ребер, позначених $E = \{e_1, e_2, \cdots\}$, такі, що кожне ребро e_k ототожнюється з неупорядкованою парою (v_i, v_j) вершин. Такий графік G позначимо $G = (V, E)$. Вершини v_i і v_j пов'язані з ребром e_k, називаються кінцевими вершинами e_k. Граф найчастіше представляється діаграмою, в якій вершини представлені як точки, а кожне ребро як лінія, що з'єднує його кінцеві вершини. У цьому графічному зображенні дві кінцеві вершини ребра називаються з'єднаними ребром, а ребро – інцидентним (або на) його кінцевими вершинами. Кількість ребер, які нападають на вершину v_i, називається степенем вершини v_i, позначеним $d(v_i)$. На рисунку 17.3 зображено граф, що складається з шести вершин і 10 ребер. Ребро b з'єднує вершини v_1 та v_2.

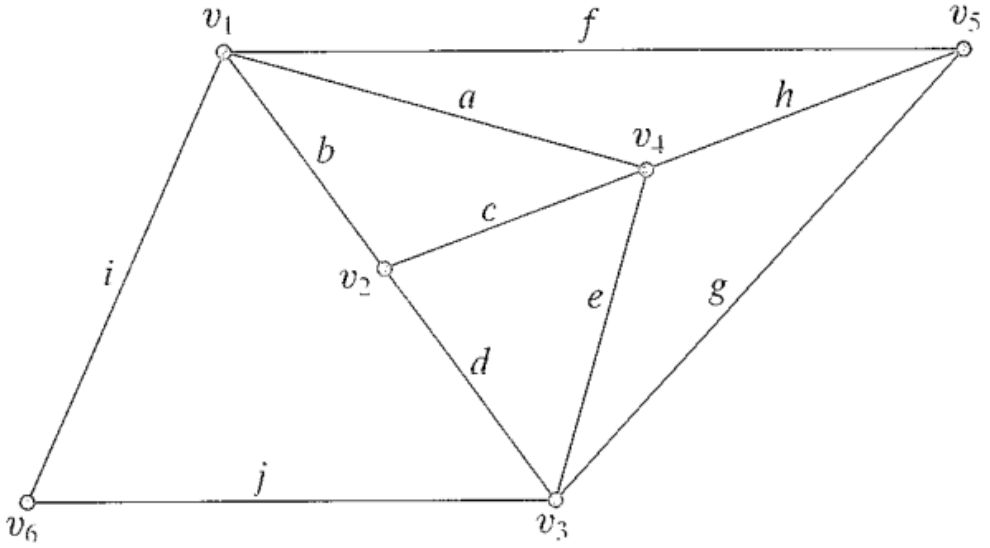

Рисунок 2.7 – Граф з шістьма вершинами і 10 ребрами.

Краї b, c. і d є інцидентними на вершину v_2, отже, ступінь вершини m дорівнює 3. Два ребра, які падають на спільну вершину, називаються суміжними або сполучними. Дві вершини називаються суміжними, якщо вони з'єднані ребром. Граф зі скінченною кількістю вершин, а також скінченною кількістю ребер називається скінченним графом.

Шлях у графі визначається як скінченна чергується послідовності і ребер, початок і кінець, що закінчуються вершинами, так що кожне ребро як попередній і наступний за ним, і жодна вершина не з'являється більше одного разу. Кількість ребер на шляху називається довжиною шляху. Можливо, що шлях починається і закінчується в одній вершині. Такий дозований шлях називається циклом. Жодна вершина циклу (крім початкової та кінцевої) не з'являється більше одного разу. Ребро, у якого початкова і кінцева вершини однакові, утворює цикл довжини 1 і називається самопетлею. Граф без циклів називається ациклічним. Довжина найкоротшого циклу в графі називається обхватом графа. Обхват графіка на рисунок 2.7 дорівнює 3.

Граф G називається зв'язним, якщо є принаймні одним шлях між кожною парою вершин у G. Обидва графіки, показані на рисунках 2.7 і 2.8, є зв'язними графами. Граф $G = (V, E)$ називається біпанітним графом, якщо його вершину V можна розбити на дві непересікаючі підмножини V_1 і V_2 так, що кожне ребро в з'єднується з вершиною в V_1 з вершиною в V_2.

Очевидно, що дводольний граф не має самопетлі. На малюнку 2.8 зображено дводольний графік з $V_1 = \{v_1, v_2, v_3\}$ та $V_2 = \{v_4, v_5, v_6, v_7, v_8\}$. якщо дводольний $G = (V, E)$ містить цикли, тоді всі цикли мають парні бордюри. Щоб переконатися, що це правда, припустимо, що ми простежимо цикл у G від вершини v_i у V_2. Перше ребро цього циклу має з'єднати вершину v_j, з venex v_k, в V_1, Друге ребро циклу повинно з'єднати v_j, з вершиною v_k у V_2. Третє ребро циклу з'єднує v_k з вершиною v_1 у V_2 Потім четверте ребро циклу з'єднує v_2 з вершиною в V_1. Цей процес налаштування триває до тих пір, поки останнє ребро циклу не закінчиться у початковій вершині v_i.

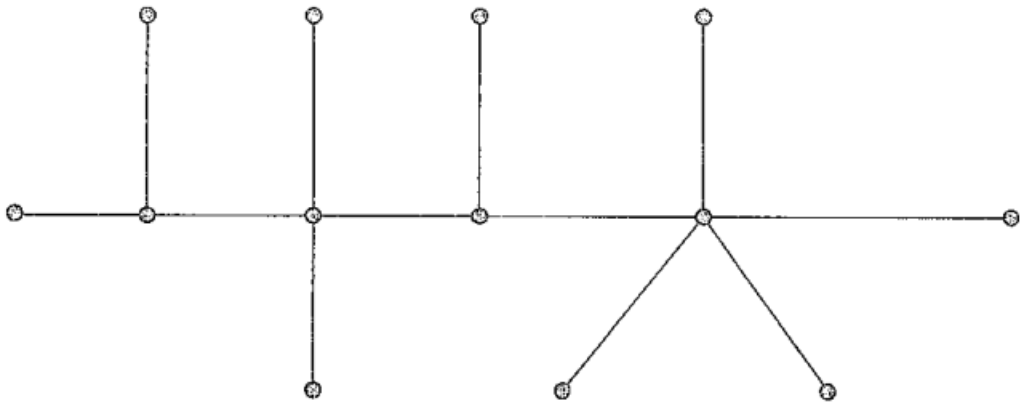

Рисунок 2.8 – Граф з шістьма вершинами і 10 ребрами.

2.8 «Жорстке» декодування

«Жорстке» декодування — це схема декодування для двійкового симетричного каналу при невеликій кількості помилок у каналі. "Жорстке" декодування інвертуванням бітів — найпростіша схема декодування кодів з низькою щільністю перевірок на парність.

Під перевіркою розуміється будь-який рядок $h = \{h_0, \ldots, h_{N-1}\}$ із перевірочної матриці коду з низькою щільністю перевірок на парність. Будемо говорити, що перевірка для якогось вектора $y = \{y_0, \ldots, y_{N-1}\}$ виконується тоді, коли скалярний добуток вектора y на перевірку дає нуль. Будемо говорити, що елемент уі прийнятного вектора у бере участь у перевірці $h = \{h_0, \ldots, h_{N-1}\}$ тоді, коли відповідний елемент перевірки h_i не дорівнює нулю.

Одна ітерація «жорсткого» декодування інвертуванням бітів проводиться наступним чином.

1) Для прийнятого вектора обчислюються всі перевірки;

2) якщо деякий біт прийнятого вектора брав участь більш ніж у половині невиконаних перевірок, біт інвертується.

3) Після такого аналізу всіх символів прийнятого вектора вектор перевіряється на належність коду. якщо вектор є кодовим словом, декодування закінчується, інакше виконується така ітерація алгоритму.

