НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені Ігоря СІКОРСЬКОГО»
Навчально-науковий фізико-технічний інститут
Кафедра математичних методів захисту інформації

«На правах рукопису»
УДК 519.651

«До захисту допущено»
В.о. завідувача кафедри
_____ Сергій ЯКОВЛЄВ
«__» _____ 2023 р.

Дипломна робота

на здобуття ступеня бакалавра
за освітньо-професійною програмою
«Математичні методи криптографічного захисту інформації»

зі спеціальності: 113 Прикладна математика
на тему: **«Аналіз ефективності варіантів алгоритму декодування Гольдрайха-Левіна»**

Виконала:
студентка IV курсу, групи ФІ-93
Мартинова Марія Євгенівна _____

Керівник:
проф. каф. ММЗІ НН ФТІ, док. тех. наук
Олексійчук Антон Миколайович _____

Рецензент:
доцент каф. ММАД, к.ф-м.н.
Терещенко Іван Миколайович _____

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.

Студентка _____

Київ — 2023

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені Ігоря СІКОРСЬКОГО»
Навчально-науковий фізико-технічний інститут
Кафедра математичних методів захисту інформації

Рівень вищої освіти — перший (бакалаврський)
Спеціальність — 113 Прикладна математика,
ОПП «Математичні методи криптографічного захисту інформації»

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри
_____ Сергій ЯКОВЛЄВ
«__» _____ 2023 р.

ЗАВДАННЯ
на дипломну роботу

Студент: Мартинова Марія Євгенівна

1. Тема роботи: «*Аналіз ефективності варіантів алгоритму декодування Гольдрайха-Левіна*», науковий керівник роботи: проф. каф. ММЗІ НН ФТІ, док. тех. наук Олексійчук Антон Миколайович,

затверджені наказом по університету №__ від «__» _____ 2023 р.

2. Термін подання студентом роботи: «__» _____ 2023 р.

3. Об'єкт дослідження: *Сучасні версії блокових шифрів та їх кодери.*

4. Предмет дослідження: *Алгоритм Гольдрайха-Левіна.*

5. Перелік завдань:

1) навести інтерпретації алгоритмів та особливості їх використання в залежності від того, чи використовуються вони тільки для задач кодування та декодування, чи для знаходження коефіцієнтів Фур'є;

2) імплементувати алгоритм Гольдрайха-Левіна для знаходження коефіцієнтів Фур'є, що надалі буде використано для кодера TurboAE;

3) провести аналіз отриманих результатів імплементації алгоритму Гольдрайха-Левіна в залежності від кількості запитів, які

необхідно пройти для визначення результатів по кожному з блоків кодера TurboAE.

6. Орієнтовний перелік графічного (ілюстративного) матеріалу: *презентація доповіді*

7. Орієнтовний перелік публікацій: *планується доповідь на всеукраїнській конференції*

8. Дата видачі завдання: 10 вересня 2022 р.

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання	Примітка
1	Узгодження теми роботи із науковим керівником	01-15 вересня 2022 р.	Виконано
2	Огляд опублікованих джерел за тематикою дослідження	Вересень-грудень 2022 р.	Виконано
3	Наведення інтерпретацій алгоритму та особливості їх використання	січень 2023 р.	Виконано
4	Імплементування алгоритму Гольдрайха-Левіна для знаходження коефіцієнтів Фур'є	лютий-березень 2023 р.	Виконано
5	Проведення аналізу отриманих результатів імплементації алгоритму Гольдрайха-Левіна	квітень 2023 р.	Виконано
6	Оформлення та захист дипломної роботи	травень-червень 2023 р.	Виконано

Студент _____ Марія МАРТИНОВА

Керівник _____ Антон ОЛЕКСІЙЧУК

РЕФЕРАТ

Кваліфікаційна робота містить: 43 стор., 8 рисунків, 1 таблицю, 20 джерел.

У дипломній роботі продемонстровано практичну значущість алгоритму Гольдрайха-Левіна для аналізу шифросистем на надійність. Об'єктом дослідження є сучасні версії блокових шифрів та їх кодери. Предметом дослідження даної роботи є алгоритм Гольдрайха-Левіна. Наведено практичну реалізацію алгоритму Гольдрайха-Левіна, який використовується для знаходження найбільшого коефіцієнта Фур'є. Даний алгоритм був імплементований для знаходження вагів Фур'є для кодера TurboAE. У дипломній роботі наведено числові результати значень коефіцієнтів Фур'є в залежності від блоків кодера TurboAE та кількості запитів, які необхідно зробити для знаходження коефіцієнтів.

АЛГОРИТМ ГОЛЬДРАЙХА-ЛЕВІНА, КОЕФІЦІЄНТИ ФУР'Є, TURBOAE, ЗАВАДОСТІЙКІ КОДИ, ДЕКОДУВАННЯ СПИСКОМ, КОДИ АДАМАРА.

ABSTRACT

The qualification work contains: 43 p., 8 drawings, 1 table, 20 sources.

The thesis demonstrates the practical significance of the Goldreich-Levin algorithm for reliability analysis of cryptosystems. The object of research is modern versions of block ciphers and their encoders. The subject of research in this work is the Goldreich-Levin algorithm. A practical implementation of the Goldreich-Levin algorithm, which is used to find the largest Fourier coefficient, is given. This algorithm was implemented to find the Fourier weights for the TurboAE encoder. The thesis gives the numerical results of the values of the Fourier coefficients depending on the blocks of the TurboAE encoder and the number of requests that must be made to find the coefficients.

GOLDREICH-LEVIN ALGORITHM, FOURIER COEFFICIENTS, TURBOAE, ERROR-CORRECTING CODES, LIST DECODING, HADAMARD CODES.