Така процедура декодування застосовна для кодів з низькою щільністю перевірок на парність тому, що більшість перевірок у такому разі міститимуть одну помилку або не будуть містити помилок взагалі і тоді невиконання великої кількості перевірок для символу прийнятого слова означатиме наявність у ньому помилки.

Складність однієї ітерації «жорсткого» декодування інвертуванням біт є лінійної, кількість ітерацій декодування зазвичай $log_2(N)$, де N – довжина кодового слова.

2.9 Декодування за ймовірностями

Декодування за ймовірностями є «м'яким» декодуванням, тобто декодуванням на основі вектора, що складається не з дискретних значень (0 і 1), а з речових величин, одержаних на виході каналу шляхом перерахунку ймовірностей.

На основі прийнятого з каналу вектора формуються два (для двійкового випадку) вектора ймовірностей того, що в прийнятому векторі на цій позиції був заданий символ.

Кожному ненульовому елементу перевірочної матриці коду з низькою щільністю перевірок на парність приписуються дві величини: $q_{i,j}^{x}$ та $r_{i,j}^{x}$.

Величина $q_{i,j}^{x}$ є ймовірністю того, що j-й символ прийнятого вектора має значення x за інформацією, одержаною з усіх перевірок, крім i-ї. Величина $r_{i,j}^{x}$ є ймовірністю того, що перевірка i виконується, якщо j-й символ прийнятого вектора дорівнює x.

Перед початком роботи алгоритму потрібна ініціалізація, далі алгоритм працює за принцип підрахунку ймовірностей символів прийнятого вектора, використовуючи для перерахування ймовірностей правило Байєса для апостеріорної ймовірності події.

Одна ітерація алгоритм є наступною послідовністю дій:

1) для всіх перевірок обчислюються величини $\Delta r_{i,j}$ і перераховуються ймовірності $r_{i,j}^{x}$ для $x = \{0,1\}$;

2) Для всіх символів прийнятого вектора перераховуються ймовірності $q_{i,j}^{x}$

3) Формуються вектори псевдо апостеріорної ймовірності q_j^0 та q_j^1.

4) формується вектор рішення c′ за таким правилом: $c_j{}' = 1$, якщо q.

Якщо вектор c′ є кодовим словом, декодування закінчується, інакше виконується така ітерація алгоритму.

Складність даного алгоритму вища, ніж складність "жорсткого"декодування інвертуванням бітів, але якість декодування підвищується за рахунок використання додаткової інформації на виході каналу однак точність роботи такого алгоритму залежить від ініціалізації: що точніше вона зроблена, то точніше буде кінцевий результат. Для каналу з Гаусівським шумом ініціалізація може бути зроблена за допомогою інформації про дисперсію шуму у каналі. для інших розподілів шуму в каналі або за невідомих характеристик шуму точна ініціалізація алгоритму може виявитися складним завданням.

2.10 Швидке декодування LDPC

Незважаючи на те, що декодування перерахунком ймовірностей є ефективним методом для каналів з безперервним виходом, той факт, що складність його значно вища, чим складність «жорсткого» декодування створює передумови для пошуку більш швидких алгоритмів декодування, що мають прийнятну якість.

Серед відомих алгоритмів швидкого декодування кодів із низькою щільністю перевірок на парність для каналів з безперервним виходом найбільш відомий алгоритм min-sum, що є спрощенням декодера "belief propagation", а також алгоритм UMP (Uniformly Most Powerful).

Складність декодера UMP (швидкого декодування за надійностями) значно нижча, чим складність декодера, що перераховує ймовірності, за рахунок того, що перерахунок надійностей виконується за спрощеною схемою (схемою «зваженого» мажоритарного голосування, як «ваг» використовується надійність перевірок), а також за рахунок можливості використання виключно цілих операцій складання та додавання по модулю два. Також до перевагам швидкого декодера за надійностями можна віднести те, що декодеру не потрібно знати характеристики шуму в каналі (дисперсію і т. д.), отже, такий декодер може працювати у будь-якому симетричному каналі з двійковим входом.

Недоліком швидкого декодера за надійностями є оцінка ймовірності помилки декодування, яке для каналу з адитивним гауссівським шумом виявляється на 0,5 дБ гірше, ніж ймовірність помилки декодування ймовірнісного декодера.

2.11 Багатопорогове декодування

Основна ідея багато порогового декодування по надійностям полягає в тому, щоб змінювати значення порогів інвертування символів від однієї ітерації до іншої наступним чином: на перших ітераціях поріг інвертування символів вибирається так, щоб кількість інвертованих символів було мінімальним (аж до інвертування лише одного символу на першій ітерації); на наступних ітераціях пороги інвертування поступово підвищуються.

При багато пороговому декодуванні, якщо на першій ітерації було виправлено хоча б одна помилка, декодування на наступних ітераціях стає значно простіше та загальне якість декодування покращується. Як і раніше, для роботи декодеру не потрібна інформація про шум у каналі, достатньо лише задати надійності.

Декодер, що працює за багатопороговою схемою, дозволяє отримати ймовірність помилки декодування на 0,1–0,4 дБ краще, ніж забезпечує швидкий декодер за надійностями UMP, практично наближаючись до

ймовірності помилки, що отримується при ймовірнісному декодуванні кодів із низькою щільністю перевірок на парність. Крім незалежності від характеристик каналу багатопороговий декодер має властивість декодерів кодів з низькою щільністю перевірок на парність, а саме універсальністю та застосовністю для будь-якої конструкції таких кодів.

Слід зазначити, що ефективність нерегулярних LDPC-кодів виявляється вищою. Ефективності регулярних кодів це пояснюється тим, що в нерегулярних кодах через різні числа одиниць у рядках і стовпцях інформаційні символи захищені по-різному. в результаті при декодуванні проявляється так званий ефект хвилі, коли більш захищені біти декодуються швидше і потім допомагають при декодуванні менш захищених біт.

2.12 Оцінка ефективності недвійкових LDPC-кодів

Схема моделі наведено на рис. 2.9. Для моделювання модему, каналу зв'язку та кодеку Рида–Соломона використовували стандартні об'єкти та функції Matlab. Декодування недвійкових LDPC–кодів проводилося за допомогою алгоритму FFTQSPA з максимальною кількістю ітерацій декодування, рівним 20. Було проведено порівняння коротких ($N \approx 120$), середніх ($N \approx 250$) та довгих ($N \approx 500$) кодів.

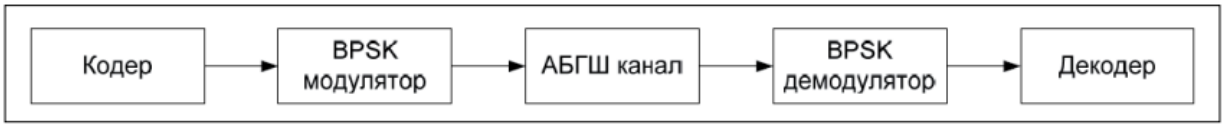

Рисунок 2.9 – Схема моделі

При аналізі коротких кодів порівнювалися недвійкові коди LDPC $N = 120$, $K = 71$, $R = 0{,}5917$, символи коду належать $GF(16)$, і коди Рида–Соломона $T = 127$, $K = 77$, $R = 0{,}6062$, символи коду належать $GF(128)$. Результати моделювання цих кодів наведено на рис. 2.10. Як видно з малюнка, недвійкові LDPC-коди значно перевершують коди Ріда-Соломона. При ймовірності помилки на біт 10-5 це перевага становить 1,6 дБ.

При аналізі кодів, що мають середню довжину блоку, порівнювалися недвійкові коди LDPC $N = 248, K = 137, R = 0{,}5524$, символи коду належать $GF(32)$, і коди Ріда–Соломона $N = 255$, $K = 141$, $R = 0{,}5529$, символи коду належать $GF(256)$. Результати моделювання цих кодів наведено на рисунку 2.11. При заданій довжині блоку при ймовірності помилки на біт 10^{-5} перевага NB-LDPC становить 2 дБ.

При аналізі довгих кодів порівнювалися недвійкові коди LDPC $N = 504$, $K = 267$, $R = 0{,}5297$, символи коду належать $GF(64)$, і коди Ріда–Соломона $N = 511$, $K = 271$, $R = 0{,}5303$ символи коду належать $GF(512)$. Результати моделювання цих кодів наведено на рис. 2.12. При ймовірності помилки на біт 10-5 перевага NB-LDPC становить 2,4 дБ.

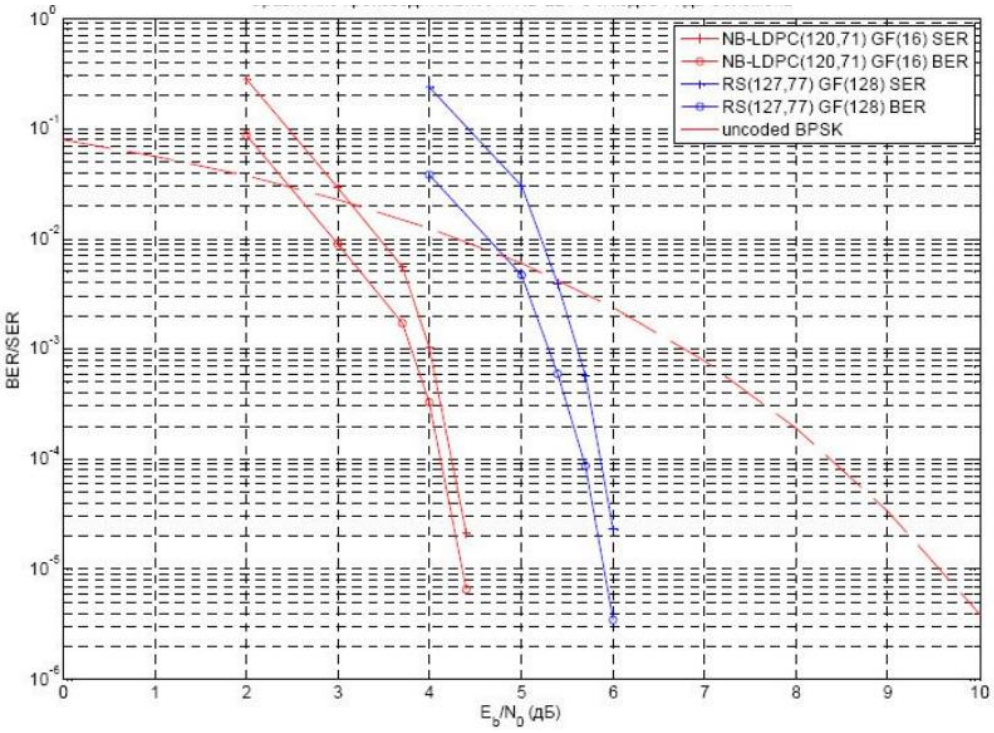

Рисунок 2.10 – Порівняння продуктивності NB-LDPC та кодів Ріда-Соломона при невеликий довжині блоку

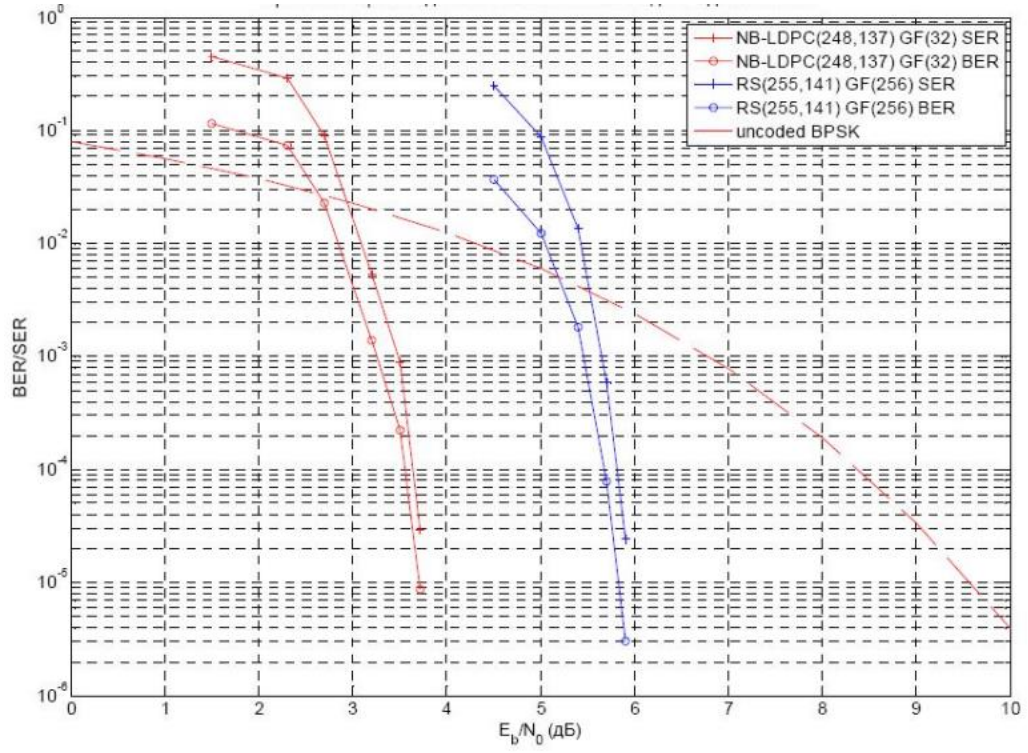

Рисунок 2.11 – Порівняння продуктивності NB-LDPC та кодів Ріда-Соломона при середній довжині блоку

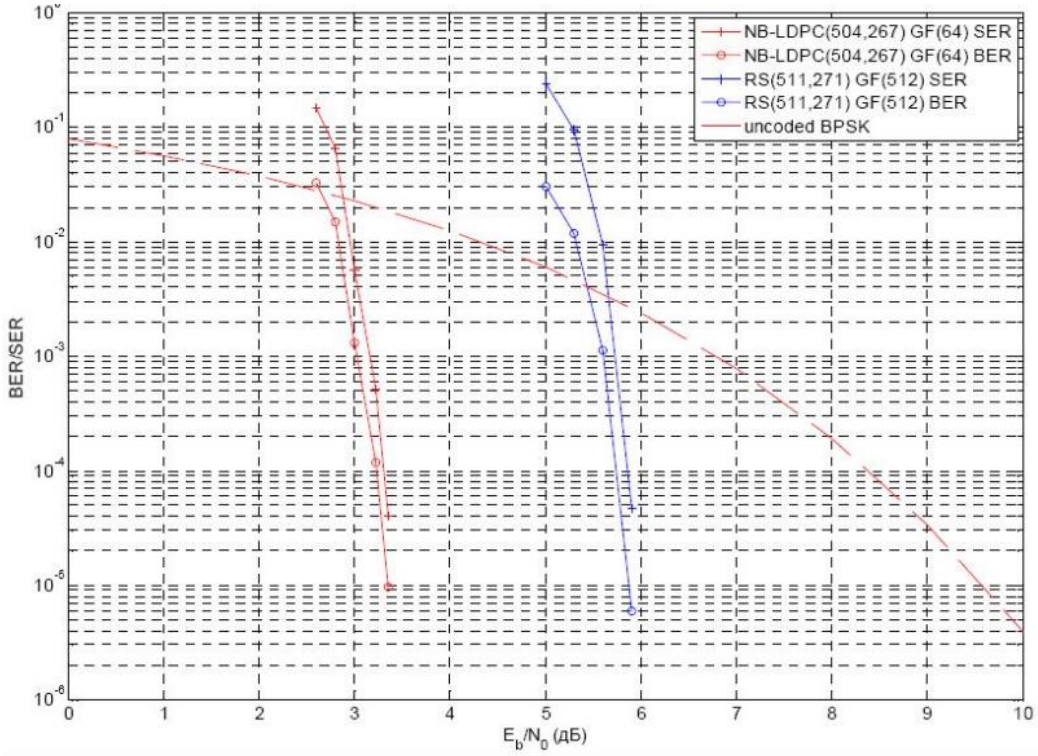

Рисунок 2.11 – Порівняння продуктивності NB-LDPC та кодів Ріда-Соломона при великій довжині блоку

2.13 Висновки до розділу 2

У 2 розділі було розглянуто різні методи та декодування, такі як:

- Жорске декодування.
- Декодування за ймовірністями.
- Швидке декодування.
- Багатопорогове декодування.