ЗМІСТ

Перелік умовних позначень, скорочень і термінів 7
Вступ 8
1 Теоретичні відомості та огляд літератури 10
1.1 Завадостійкі коди 10
1.2 Декодування списком 13
1.3 Коди Адамара 16
1.4 Оригінальний алгоритм Гольдрайха-Левіна для декодування кодів Адамара 18
1.5 Алгоритм Гольдрайха-Левіна зниженої складності для декодування кодів Адамара 22
1.6 Інтерпретація алгоритму GL для аналізу Фур'є 26
Висновки до розділу 1 29
2 Експериментальне дослідження використання алгоритму Гольдрайха-Левіна для знаходження найбільших коефіцієнтів Фур'є . 30
2.1 Архітектура кодера TurboAE 30
2.2 Використання алгоритму Гольдрайха-Левіна для знаходження коефіцієнтів Фур'є 34
2.3 Практична реалізація знаходження вагів Фур'є для кодера TurboAE 36
Висновки до розділу 2 38
Висновки 40
Перелік посилань 41
Додаток А Тексти програм 43
А.1 Програма 1 43

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

GL — Гольдрайх-Левін

$\oplus$ — операція побітового додавання

ВСТУП

Актуальність дослідження. Алгоритм Гольдрайха-Левіна було винайдено для задач криптографії, проте існує велика кількість публікацій, в яких даний алгоритм використовується також для задач навчання. На сьогодні він відіграє велику роль як в задачах лінійної, так і в задачах поліноміальної апроксимації. Класичний (лінійний) алгоритм Гольдрайха-Левіна має широке застосування в теорії навчання, теорії кодування та побудові генераторів псевдовипадкових чисел у криптографії, а також тісно пов'язаний з аналізом Фур'є. Через широкий вибір методів його застосування на практиці, ефективність варіантів використання алгоритму Гольдрайха-Левіна є актуальною задачею на сьогодні.

Метою дослідження є демонстрація практичної значущості алгоритму Гольдрайха-Левіна для аналізу шифросистем на надійність. **Задачею дослідження** є імплементація алгоритму для знаходження вагів Фур'є для кодера TurboAE та аналіз отриманих результатів. Для розв'язання задачі необхідно вирішити такі завдання:

1) провести огляд літератури щодо алгоритмів Гольдрайха-Левіна, які на сьогодні використовуються на практиці;

2) навести інтерпретації алгоритмів та особливості їх використання в залежності від того, чи використовуються вони тільки для задач кодування та декодування, чи для знаходження коефіцієнтів Фур'є;

3) імплементувати алгоритм Гольдрайха-Левіна для знаходження коефіцієнтів Фур'є, що надалі буде використано для кодера TurboAE;

4) провести аналіз отриманих результатів імплементації алгоритму Гольдрайха-Левіна в залежності від кількості запитів, які необхідно пройти для визначення результатів по кожному з блоків кодера TurboAE.

Об'єктом дослідження є сучасні версії блокових шифрів та їх кодери.

Предметом дослідження є алгоритм Гольдрайха-Левіна.

При розв'язанні поставлених завдань використовувались такі *методи дослідження*: лінійного криптоаналізу, аналізу Фур'є, теорії кодування, глибокого навчання.

Наукова новизна отриманих результатів полягає у наступному:

- отримало подальший розвиток використання алгоритму Гольдрайха-Левіна для знаходження коефіцієнтів Фур'є;

- вперше запропоновано використовувати алгоритм Гольдрайха-Левіна для кодера $TurboAE$.

Практичне значення отриманих результатів:

- проведено класифікацію всіх можливих методів використання алгоритму Гольдрайха-Левіна на практиці;

- впроваджено алгоритм Гольдрайха-Левіна, який спрямований на пошук множин S з величиною коефіцієнта Фур'є для кодера TurboAE;

- результати проведених експериментальних досліджень показують доцільність використання алгоритму Гольдрайха-Левіна в залежності від кількості запитів та заданого порогового значення.

1 ТЕОРЕТИЧНІ ВІДОМОСТІ ТА ОГЛЯД ЛІТЕРАТУРИ

У даному розділі було введено деякі поняття з області вивчення завадостійких кодів, зокрема декодування списком та коди Адамара, які використовуються в одній з інтерпретацій алгоритму Гольдрайха-Левіна. Також були наведені оригінальний алгоритм Гольдрайха-Левіна та його версія зниженої складності. Була представлена також інша інтерпретація алгоритму, застосовна для аналізу Фур'є.

1.1 Завадостійкі коди

У повсякденні, коли одна сторона хоче передати повідомлення іншій, віддаленій стороні, найчастіше проміжний канал зв'язку є шумним і спотворює повідомлення під час передачі. Проблема надійної передачі інформації за таким зашумленим каналом є фундаментальною і складною. Завадостійкі коди розв'язують цю проблему.

Неформально завадостійкі коди забезпечують систематичний спосіб додавання надлишковості до повідомлення перед його передачею так, що навіть при отриманні дещо спотвореного повідомлення, надлишковість в повідомленні дозволяє одержувачу зрозуміти вихідне повідомлення, яке відправник хотів передати.

Для початку наведемо деякі фундаментальні означення (подальший матеріал викладено згідно [**2**]):

Означення 1.1. *Лінійний блоковий код C довжини n – це лінійний підпростір векторного простору V_n.*

Означення 1.2. *Розмірністю k_c коду C називається його розмірність як векторного простору*

$$k_c = dimC.$$

Означення 1.3. *Мінімальною відстанню d_c коду (кодовою відстанню) C називається мінімальна відстань Геммінга між його кодовими словами*

$$d_c = \min\{dist(c, c')|c, c' \in C, c \neq c'\}.$$

Зауваження. *Оскільки код C є векторним простором, то його мінімальна відстань d_c задовольняє такій рівності*

$$d_c = \min\{wt(c)|c \in C, c \neq 0\}.$$

Означення 1.4. *Код довжини n, розмірності k та з мінімальною відстанню d називається $[n,k,d]$-кодом. У разі коли мінімальна відстань d не є центральною в міркуваннях або невідома, використовується позначення $[n,k]$-код.*

Означення 1.5. *Швидкість R_c $[n, k]$-коду C визначається наступною рівністю*

$$R_c = \frac{k}{n}$$

Нехай $C \subset V_n$ – лінійний код. Кажуть, що вектор $x \in V_n$ є r-покритим кодом C, якщо існує кодове слово $c \in C$ таке, що $dist(x, c) \leqslant r$.