Була дана оцінка ефективності недвійкових LDPC-кодів. При аналізі коротких кодів ймовірності помилки на біт 10-5 це перевага становить 1,6 дБ. При аналізі кодів, що мають середню довжину блоку ймовірності помилки на біт 10^{-5} перевага NB-LDPC становить 2 дБ. При аналізі довгих кодів ймовірності помилки на біт 10-5 перевага NB-LDPC становить 2,4 дБ.

РОЗДІЛ 3
РОЗРОБКА ПРОГРАМНОЇ ЧАСТИНИ

Проєктування програмного забезпечення – етап життєвого циклу програмного забезпечення, під час якого досліджується структура і взаємозв'язки елементів системи, що розробляється. Результатом цього етапу є, насамперед, декодуванная LDPC, а саме отримання інвертованого слова шляхом інвертування. Для реалізації застосунку було обрано рішення використовувати Next.js.

Next.js – це гнучкий фреймворк React, який дає змогу створювати швидкі веб–додатки завдяки окремим “будівельним” блокам.

Є кілька речей, які потрібно враховувати при створенні сучасних програм. Такі як:

1) Інтерфейс користувача – як користувачі будуть споживати програму та взаємодіяти з нею;

2) Маршрутизація – як користувачі переміщуються між різними частинами програми;

3) Отримання даних – де знаходяться дані та як їх отримати;

4) Відтворення – коли і де ви відтворюєте статичний або динамічний вміст;

5) Інтеграції – які сторонні сервіси ви використовуєте (CMS, авторизація, платежі тощо) і як ви підключаєтеся до них;

6) Інфраструктура – де ви розгортаєте, зберігаєте та запускаєте код програми (безсерверний, CDN, Edge тощо);

7) Продуктивність – як оптимізувати вашу програму для кінцевих користувачів;

8) Масштабованість – як ваша програма адаптується до зростання вашої команди, даних і трафіку;

9) Досвід розробника – досвід вашої команди у створенні та підтримці вашої програми.

React – це бібліотека JavaScript для створення інтерактивних інтерфейсів користувача. Під користувацькими інтерфейсами мається на увазі елементи, які користувачі бачать і взаємодіють з ними на екрані.

Під бібліотекою мається на увазі, що React надає корисні функції для побудови інтерфейсу користувача, але залишає розробнику вирішувати, де використовувати ці функції у своїй програмі.

Частиною успіху React є те, що він відносно неосмислений щодо інших аспектів створення програм. Це призвело до процвітання екосистеми сторонніх інструментів і рішень. Однак це також означає, що створення повної програми React з нуля вимагає певних зусиль. Розробникам доводиться витрачати час на налаштування інструментів і переосмислення рішень для загальних вимог додатків.

Next.js – це вже фреймворк React, який дає вам будівельні блоки для створення веб-додатків. Next.js обробляє інструменти та конфігурацію, необхідні для React, і надає додаткову структуру, функції та оптимізацію для програми.

3.1 Ініціалізація проєкту

Щоб почати працювати нам необхідно встановити Node.js. Для цього нам необхідно перейти на офіційний сайт та завантажити LTS або Current версію. Рекомендується обирати саме LTS, хоч вона і має меншу версію але вона є більш стабільною ніж Current версія.

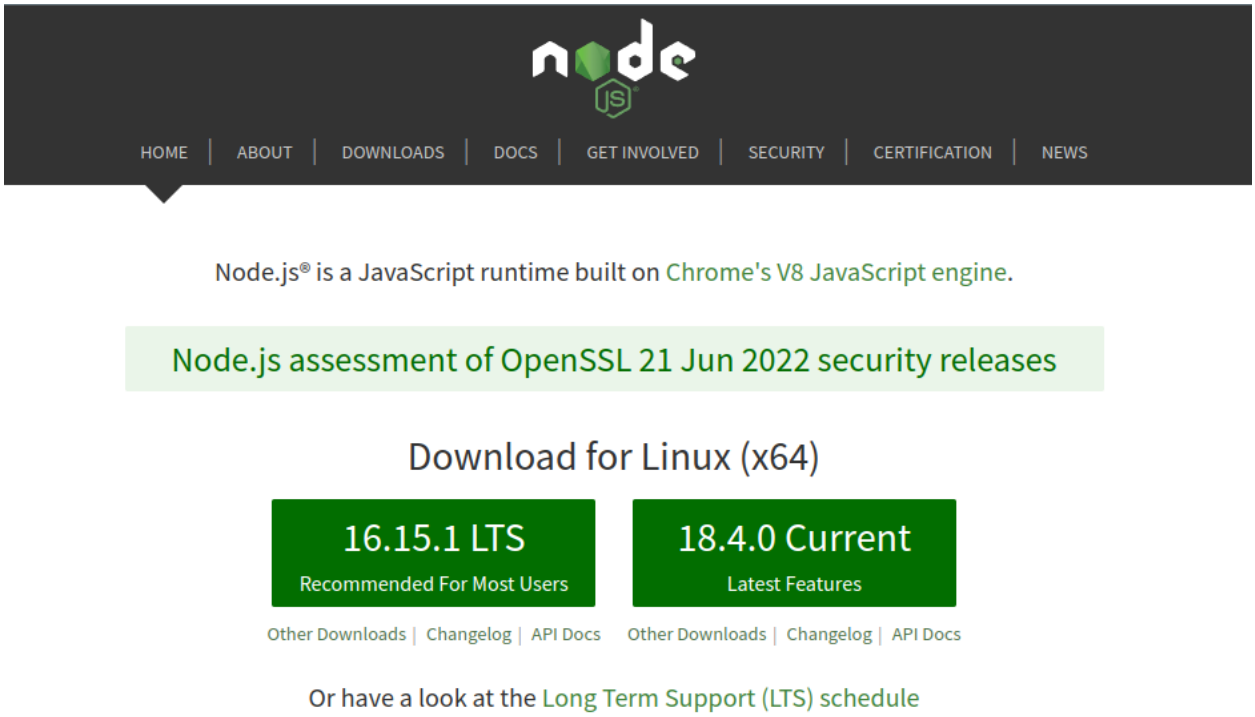

Рисунок 3.1 – Завантаження node.js

Середою розробки оберемо Visual Studio Code, також відомий VS Code, – це редактор вихідного коду, створений Microsoft для Windows, Linux і mac OS. Функції включають підтримку налагодження, виділення синтаксису, інтелектуальне завершення коду, фрагменти, рефакторинг коду та вбудований Git.

Для того щоб створити стартовий застосунок потрібно відкрити термінал та прописати одну з команд на рис. 3.2.

```
npx create-next-app@latest
# or
yarn create next-app
# or
pnpm create next-app
```

Рисунок 3.2 – Створення застосунку Next.js

3.2 Запуск застосунку

Для того щоб запустити проєкт нам необхідно відкрити термінал та ввести одну з наступних команд, залежно від того який пакет використовуємо.

```
user@NB-108351:~/Public/decoder$ npm run dev
```

Рисунок 3.2 – Запуск з npm

```
user@NB-108351:~/Public/decoder$ yarn dev
```

Рисунок 3.3 – Запуск з yarn

Після запуску однієї з команд нам буде доступна url адреса (http://localhost:3000) за якою ми можемо перейти у браузер.

```
user@NB-108351:~/Public/decoder$ npm run dev

> decoder@0.1.0 dev
> next dev

ready - started server on 0.0.0.0:3000, url: http://localhost:3000
wait  - compiling...
event - compiled client and server successfully in 309 ms (128 modules)
```

Рисунок 3.4 – Запуск з yarn

Коли ми перейдемо на http://localhost:3000, но ми побачимо базову сторінку Next.js затосунку з різними корисними посиланнями.