Означення 1.6. *Радіусом покриття r_c коду C називається мінімальне натуральне число r таке, що будь-який вектор з V_n є r-покритим кодом C.*

Твердження 1.1. Нехай $C \subset V_n$ – лінійний код і d_c – його мінімальна відстань. Тоді

$$r_c \geqslant \lfloor \frac{d_c - 1}{2} \rfloor.$$

Означення 1.7. *Матриця G називається породжувальною матрицею $[n, k]$-коду C. Множиною кодових слів коду C називається простір, натягнутий на рядки матриці G.*

Породжувальна матриця є стислим описом коду. Для опису коду C

потрібно $k \cdot n$ бітів, що визначають елементи матриці G. У той самий час для запису таблиці всіх кодових слів коду C необхідно $2^k \cdot n$ бітів.

Нехай C – $[n, k]$-код. Припустимо, що передане кодове слово $y \in C$. У каналі зв'язку передане кодове слово спотвориться. На виході одержувач інформації спостерігатиме вектор $z = y \oplus e$, де $e = (e^{(1)}, e^{(2)}, ..., e^{(n)}) \in V_n$ — вектор помилок.

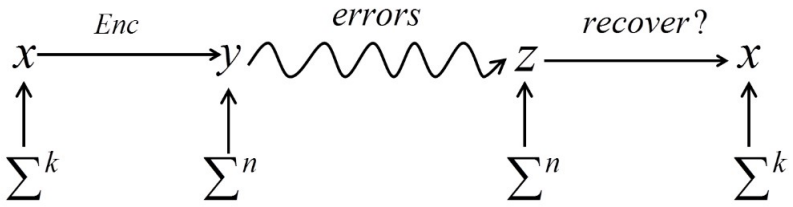

Рисунок 1.1 – Процедура кодування й декодування

На зображенні вище показано процедуру кодування та декодування. Тут помилки означають зміну до t символів в y, а "зміна" тут означає перетворення на інший символ. Декодування відновлює оригінальний x із z.

Наведемо теорему згідно [**2**]:

Теорема 1.1. Код C з мінімальною відстанню d_c може виправляти будь-які t помилок, якщо

$$d_c \geqslant 2t + 1$$

Визначимо деякі основні поняття щодо завадостійких кодів [**1**]:

– **Кодування**. Функція кодування з параметрами k, n є функцією $E : \Sigma^k \longrightarrow \Sigma^n$, яка перетворює повідомлення m, що складається з k символів над деяким алфавітом Σ у довший, надлишковий рядок $E(m)$ довжини n над Σ. Закодований рядок $E(m)$ називають кодовим словом.

– **Завадостійкий код**. Сам завадостійкий код визначається як образ функції кодування. Іншими словами, це набір усіх кодових слів, які використовуються для кодування різних повідомлень.

– **Швидкість**. Відношення кількості символів інформації до довжини кодування — величина k/n у наведеному вище означенні — називається швидкістю коду. Це важливий параметр коду, оскільки він є мірою кількості надлишковості, доданої кодуванням.

– **Декодування**. Перед передачею повідомлення відправник спочатку кодує його за допомогою завадостійких кодів, а потім передає отримане кодове слово по каналу. Одержувач отримує, можливо, спотворену копію переданого кодового слова, і йому потрібно відновити оригінальне повідомлення. Це робиться за допомогою функції декодування $D : \Sigma^n \longrightarrow \Sigma^k$, яка зіставляє рядки довжини n (тобто отримані слова з шумом) рядкам довжини k (тобто ті, що декодер вважає переданим повідомленням).

– **Відстань**. Мінімальна відстань коду визначає, на скільки "віддалені" одне від одного різні кодові слова. Визначається відстань між словами як число компонент, за якими вони відрізняються. Тоді відстань коду визначається як найменша відстань між двома різними кодовими словами.

Більшість алгоритмів для декодування завадостійких кодів вимагають, щоб отримане слово знаходилося в межах менше ніж половини мінімальної відстані кодового слова, щоб кодове слово можна було однозначно відновити. У 1950-х роках було представлене поняття спискового декодування, що дозволяє декодувати поза цим "бар'єром" у половину мінімальної відстані. Замість того, щоб виводити одне кодове слово, алгоритм декодування списком виводить усі кодові слова в заданому радіусі отриманого слова.

1.2 Декодування списком

У теорії кодування, декодування списком є альтернативою унікальному декодуванню завадостійких кодів для великої кількості помилок. Основна ідея декодування списком полягає у тому, що

прийнятій послідовності символів ставиться у відповідність деякий список цілих чисел. Якщо ціле число, що відповідає переданому повідомленню, не входить у цей список, отриманий у результаті декодування, то вважається, що виникла помилка при декодуванні списком.

Наведемо формальне означення кодів, декодовних списком [**12**]:

Означення 1.8. *Нехай* $\rho \in [0,1]$ *та* $L \geqslant 1$. *Тоді* $C \subseteq \Sigma^n$ *є* (ρ,L)-*декодовним списком, якщо* $\forall y \in \Sigma^n$,

$$|\{c \in C : \delta(c,y) \leqslant \rho\}| \leqslant L,$$

де L *– розмір списку,* ρ *– радіус декодування списком, а* $\delta(x,y)$ *– це відносна відстань Геммінга.*

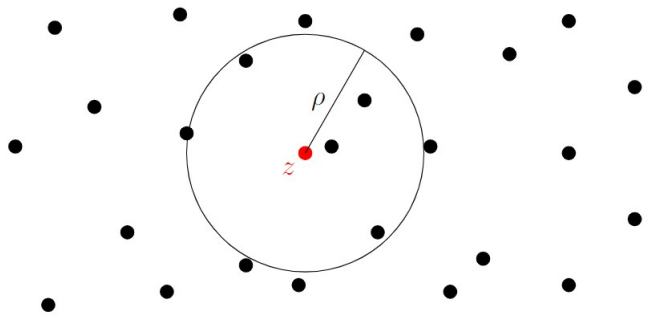

Рисунок 1.2 – Ілюстрація (ρ,L)-декодовності списком. Чорні точки представляють кодові слова; червона точка – будь-який центр.

Гарантовано, що будь-яка куля Геммінга [**12**] містить не більше L кодових слів.

Наведемо формальне означення відстані Геммінга [**4**]:

Означення 1.9. *Відстань Геммінга — це кількість позицій, у яких відповідні символи відрізняються*

$$\Delta(y,z) = |\{i : y_i \neq z_i\}|$$

Означення 1.10. *Відносною відстанню Геммінга між* $y, z \in \Sigma^n$ *є*

$$\delta(y,z) = \frac{\Delta(y,z)}{n} = \frac{1}{n}\sum_{i=1}^{n} \mathbb{1}\{y_i \neq z_i\}$$

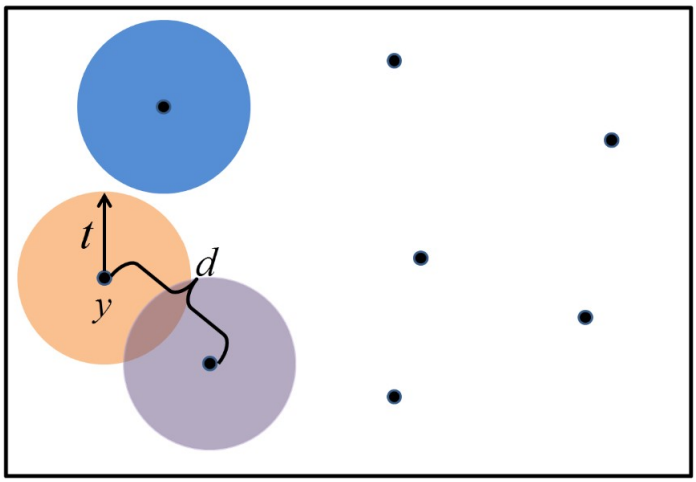

Рисунок 1.3 – Відстань Геммінга

На зображенні вище прямокутник позначає всі елементи в Σ^n. Кожна вершина є кодовим словом, а кожна куля навколо є можливим z, яке отримує одержувач. t означає максимальну кількість помилок. Тут для кодового слова y, усе що знаходиться у межах кулі Геммінга має відстань Геммінга не більше ніж t до y. Якщо не існує двох куль Геммінга, які перетинаються, ми можемо відновити кожне повідомлення. На зображенні помаранчева куля перекриває фіолетову. Тоді інколи ми не можемо правильно відновити x, пов'язаний з y. Тут d вказує на мінімальну відстань між двома вершинами [4].

Означення 1.11. *Мінімальна відстань — це найменша відстань між двома різними кодовими словами:*

$$d = \min_{y \neq y' \in C} \{\Delta(y, y')\}$$

Зі списку обирається кодове слово, яке має найвищу ймовірність того, що воно є саме переданим словом. Пошук такого кодового слова

називається декодуванням максимальної правдоподібності. Декодування за принципом максимальної правдоподібності означає вибір з усіх можливих кодових слів того слова y, що знаходиться на мінімальній відстані Геммінга від прийнятого слова y', і лише потім – відновлення вихідного слова x з його коду y.

Класичні унікальні алгоритми декодування декодують лише на половину відстані. Зокрема, вони ніколи не можуть декодувати, якщо більш ніж половина символів є помилковими. Декодування списком, з іншого боку, має на меті обробку помилок, що перевищують половину відстані, і, отже, повинно впоратися із ситуаціями, коли більш ніж половина символів є помилковими, включаючи випадки, коли шум надто великий і значно переважає кількість правильної інформації.

1.3 Коди Адамара

Добре вивченим сімейством лінійних кодів з надлишковістю є класичні двійкові коди Ріда-Мюллера.

Код Ріда-Мюллера, що позначається як $RM(r, m)$, визначається параметрами r і m, де $0 \leqslant r \leqslant m$, r – порядок коду, 2^m – довжина коду.

Особливий інтерес становлять коди Ріда-Мюллера першого порядку [**5**], тобто коди, засновані на мультилінійних поліномах, також відомих як симплексні коди. Їх різновид, заснований на однорідних поліномах без постійного члена, називають кодами Адамара.

Код Адамара — це код із надзвичайно низькою швидкістю, але великою відстанню. Він завжди використовується для виявлення та виправлення помилок під час передачі повідомлень через дуже шумні або ненадійні канали.

Наведемо означення породжувальної матриці Адамара згідно [**3**]:

Означення 1.12. *Нехай $r \in \mathbb{N}^+$. Породжувальна матриця коду Адамара є $2^r \times r$ матрицею, де рядки є всіма можливими двійковими*

рядками у $\mathbb{F}_2^r$.

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

Подальший матеріал викладено згідно [**2**]:

Означення 1.13. *Кодування Адамара визначається як послідовність усіх скалярних добутків із x:*

$$x \mapsto \langle a, x \rangle_{a \in \mathbb{F}_2^r}$$

Означення 1.14. *Дано* $x \in \mathbb{F}_2^r$*, лінійний поліном* $L_x : \mathbb{F}_2^r \to \mathbb{F}_2$ *із r змінними визначаємо як*

$$a \mapsto x^T a = \sum_{i=1}^{r} x_i a_i$$

Фактично, для кожного x, це лінійне відображення. Коли x є фіксованим, ми можемо розглядати x_i як коефіцієнти, а a_i як змінні.

Це схоже на відображення x у «таблицю істинності» L_x, оскільки $\langle a, x \rangle_{a \in \mathbb{F}_2^r} = (L_x(a))_{a \in \mathbb{F}_2^r}$

Твердження 1.2. Код Адамара є $[2^r, r, 2^r]$ кодом.

Зауваження. *Іншими словами, код Адамара є* $[n, log_2 n, \frac{n}{2}]_2$ *кодом із* $n = 2^r$.

Код Адамара має дуже низьку швидкість, оскільки він зіставляє l символам над $\mathbb{F}_q$ q^l символів. Але він має дуже хороші властивості відстані — його відносна відстань дорівнює $(1 - 1/q)$, і фактично кожне ненульове кодове слово має вагу Геммінга, що дорівнює $(q^l - q^{l-1})$.

1.4 Оригінальний алгоритм Гольдрайха-Левіна для декодування кодів Адамара

Оскільки код Адамара має відстань $\frac{1}{2}$, його можна однозначно декодувати з часткою помилок менш ніж $\frac{1}{4}$ за допомогою поліноміального алгоритму декодування. Теорема Гольдрайха-Левіна каже, що можна декодувати код Геммінга [**2**] локально, тобто лише кількома запитами до бітів отриманого рядка, не розглядаючи його повністю. Крім того, можна декодувати з часткою помилок $\frac{1}{2} - \epsilon$. Звичайно, це означає, що розшифроване повідомлення більше не є унікальним. Однак існує щонайбільше $poly(\frac{1}{\epsilon})$ таких повідомлень, і цей алгоритм виводить їх усі.