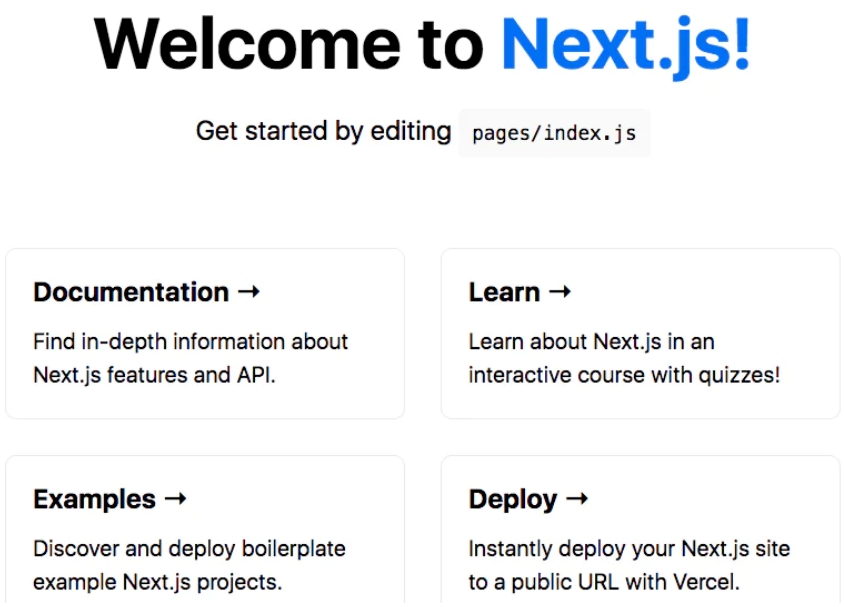

Рисунок 3.5 – Базова сторінка Next.js

3.3 Візуальна складова

Для полегшення роботи візуальною складовою будемо використовувати Ant Design, це бібліотека різноманітних компонентів у React (від простих кнопок до складних форм та шаблонів).

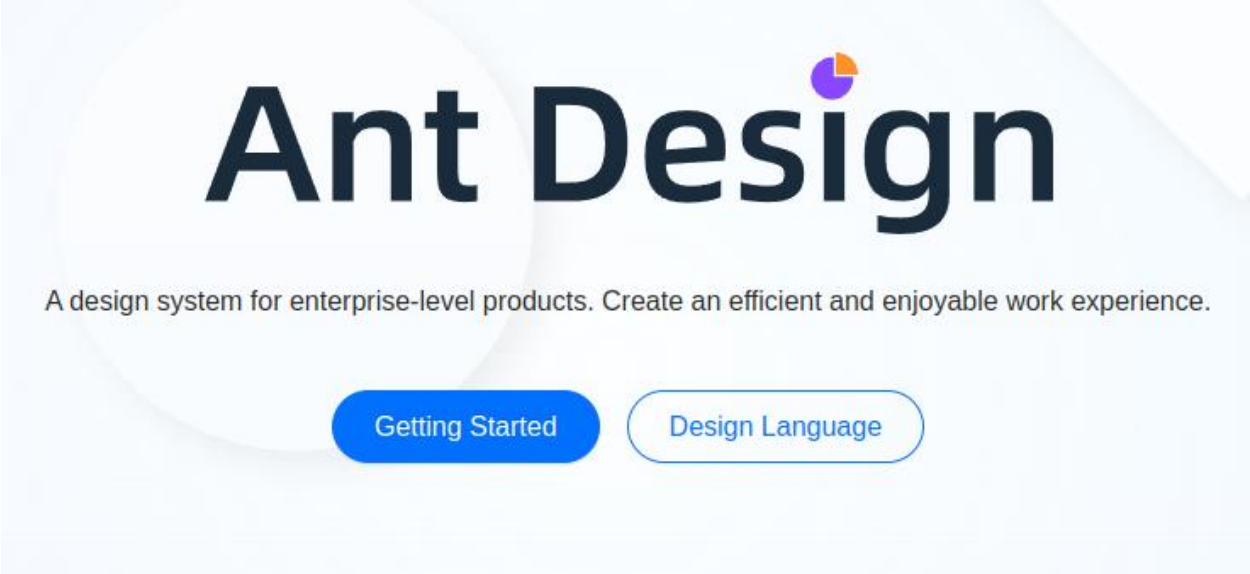

Рисунок 3.6 – Ant Design

Щоб додати цю бібліотеку треба скористатись однією с наступних команд.

```
$ npm install antd
```

```
$ yarn add antd
```

Рисунок 3.7 – Команди для завантаження Ant Design

Далі щоб використати необхідний компонент його треба імпортувати з бібліотеки наступним чином.

```
import { Button } from "antd";
```

Рисунок 3.8 – Імпорт компоненту

3.4 Структура папок

Структура нашого проєкту виглядає наступним чином

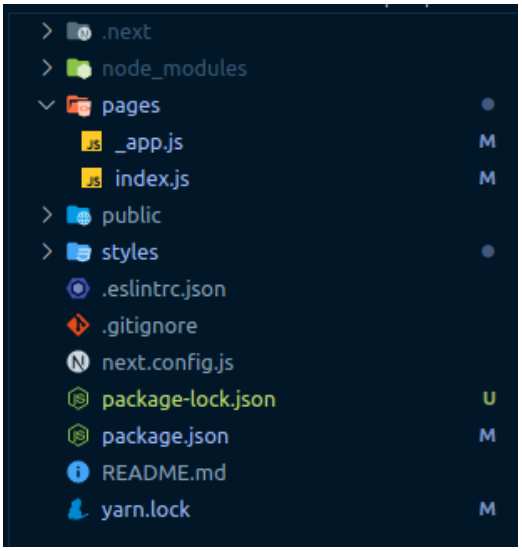

Рисунок 3.9 – Структура папок

У папці pages зберігаються всі доступні сторінки, у нашому випадку нам знадобиться лише одна домашня сторінка за яку відповідає файл index.js.

У папці public та styles знаходяться іконки та css стилі відповідно.

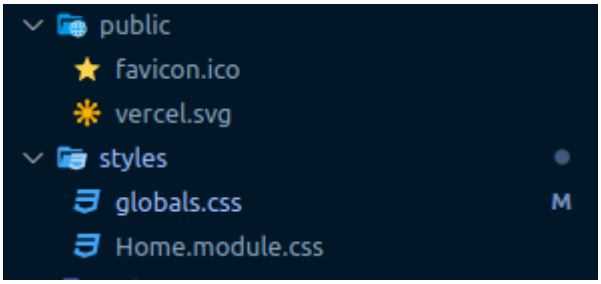

Рисунок 3.10 – Папки public та styles

У файлі package.json можна переглянути доступні команди за встановлені залежності в проєкті.

```
{
  "name": "decoder",
  "version": "0.1.0",
  "private": true,
  Отладка ▷
  "scripts": {
    "dev": "next dev",
    "build": "next build",
    "start": "next start",
    "lint": "next lint"
  },
  "dependencies": {
    "antd": "^4.21.3",
    "next": "12.1.6",
    "react": "18.2.0",
    "react-dom": "18.2.0"
  },
  "devDependencies": {
    "eslint": "8.18.0",
    "eslint-config-next": "12.1.6"
  }
}
```

Рисунок 3.11 – Файл package.json

3.5 Реалізація декодування LDPC

Так як у нас невеликий застосунок було прияняте рішення зробити всю програму на одній сторінці. а саме в $/pages/index.js$, де і знаходиться увесь основний код.

3.5.1 Створення матриці

Імпортуємо необхідні компоненти з Ant Design.

- Button – необхіден для інтерактивності застосунку, завдяки натисканням на кнопки відбуваються різні дії;
- Select – необхіден для створення та візуалізації матриці;
- Divider – необхіден для розмежування елементів на секції;
- Layuot – створює сітку для всього контенту на сторінці.

```
import { Button, Select, Divider, Layout } from "antd";

const { Header, Content, Footer } = Layout;

const { Option } = Select;
```

Рисунок 3.12 – Імпорт компонентів

Завдяки коду на на рис 3.13 користувач може обрати яку матрицю він хоче створитию Буде створена матриця з обраним значенням.

```
<Select
  style={{ width: 140 }}
  onChange={createTable}
  placeholder="Обрати розмір"
>
  <Option value="4">4</Option>
  <Option value="5">5</Option>
  <Option value="6">6</Option>
  <Option value="7">7</Option>
  <Option value="8">8</Option>
  <Option value="9">9</Option>
  <Option value="10">10</Option>
  <Option value="11">11</Option>
  <Option value="12">12</Option>
  <Option value="13">13</Option>
  <Option value="14">14</Option>
</Select>
```

Рисунок 3.13 – Вибір розміру матриці

У браузері це виглядає наступним чином

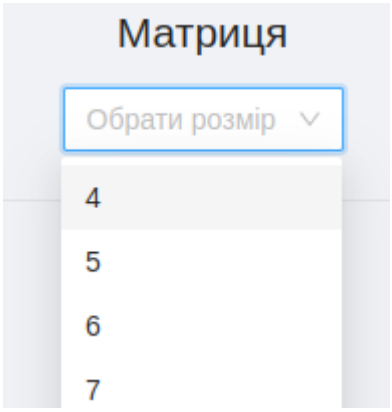

Рисунок 3.14 – Вибір розміру матриці у браузері

Коли ми обираємо розмір то завдяки функції createTable ми створюємо новий масив який заповнюємо нулями та рендеремо його користувачу у вигляді таблиці.

```
const [matrix, setMatrix] = useState([]);

let createTable = (value) => {
  let newMatrix = Array(Number(value))
    .fill()
    .map(() => Array(Number(value)).fill(0));

  setMatrix(newMatrix);
};
```

Рисунок 3.15 – Функція createTable

```
<table className="matrix__table">
  <tbody>
    {matrix.map((row, rowIndex) => (
      <tr key={rowIndex}>
        {row.map((number, cellIndex) => (
          <td key={cellIndex} onClick={handleMatrixCell}>
            {number}
          </td>
        ))}
      </tr>
    ))}
  </tbody>
</table>
```

Рисунок 3.16 – Рендер матриці

Створена матриця буде представлена користувачу у вигляді таблиці значень цієї матриці. Користувач може взаємодіяти с матрицею натискаючи на її елементи.

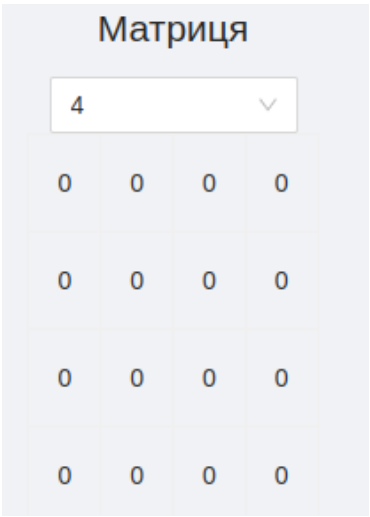

Рисунок 3.17 – Матриця 4 на 4

Далі, змінюючи значення 0 на 1 і навпаки, користувач може заповнити матрицю так, як йому заманеться. За цей функционал відповідає функція handleMatrixCell. Перевіряється чому рівен обраний елемент, якщо 1 до

змінюємо його на 0 і навпаки. Потім створюємо новий масив який згодом використовуємо як нову матрицю.

```
const handleMatrixCell = (event) => {
  let matrixTable = document.querySelector(".matrix__table");

  let cells = matrixTable.getElementsByTagName("td");

  if (event.target.textContent == 0) {
    event.target.textContent = 1;
  } else {
    event.target.textContent = 0;
  }

  let cellsArray = [];

  for (let i = 0; i < cells.length; i++) {
    cellsArray.push(Number(cells[i].textContent));
  }

  let subArray = [];
  let size = matrix.length;

  for (let i = 0; i < Math.ceil(cellsArray.length / size); i++) {
    subArray[i] = cellsArray.slice(i * size, i * size + size);
  }

  setMatrix(subArray);
};
```

Рисунок 3.18 – Функція handleMatrixCell

3.5.2 Кодове слово

В залежності від розміру своренної матриці буде сформоване кодове слово з такою ж довжиною. На рис. 3.19 ми оновлюємо кодове слово кожен раз коли змінюємо розмір початкової матриці.

```
useEffect(() => {
  setCodeWord(Array(Number(matrix.length)).fill(0));
}, [matrix.length]);
```

Рисунок 3.19 – Оновлення кодового слова

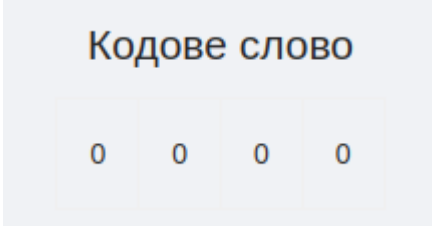

Рисунок 3.20 – Рендер кодового слова

Для цього елемента ми також здатні змінювати значення між 0 та 1.

```
const handleCodeWordCell = (event) => {
  let codeWordTable = document.querySelector(".code-word__table");

  let cells = codeWordTable.getElementsByTagName("td");

  if (event.target.textContent == 0) {
    event.target.textContent = 1;
  } else {
    event.target.textContent = 0;
  }

  let cellsArray = [];

  for (let i = 0; i < cells.length; i++) {
    cellsArray.push(Number(cells[i].textContent));
  }

  setCodeWord(cellsArray);
};
```

Рисунок 3.21 – Функція handleCodeWordCell

3.5.3 Виявлення синдрому

Після того як ми з матрицею та кодовим словом нам стане доступна секція зі знаходженням синдрому. Код можна переглянути у додатку А. Синдром показує нам у якому біті кодового слова є помилка.

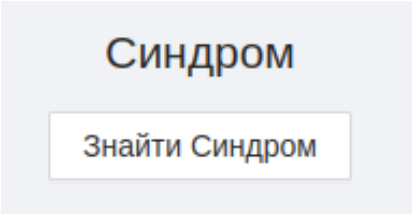

Рисунок 3.22 – Секція для знакодження синдрому

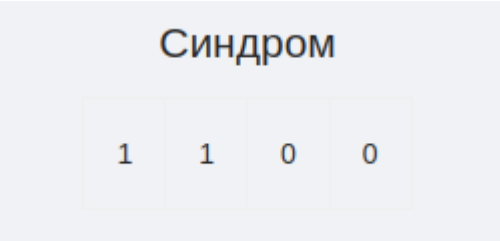

Рисунок 3.23 – Створений синдрому

Після цього нам стане доступна табличка, завдяки якій ми можемо зрозуміти які біти у кодовому слові нам потрібно інвертувати. Наприклад у колонці $F0$ ми бачимо $X2$, це означає що нам потрібно інвертувати елемент кодового слова з індексом 2, тобто третій елемент слова, тому що індекс починається з 0. Але нам потрібно інвертувати елемент лише один раз, тобто

якщо ми вже інверували $X2$ то, якщо ми зустрінемо цей індекс далі, ми не будемо його інвертувати знову.