Тепер ми сформулюємо теорему Гольдрайха-Левіна [**6**]:

Теорема 1.2. Для кожного $\delta > 0$ існує рандомізований алгоритм $\mathcal{A}$ такий, що для будь-якої функції $B : \{0,1\}^m \to \{0,1\}^n$, якщо існує $x \in \{0,1\}^m$, що задовольняє

$$\Pr_{y \sim \{0,1\}^m}[B(y) = \langle x, y \rangle] \geqslant \frac{1}{2} + \epsilon$$

алгоритм $\mathcal{A}$ виконує $poly(\frac{m}{\epsilon})$ запитів до B і виводить список $L \subseteq \{0,1\}^n$ із наступними властивостями:

1) $|L| \leqslant poly(1/\epsilon)$

2) $x \in L$

з імовірністю не меншою за $1 - \delta$.

Мовою теорії кодування наведена вище теорема каже наступне. Ми можемо локально декодувати списком код Адамара з часткою помилок

$\frac{1}{2} - \epsilon$.

Буде зручно представляти кодові слова коду Адамара як функції [**7**]:

Означення 1.15. *Нехай* $a \in \{0,1\}^k$ *і визначимо* $L_x : \{0,1\}^k \to \{0,1\}$ *як функцію* $L_x(a) = \langle x, a \rangle$. *Тоді* $L_x(\cdot)$ *є кодуванням Адамара* x.

Ми починаємо з розгляду простішого випадку, коли нам надано доступ, як до оракула, до функції $B(\cdot)$ такої, що $\Pr_r[B(r) = \langle x,r \rangle] \geqslant \frac{7}{8}$, і ми хочемо знайти x.

Якщо ми позначимо як e_i вектор, що має 1 в i-й координаті та 0 в інших координатах, ми бачимо, що $x_i = \langle x,e_i \rangle$. Крім того, для кожного r ми маємо $x_i = \langle x, (r \oplus e_i) \rangle \oplus \langle x, r \rangle$ внаслідок лінійності оператора $\langle \cdot, \cdot \rangle$ та того факту, що $r \oplus r$ є повністю нульовим вектором.

Розглянемо тепер процес вибору випадкового r та обчислення $B(r \oplus e_i) \oplus B(r)$. За винятком випадків, коли ймовірність не перевищує $1/8$, $B(r \oplus e_i) = \langle x, (r \oplus e_i) \rangle$ і, за винятком випадків, коли ймовірність не перевищує $1/8$, $B(r) = \langle x,r \rangle$. Тому, з імовірністю не менше $3/4$,

$$B(r \oplus e_i) \oplus B(r) = \langle x, (r \oplus e_i) \rangle \oplus \langle x, r \rangle$$

Це передбачає наступний алгоритм [**8**]:

Алгоритм $A_{\frac{7}{8}}$:

для $i := 1$ до n виконувати:

обрати $k = O(\log n)$ рандомних елементів $r^1,...,r^k \in \{0,1\}^n$

обчислити:

$B(r^1 \oplus e_i) \oplus B(r^1)$

$B(r^2 \oplus e_i) \oplus B(r^2)$

...

$B(r^k \oplus e_i) \oplus B(r^k)$

присвоїти x_i значення, яке зустрічається в більшості цих обчислень

повернути x

Щоб проаналізувати алгоритм, треба зауважити, що для певного

значення i, ми очікуємо отримати правильне значення x_i у $3/4$ від k випробувань, і алгоритм виводить правильне значення x_i за умови, що більш ніж половина випробувань правильні. Тоді, згідно з границею Чернова [**13**], ймовірність неправильного оцінювання x_i дорівнює $e^{-\Omega(k)}$. Тоді ми можемо обрати $k = O(\log n)$ і переконатися, що ймовірність помилки не перевищує, скажімо, $1/100n$, і, отже, ми робимо висновок, що алгоритм повертає правильний результат з імовірністю щонайменше $99/100$.

Зауважимо, що час виконання цієї програми становить $O(n^2k) = O(n^2 \log n)$ і що вона здійснює $O(nk) = O(n \log n)$ викликів оракула.

Розглянемо тепер загальний випадок. Нам дано оракул $B(\cdot)$ такий, що $B(r) = \langle x,r \rangle$ для частки $1/2 + \epsilon$ від кількості значень r. Нашою метою буде використання $B(\cdot)$ для симуляції оракула, який має узгодження у $7/8$ випадків з $\langle x,r \rangle$, щоб ми могли використовувати алгоритм $A_{\frac{7}{8}}$ для знаходження x.

Спочатку ми обираємо k випадкових точок $r_1,...,r_k \in \{0,1\}^n$, де $k = O(1/\epsilon^2)$. Припустимо, що ми деяким чином отримали значення $L_x(r_1),...,L_x(r_k)$. Потім визначаємо $B'(r)$ мажоритарним голосуванням від:

$$L_x(r_j) \oplus B(r \oplus r_j), j = 1,2,...,k$$

Оскільки для кожного j вираз вище дорівнює $L_x(r)$ з імовірністю принаймні $\frac{1}{2} + \epsilon$, беручи $k = O(1/\epsilon^2)$, ми можемо впевнитися, що

$$\Pr_{r,r_1,...,r_k}[B'(r) = L_x(r)] \geqslant \frac{31}{32}$$

з чого випливає

$$\Pr_{r_1,...,r_k}[\Pr_r[B'(r) = L_x(r)] \geqslant \frac{7}{8}] \geqslant \frac{3}{4}.$$

Розглянемо першу версію оригінального алгоритму [**8**]:

Алгоритм $GL^{(1)}$:

обрати $r_1,...,r_k \in \{0,1\}^n$, де $k = O(1/\epsilon^2)$

для всіх $L_x(r_1),...,L_x(r_k) \in \{0,1\}$

визначити $B'_{L_x(r_1),...,L_x(r_k)}(r)$ як

мажоритарне голосування від: $L_x(r_j) \oplus B(r \oplus r_j)$

застосувати алгоритм $A_{\frac{7}{8}}$ до $B'_{L_x(r_1),...,L_x(r_k)}(r)$

додати результат до списку

Ідея цієї програми полягає в тому, що ми насправді не знаємо значень $\langle x,r_j \rangle$, але ми можемо «вгадати» їх, розглядаючи всі варіанти для бітів $L_x(r_j)$. Якщо $B(r)$ узгоджується з $L_x(r)$ принаймні на частці $1/2 + \epsilon$ від кількості значень r, то існує ймовірність принаймні $3/4$ того, що в одній з ітерацій ми викличемо алгоритм $A_{\frac{7}{8}}$ зі змодельованим оракулом, який узгоджується з $L_x(r)$ з імовірністю $7/8$. Отже, остаточний список містить x з імовірністю принаймні $\frac{3}{4} - \frac{1}{100} = \frac{37}{50} > \frac{1}{2}$.

Очевидна проблема з цим алгоритмом полягає в тому, що його час роботи є експоненційним у $k = O(1/\epsilon^2)$, а результівний список також може бути експоненціально більшим, ніж обмеження $O(1/\epsilon^2)$.

Щоб подолати ці проблеми, розглянемо наступний аналогічний алгоритм, який є другою версією оригінального [**8**].

Алгоритм $GL^{(2)}$:

обрати $r_1,...,r_l \in \{0,1\}^n$, де $l = \log O(1/\epsilon^2)$

визначити $r_S := \bigoplus_{j \in S} r_j$ для кожного

непорожнього $S \subseteq \{1,...,l\}$

для всіх $L_x(r_1),...,L_x(r_l) \in \{0,1\}$

визначити $L_x(r_S) := \bigoplus_{j \in S} L_x(r_j)$ для кожного

непорожнього $S \subseteq \{1,...,l\}$

визначити $B'_{L_x(r_1),...,L_x(r_l)}(r)$ як мажоритарне

голосування від $L_x(r_S) \oplus B(r \oplus r_S)$

над непорожнім $S \subseteq \{1,...,l\}$

запустити Алгоритм $A_{\frac{7}{8}}$ з оракулом $B'_{L_x(r_1),...,L_x(r_l)}$

додати результат до списку

Тепер подивимося, чому цей алгоритм працює. Спочатку визначаємо $r_S := \bigoplus_{j \in S} r_j$ для будь-якого непорожнього $S \subseteq \{1,...,l\}$. Тоді, оскільки $r_1,...,r_l \in \{0,1\}^n$ є випадковими, звідси випливає, що для будь-яких $S \neq T$, r_S і r_T незалежні та рівномірно розподілені. Тепер розглядаємо x таким, що $L_x(r)$ та $B(r)$ узгоджувалися на частці $\frac{1}{2} + \epsilon$ від кількості значень r. Тоді для вибору $L_x(r_j)$, де $L_x(r_j) = \langle x,r_j \rangle$ для всіх j, ми маємо

$$L_x(r_S) = \langle x,r_S \rangle$$

для кожного непорожнього S. У такому випадку, для кожного S і кожного r існує ймовірність щонайменше $\frac{1}{2} + \epsilon$ для вибору r_j, щоб

$$L_x(r_S) \oplus B(r \oplus r_S) = \langle x,r \rangle$$

і ці події попарно незалежні.

Теорема про складність алгоритму $GL^{(2)}$ згідно [**9**]:

Теорема 1.3. Нехай $m \to \infty$, $n = 2^m$, $\epsilon \in (\frac{m}{\sqrt{n}}, 1)$, і нехай додатне число s задовольняє: $1 < s \leqslant n\epsilon^2/80$. Тоді для будь-якого отриманого вектора $L_x(r) \in \{0,1\}^n$, алгоритм $GL^{(2)}$ реконструює весь список $\mathcal{L}$ лінійних функцій, розташованих на відстані $\frac{n}{2}(1 - \epsilon)$ від $L_x(r)$ з імовірністю не менше ніж $1 - 2^{-s}$ і складністю

$$O(m\epsilon^{-4}s + m\epsilon^{-4} \log \frac{m}{\epsilon^2}).$$

1.5 Алгоритм Гольдрайха-Левіна зниженої складності для декодування кодів Адамара

Алгоритм GL [**8**] істотно просунув всю теорію жорстких предикатів [**14**]. Однак його складність зростає як ϵ^{-4}, що робить його менш ефективним з точки зору кодування. Зокрема, його складність має квадратичний порядок n^2s, якщо декодер працює близько до пропускної

здатності каналу. Таким чином, він значно перевищує складність $O(n \ln^2 n)$ машини Гріна [**15**], яка безпомилково виконує ту ж задачу.

Г.Кабатянський, І.Думер та С.Таверньє [**9**] звели складність до порядку $m\epsilon^{-2}$ за параметрами m та ϵ.

Перш ніж розглянути модифікований алгоритм зниженої складності, введемо декілька нових позначень та формул (подальше викладення матеріалу згідно [**8**]).

Розглянемо m-вимірний булевий куб $\mathbb{F}_2^m$, який містить $n = 2^m$ точок $x = (x_1,...,x_m)$. Дано будь-який двійковий рядок $a = (a_1,...,a_m)$, визначаємо лінійну булеву функцію $a(x)$ як скалярний добуток

$$a(x) = \langle a, x \rangle = \sum_{j=1}^{m} a_j x_j.$$

Кожна лінійна функція $a(x)$ представлена в $\mathcal{H}(m)$ вектором $\mathbf{a}$ із символами $a(x)$, отриманими, коли x проходить через $\mathbb{F}_2^m$. Для будь-якої підмножини $X \subseteq \mathbb{F}_2^m$, нехай $\mathbf{a}(X)$ — підвектор $\mathbf{a}$ визначений на позиціях $x \in X$.

Нехай дані параметри m, ϵ, s і отримане слово $\mathbf{g} \in \mathbb{F}_2^n$. Метою є відновити список функцій

$$\mathcal{L}_\epsilon(\mathbf{g}) = \{a(x) : d(\mathbf{g}, \mathbf{a}) \leqslant \frac{n}{2}(1 - \epsilon)\}.$$

Тут кожна функція $a(x)$ буде отримана як набір її коефіцієнтів $(a_1,...,a_m)$.