XOR

F0	F1	F2	F3
X2	X1 XOR X3	X0 XOR X2	X0 XOR X1 XOR X3

Рисунок 3.24 – XOR таблиця

3.5.4 Інвертоване слово

Після того як ми пройшли всі вищезазначені операції ми можемо отримати інвертоване слово. Проходимось циклом по синдрому, якщо елемент дорівнює 1 ми можемо працювати далі. Якщо ця умова дійсна то далі ми використовуємо таблицю XOR з якої ми беремо індех бітів кодового слова які необхідно інвертувати, але тільки якщо цей елемент вже не був інвертований.

```
const invertWord = () => {
  let newCode = codeWord;

  for (let i = 0; i < syndrome.length; i++) {
    if (syndrome[i] === 1) {
      for (let j = 0; j < indexArray[i].length; j++) {
        if (codeWord[indexArray[i][j]] === 1) {
          newCode[indexArray[i][j]] = 0;
        } else {
          newCode[indexArray[i][j]] = 1;
        }
      }
    }
  }

  setInvertedWord(newCode);
};
```

Рисунок 3.25 – Функція знаходження кодового слова

```
<table>
  <tbody>
    <tr>
      {invertedWord.map((item, index) => (
        <td key={index}>{item}</td>
      ))}
    </tr>
  </tbody>
</table>
```

Рисунок 3.26 – Рендер кодового слова

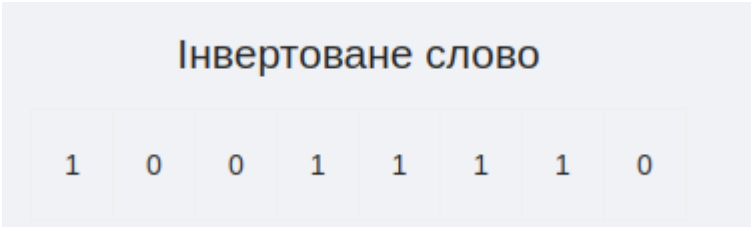

Рисунок 3.27 – Фінальний варіант кодового слова

3.6 Висновки до розділу 3

У 3 розділі спроектовано систему для декодування LDPC–кодів шляхом виявлення синдрому та інвертуванням кодового слова, за допомогою бібліотек javascript. Так як проєкт побудований завдяки Nex.js, це дає змогу для швидкого маштабування та розширення системи, зокрема додавання серверної частини та API. Front-end частина використовує новітній фреймворк, тому виконує поставленні задачі максимально чітко та швидко, що дає новітній та легкодоступний інтерфейс для користувача. Працездатність застосунку перевірено, експлуатації даної програми.

ВИСНОВКИ

Метою бакалаврської кваліфікаційної роботи є розробка системи моделювання роботи LDPC-кодів для дослідження характеристик таких кодів.

При виконанні завдання було реалізовані наступні етапи:

- опис предметної області;
- огляд та аналіз бібліотек;
- постановка задачі;
- вибір технології створення вебзастосунків;
- вибір найкращих інструментів та методології для вирішення поставленої задачі;
- проєктування декодування LDPC-кодів;
- розробка структури вебзастосунку;
- опис програмної реалізації;
- проведення тестування.

У першому розділі був зроблений аналітичний огляд літератури та патентної інформації з декодування LDPC-кодів, проведено аналіз предметної області.

У другому розділі було проаналазовано методи та алгоритми декодування та оцінка ефективності LDPC-кодів.

У третьому розділу було спроєктовано взаємодії користувача з вебзастосунком.

У розробленому вебзастосуноку реалізовано мінімалістичний та зручний інтерфейс для користування.

Функціональні можливості розробленого застосунку:

- Створення матриці, та заповнення її бітами.
- Створення кодового за рахунок довжини матриці, та заповнення її бітами.
- Знаходження синдрому для декодування.
- Знаходження варіантів інвертування для бітів кодового слова.

- Знаходження інвертованого кодового слова.

Серед інструментів обрано:

1. Фреймворк Next.js;
2. Кодовий редактор Visual Studio Code.

Враховуючи вищенаведене, всі задачі роботи було виконано, а мету роботи, що полягала у створенні системи моделювання роботи LDPC-кодів для дослідження характеристик таких кодів досягнуто.

У спеціальному розділі з охорони праці було розглянуто питання охорони праці в університеті «Чорноморський національний університет імені Петра Могили», виконана інтегральна оцінка умов праці, проведені заходи, спрямовані на їх покращення.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. David J.C. MacKay (2003) Information theory, Inference and Learning Algorithms, CUP, ISBN 0-521-64298-1.

2. Todd K. Moon (2005) Error Correction Coding, Mathematical Methods and Algorithms. Wiley, ISBN 0-471-64800-0.

3. Amin Shokrollahi (2003) LDPC Codes: An Introduction.

4. US 5446747.

5. NewScientist, Communication speed nears terminal velocity, by Dana Mackenzie, 9 July 2005.

6. Larry Hardesty (January 21, 2010). "Explained: Gallager codes". MIT News. Retrieved August 7, 2013.

7. R. G. Gallager, "Low density parity check codes," IRE Trans. Inf. Theory, vol. IT-8, no. 1, pp. 21- 28, Jan. 1962.

8. J. Moshieff, N. Resch, N. Ron-Zewi, S. Silas, M. Wootters, "Low-density parity-check codes achieve list-decoding capacity," SIAM Journal on Computing, FOCS20-38-FOCS20-73.

9. Robert G. Gallager (1963). Low Density Parity Check Codes (PDF). Monograph, M.I.T. Press. Retrieved August 7, 2013.

10. David J.C. MacKay and Radford M. Neal, "Near Shannon Limit Performance of Low Density Parity Check Codes," Electronics Letters, July 1996.

11. Telemetry Data Decoding, Design Handbook.

12. Presentation by Hughes Systems Archived 2006-10-08 at the Wayback Machine.

13. HomePNA Blog: G.hn, a PHY For All Seasons.

14. IEEE Communications Magazine paper on G.hn Archived 2009-12-13 at the Wayback Machine.

15. IEEE Standard, section 20.3.11.6 "802.11n-2009", IEEE, October 29, 2009, accessed March 21, 2011.

16. "IEEE SA - IEEE 802.11ax-2021". IEEE Standards Association. Retrieved May 22, 2022.

17. Chih-Yuan Yang, Mong-Kai Ku. "LDPC coded OFDM modulation for high spectral efficiency transmission".

18. Nick Wells. "DVB-T2 in relation to the DVB-x2 Family of Standards" Archived 2013-05-26 at the Wayback Machine.

19. "5G Channel Coding" (PDF). Archived from the original (PDF) on December 6, 2018. Retrieved January 6, 2019.

20. Maunder, Robert (September 2016). "A Vision for 5G Channel Coding" (PDF). Archived from the original (PDF) on December 6, 2018. Retrieved January 6, 2019.

21. A.I. Vila Casado, M. Griot, and R.Wesel, "Informed dynamic scheduling for belief propagation decoding of LDPC codes," Proc. IEEE Int. Conf. on Comm. (ICC), June 2007.

22. T. Richardson, "Error floors of LDPC codes," in Proc. 41st Allerton Conf. Comm., Control, and Comput., Monticello, IL, 2003.

23. Thomas J. Richardson and M. Amin Shokrollahi and Rüdiger L. Urbanke, "Design of Capacity-Approaching Irregular Low-Density Parity-Check Codes," IEEE Transactions on Information Theory, 47(2), February 2001.

24. Thomas J. Richardson and Rüdiger L. Urbanke, "Efficient Encoding of Low-Density Parity-Check Codes," IEEE Transactions on Information Theory, 47(2), February 2001.

25. Ahmad Darabiha, Anthony Chan Carusone, Frank R. Kschischang. "Power Reduction Techniques for LDPC Decoders".

26. Zhengya Zhang, Venkat Anantharam, Martin J. Wainwright, and Borivoje Nikolic. "An Efficient 10GBASE-T Ethernet LDPC Decoder Design With Low Error Floors".

27. Y. Kou, S. Lin and M. Fossorier, "Low-Density Parity-Check Codes Based on Finite Geometries: A Rediscovery and New Results," IEEE Transactions on Information Theory, vol. 47, no. 7, November 2001, pp. 2711- 2736.

28. Tahir, B., Schwarz, S., & Rupp, M. (2017, May). BER comparison between Convolutional, Turbo, LDPC, and Polar codes. In 2017 24th International Conference on Telecommunications (ICT) (pp. 1-7). IEEE.