Накладемо деякі обмеження на параметри m, ϵ, s, щоб отримати експоненційно спадну ймовірність помилки 2^{-s}. Беремо $\epsilon \geqslant \frac{m}{n^{1/2}}$ і використовуємо параметри

$$l = \lceil \log \epsilon^{-2} + 4 \rceil, \qquad k = 2(s + \log \frac{m}{\epsilon^2}).$$

Тепер незалежно і рівномірно підберемо l векторів $X = \{x_{(1)},...,x_{(l)}\}$

з $\mathbb{F}_2^m$ та розглянемо лінійний підпростір

$$\mathbb{X} = \{ \sum_{i=1}^{l} h_i x_{(i)} | h_i = 0, 1 \}.$$

Якщо $Rank(\mathbb{X}) < l$ або $\mathbb{X}$ містить вектор $\mathbf{j}$ з одиницею на рівно одній позиції, ми обираємо нову підмножину X.

Дано рядок $b = (b_1, ..., b_l) \in \mathbb{F}_2^l$, будемо шукати будь-яку функцію

$$a_b(x) \in \mathcal{L}_\epsilon(\mathbf{g}) : \mathbf{a}_b(X) = b.$$

Зауважимо, що будь-яка лінійна функція $a_b(x)$ визначена за основою X, також відома у кожній точці його інтервалу $\mathbb{X}$:

$$x = \sum_{i=1}^{l} h_i x_{(i)} \longrightarrow a_b(x) = \sum_{i=1}^{l} h_i b_i$$

Визначимо коефіцієнти $a_{j,b}$ з $a_b(x)$, які є значеннями функції $a_b(x)$ у точках $\mathbf{j}$:

$$a_{j,b} = a_b(\mathbf{j}), \qquad j = 1,...,m.$$

Для будь-якого b функція $a_b(x)$ може бути оцінена в будь-якій точці $y = y_{(i)}$ як

$$\tilde{a}_b(y_{(i)}) = Maj_{x \in \mathbb{X}} \{ a_b(x) \oplus \mathbf{g}(x \oplus y_{(i)}) \} \qquad (1).$$

Подібним чином, задані b та j, ми обчислюємо ту саму функцію $a_b(x)$ у точці $y = y_{(i)} \oplus \mathbf{j}$,

$$\tilde{a}_b(y_{(i)} \oplus \mathbf{j}) = Maj_{x \in \mathbb{X}} \{ a_b(x) \oplus \mathbf{g}(x \oplus y_{(i)} \oplus \mathbf{j}) \} \qquad (2).$$

Дані b та j, тепер ми можемо оцінити коефіцієнт $a_b(\mathbf{j})$ як функцію у точці $y = y_{(i)}$ наступним чином

$$\tilde{a}_{b,i}(\mathbf{j}) = \tilde{a}_b(y_{(i)}) \oplus \tilde{a}_b(y_{(i)} \oplus \mathbf{j}) \qquad (3).$$

Тепер ми можемо об'єднати k оцінок $\tilde{a}_{b,i}(\mathbf{j})$, отриманих для різних i,

в одне мажоритарне значення

$$\tilde{a}_b(\mathbf{j}) = Maj_{i=1,...,k}\{\tilde{a}_{b,i}(\mathbf{j})\}.$$

У певному сенсі вдосконалення алгоритму GL базується на наступному зауваженні [**8**]:

Лема 1.1. Лінійні функції $a(x)$, визначені на $\mathbb{F}_2^m$ утворюють код Адамара $\mathcal{H}(l)$ на будь-якому l-вимірному лінійному підпросторі $\mathbb{X} \subset \mathbb{F}_2^m$

Далі розглянемо мажоритарне голосування за вектором $\mathbf{g}(\mathbb{X} \oplus y_{(i)})$ в (1). Дано будь-яке $y = y_{(i)}$, ми обираємо на користь деякої сталої $\tilde{a}_b(y)$ у (1), замість $\tilde{a}_b(y) \oplus 1$, якщо відповідна афінна функція $a_b(x) \oplus \tilde{a}_b(y)$ ближча до $\mathbf{g}(\mathbb{X} \oplus y)$, ніж протилежна функція $a_b(x) \oplus \tilde{a}_b(y) \oplus 1$. Таким чином, оцінки $\tilde{a}_b(y)$ можна отримати одночасно для різних $b \in \mathbb{F}_2^l$ шляхом декодування вектора $\mathbf{g}(\mathbb{X} \oplus y)$ у список із 2^l найближчих афінних функцій

$$\{b(h) \oplus \tilde{a}_b(y), b \in \mathbb{F}_2^l\}.$$

Тут для кожного b з'являється рівно одна функція. Іншими словами, $\mathbf{g}(\mathbb{X} \oplus y)$ є 2^l-декодовною списком у біортогональний код $RM(1,l)$ розміру 2^{l+1}. Таким чином, отримуємо 2^l рядків коефіцієнтів $(b_1,...,b_l,\tilde{a}_b(y))$, що, своєю чергою, можна переписати як вектор $\tilde{A}(y)$, який має символ $\tilde{a}_b(y)$ на кожній "позиції" $b = (b_1,...,b_l)$ з $\mathbb{F}_2^l$.

Загалом, для кожного $i = 1,...,k$ та $j = 0,...,m$ виконуємо декодування списком F_L вектора $\mathbf{g}(\mathbb{X} \oplus y_{(i)} \oplus \mathbf{j})$ у 2^l найближчих афінних функцій і формуємо вектор $\tilde{A}(y_{(i)} \oplus \mathbf{j})$ довжини 2^l, утворений вільними членами цих афінних функцій. Ця операція еквівалентна (2). Своєю чергою, для кожного $j \geqslant 1$, два вектори $\tilde{A}(y_{(i)})$ і $\tilde{A}(y_{(i)} \oplus \mathbf{j})$ дають новий вектор

$$\tilde{A}_i(\mathbf{j}) = \tilde{A}(y_{(i)}) \oplus \tilde{A}(y_{(i)} \oplus \mathbf{j}),$$

що включає всі символи $\tilde{a}_{b,i}(\mathbf{j})$, отримані в (3). Нарешті, для кожного $j = 1,...,m$ виконуємо мажоритарне голосування на k векторах $\tilde{A}_i(\mathbf{j})$, які

дають m векторів

$$\tilde{A}(\mathbf{j}) = Maj_{i=1,...,k}\{\tilde{A}_i(\mathbf{j})\}.$$

Таким чином, вектори $\tilde{A}(\mathbf{j}) = Maj_{i=1,...,k}\tilde{A}_i(\mathbf{j})$ утворюють $m \times 2^l$ матрицю та дають коефіцієнти $\tilde{a}_b = (\tilde{a}_{1,b},...,\tilde{a}_{m,b})$ функції $\tilde{a}_b(x)$ у будь-якому стовпці $b \in \mathbb{F}_2^l$.