29. Moon Todd, K. Error correction coding: mathematical methods and algorithms. 2005 by John Wiley & Sons. ISBN 0-471-64800-0. - p.614.

30. Moon Todd, K. Error correction coding: mathematical methods and algorithms. 2005 by John Wiley & Sons. ISBN 0-471-64800-0. - p.653.

31. Andrews, Kenneth S., et al. "The development of turbo and LDPC codes for deep-space applications." Proceedings of the IEEE 95.11 (2007): 2142-2156.

32. Hassan, A.E.S., Dessouky, M., Abou Elazm, A. and Shokair, M., 2012. Evaluation of complexity versus performance for turbo code and LDPC under different code rates. Proc. SPACOMM, pp.93-98.

33. "IEEE Spectrum: Does China Have the Best Digital Television Standard on the Planet?". spectrum.ieee.org. Archived from the original on December 12, 2009.

ДОДАТОК А

КОД ПРОГРАМИ REACT

```
import React, { useEffect, useState } from "react";

import { Button, Select, Divider, Layout } from "antd";

const { Header, Content, Footer } = Layout;

const { Option } = Select;

const Home = () => {
  const [matrix, setMatrix] = useState([]);

  const createTable = (value) => {
    let newMatrix = Array(Number(value))
      .fill()
      .map(() => Array(Number(value)).fill(0));

    setMatrix(newMatrix);
  };

  const handleMatrixCell = (event) => {
    let matrixTable = document.querySelector(".matrix__table");

    let cells = matrixTable.getElementsByTagName("td");

    if (event.target.textContent == 0) {
      event.target.textContent = 1;
    } else {
      event.target.textContent = 0;
    }

    let cellsArray = [];

    for (let i = 0; i < cells.length; i++) {
      cellsArray.push(Number(cells[i].textContent));
    }

    let subArray = [];
    let size = matrix.length;

    for (let i = 0; i < Math.ceil(cellsArray.length / size); i++) {
      subArray[i] = cellsArray.slice(i * size, i * size + size);
    }

    setMatrix(subArray);
  };

  const [dataSubArray, setDataSubArray] = useState([]);

  useEffect(() => {
    setCodeWord(Array(Number(matrix.length)).fill(0));
  }, [matrix.length]);

  const [codeWord, setCodeWord] = useState([]);
```

```
const handleCodeWordCell = (event) => {
  let codeWordTable = document.querySelector(".code-word__table");

  let cells = codeWordTable.getElementsByTagName("td");

  if (event.target.textContent == 0) {
    event.target.textContent = 1;
  } else {
    event.target.textContent = 0;
  }

  let cellsArray = [];

  for (let i = 0; i < cells.length; i++) {
    cellsArray.push(Number(cells[i].textContent));
  }

  setCodeWord(cellsArray);
};

const [syndrome, setSyndrome] = useState([]);

const findSyndrome = () => {
  let newArray = [];
  let size = matrix.length;

  for (let i = 0; i < matrix.length; i++) {
    for (let j = 0; j < matrix.length; j++) {
      newArray.push(matrix[j][i]);
    }
  }

  let subArray = [];

  for (let i = 0; i < Math.ceil(newArray.length / size); i++) {
    subArray[i] = newArray.slice(i * size, i * size + size);
  }

  setDataSubArray(subArray);

  for (let i = 0; i < subArray.length; i++) {
    let syndromeSubArray = [];

    for (let j = 0; j < subArray[i].length; j++) {
      if (subArray[i][j] === 1 && codeWord[j] === 1) {
        syndromeSubArray.push(1);
      }
    }

    setSyndrome((oldArray) => [
      ...oldArray,
      syndromeSubArray.length === 1 ? 1 : 0,
    ]);
  }
};

const [indexArray, setIndexArray] = useState([]);
```

```
  let findFn = () => {
    for (let i = 0; i < dataSubArray.length; i++) {
      let array = [];

      for (let j = 0; j < dataSubArray[i].length; j++) {
        if (dataSubArray[i][j] === 1) {
          array.push(j);
        }
      }

      setIndexArray((oldArray) => [...oldArray, array]);
    }
  };

  const [invertedWord, setInvertedWord] = useState([]);

  const invertWord = () => {
    let newCode = codeWord;

    for (let i = 0; i < syndrome.length; i++) {
      if (syndrome[i] === 1) {
        for (let j = 0; j < indexArray[i].length; j++) {
          if (codeWord[indexArray[i][j]] === 1) {
            newCode[indexArray[i][j]] = 0;
          } else {
            newCode[indexArray[i][j]] = 1;
          }
        }
      }
    }

    setInvertedWord(newCode);
  };

  return ();
};

export async function getStaticProps() {
  return {
    props: {},
  };
}

export default Home;
```

ДОДАТОК Б

КОД ПРОГРАМИ JSX

```
<Layout className="layout">
  <Header>LDPC Decoder</Header>

  <Content>
    <div className="site-layout-content">
      <h2>Матриця</h2>

      <Select
        style={{ width: 140 }}
        onChange={createTable}
        placeholder="Обрати розмір"
      >
        <Option value="4">4</Option>
        <Option value="5">5</Option>
        <Option value="6">6</Option>
        <Option value="7">7</Option>
        <Option value="8">8</Option>
        <Option value="9">9</Option>
        <Option value="10">10</Option>
        <Option value="11">11</Option>
        <Option value="12">12</Option>
        <Option value="13">13</Option>
        <Option value="14">14</Option>
      </Select>

      <table className="matrix__table">
        <tbody>
          {matrix.map((row, rowIndex) => (
            <tr key={rowIndex}>
              {row.map((number, cellIndex) => (
                <td key={cellIndex} onClick={handleMatrixCell}>
                  {number}
                </td>
              ))}
            </tr>
          ))}
        </tbody>
      </table>

      <Divider />

      {matrix.length > 0 && (
        <>
          <h2>Кодове слово</h2>

          <table className="code-word__table">
            <tbody>
              <tr>
                {codeWord.map((item, index) => (
                  <td key={index} onClick={handleCodeWordCell}>
                    {item}
                  </td>
                ))}
              </tr>
```

```
            </tbody>
          </table>

          <Divider />
        </>
      )}

      {codeWord.length > 0 && (
        <>
          <h2>Синдром</h2>

          {syndrome.length > 0 ? (
            <table>
              <tbody>
                <tr>
                  {syndrome.map((item, index) => (
                    <td key={index}>{item}</td>
                  ))}
                </tr>
              </tbody>
            </table>
          ) : (
            <Button onClick={findSyndrome}>Знайти Синдром</Button>
          )}

          <Divider />
        </>
      )}

      {syndrome.length > 0 && (
        <>
          <h2>XOR</h2>

          {indexArray.length > 0 ? (
            <table>
              <thead>
                <tr>
                  {matrix.map((item, index) => (
                    <th key={index}>F{index}</th>
                  ))}
                </tr>
              </thead>

              <tbody>
                <tr>
                  {indexArray.map((row, index) => (
                    <td key={index}>
                      {row.map((item, index) => (
                        <span key={index}>
                          X{item}
                          {row.length - 1 !== index && " XOR "}
                        </span>
                      ))}
                    </td>
                  ))}
                </tr>
              </tbody>
            </table>
```

```
              ) : (
                <Button onClick={findFn}>Знайти XOR</Button>
              )}

              <Divider />
            </>
          )}

          {indexArray.length > 0 && (
            <>
              <h2>Інвертоване слово</h2>

              {invertedWord.length > 0 ? (
                <table>
                  <tbody>
                    <tr>
                      {invertedWord.map((item, index) => (
                        <td key={index}>{item}</td>
                      ))}
                    </tr>
                  </tbody>
                </table>
              ) : (
                <Button onClick={invertWord}>Знайти інвертоване
слово</Button>
              )}
            </>
          )}
        </div>
      </Content>
      <Footer style={{ textAlign: "center" }}>LDPC Decoder</Footer>
    </Layout>
```

ДОДАТОК В

ПРОЦЕС РОБОТИ ЗАСТОСУНКУ