Алгоритм GL тепер змінено наступним чином [**8**]:

Алгоритм $GL_{mod}(\mathbf{g}, m, \epsilon, s)$ для коду $\mathcal{H}(m)$

Вхід: значення m, ϵ, s, вектор $\mathbf{g} \in \{0,1\}^n$

обрати l-вимірний підпростір $\mathbb{X}$, за $\leqslant s$ спроб.

обрати точки $y_{(i)}$, $i = 1,...,k$, за $\leqslant 9k$ спроб.

для кожного i та $j = 0,...,m$

декодувати вектор $\mathbf{g}(\mathbb{X} \oplus y_{(i)} \oplus \mathbf{j})$ у вектор $\tilde{A}(y_{(i)} \oplus \mathbf{j})$

знайти вектор $\tilde{A}_i(\mathbf{j}) = \tilde{A}(y_{(i)}) \oplus \tilde{A}(y_{(i)} \oplus \mathbf{j})$

для кожного $j \geqslant 1$

знайти $\tilde{A}(\mathbf{j}) = Maj_{i=1,...,k}\{\tilde{A}_i(\mathbf{j})\}$

повернути список $\{\tilde{a}_b(x) = \sum_{j=1}^{m} \tilde{a}_b(\mathbf{j})x_j | b \in \mathbb{F}_2^l\}$

Нам потрібно порядку $mkl^2 2^l$ операцій для декодування всіх mk векторів $\mathbf{g}(\mathbb{X} \oplus y_{(i)} \oplus \mathbf{j})$. Нарешті, ми обчислюємо кожен вектор $\tilde{A}_i(\mathbf{j})$, що вимагає 2^l операцій на вектор. Тому алгоритм GL_{mod} має загальну складність

$$\Phi_{mod} \sim mkl^2 2^l = O\big(\big(s + \log\frac{m}{\epsilon^2}\big)\frac{m}{\epsilon^2}\log^2\epsilon\big)$$

Це завершує наш розгляд застосування алгоритму GL для декодування коду Адамара $\mathcal{H}(m)$.

1.6 Інтерпретація алгоритму GL для аналізу Фур'є

Аналіз Фур'є – це дослідження того, як загальні функції можна розкласти на тригонометричні або експоненціальні функції з певними частотами. Причина, чому аналіз Фур'є такий важливий у фізиці, полягає

в тому, що багато диференціальних рівнянь, які керують фізичними системами, є лінійними, що означає, що сума двох розв'язків також є розв'язком. Отже, оскільки аналіз Фур'є говорить нам, що будь-яку функцію можна записати в термінах синусоїдальних функцій, ми можемо обмежити нашу увагу цими функціями під час розв'язання диференціальних рівнянь. І тоді ми можемо створити будь-яку іншу функцію з цих спеціальних. Це дуже корисна стратегія, оскільки незмінно легше мати справу із синусоїдальними функціями, ніж із загальними.

Ми розглянемо алгоритми навчання, які мають "членський доступ" до невідомої функції [**11**]. Тобто вони можуть запитувати у f будь-яку точку на свій вибір. Оскільки це більш потужно, ніж просто отримання випадкової вибірки, очікується, що ці алгоритми будуть більш ефективними. Аналіз Фур'є є корисним інструментом і тут.

Одним із ключових алгоритмів тут є алгоритм Гольдрайха-Левіна (GL) [**11**]. Він ефективно виявляє всі помітні коефіцієнти Фур'є певної функції, роблячи при цьому дуже мало запитів.

Наведемо теорему згідно [**11**]:

Теорема 1.4. Існує рандомізований алгоритм такий, що якщо наданий доступ для запиту до функції $f : \{-1,1\}^n \to \{-1,1\}$ і порогове значення $\epsilon > 0$, то він відновлює список L коефіцієнтів Фур'є так, що

– Якщо $|\hat{f}(S)| \geqslant \epsilon$, тоді $S \in L$.

– Якщо $S \in L$, тоді $|\hat{f}(S)| \geqslant \epsilon/2$.

Алгоритм досягає успіху з імовірністю приблизно 99%, і вимагає $poly(n, 1/\epsilon)$ запитів і часу.

Опишемо типове застосування алгоритму Гольдрайха-Левіна. По-перше, нам потрібне наступне твердження, яке показує, що будь-який конкретний коефіцієнт Фур'є може бути добре наближено за допомогою кількох вибірок [**11**].

Лема 1.2. Нехай $f : \{-1,1\}^n \to \{-1,1\}$. Припустимо, що $|\hat{f}|_1 \leqslant C$. Тоді існує алгоритм навчання із членськими запитами, яким PAC

(Probably Approximately Correct) навчає f в межах помилки $\epsilon > 0$, використовуючи членські запити $poly(n,C,1/\epsilon)$.

Нам потрібно декілька основних фактів про те, як розклад Фур'є булевих функцій поводиться під час зміни базису та/або обмежень. У цьому розділі зручніше буде розглянути булеві функції $f : \mathbb{F}_2^n \to \{-1, 1\}$. Їх розклад за Фур'є є $f(x) = \sum_S \hat{f}(S)(-1)^{\langle x,S\rangle}$.

Ми починаємо з розгляду обмежень, фіксуючи деякі змінні. Наведемо такі твердження [**11**]:

Твердження 1.3. Нехай $1 \leqslant m \leqslant n-1$. Для $a \in \mathbb{F}_2^{n-m}$ визначимо обмеження, позначене $f_{m;a}(x)$, як обмеження, утворене фіксацією останніх $n-m$ елементів як a, а саме

$$f_{m;a}(x) = f(x_1,...,x_m,a_1,...,a_{n-m})$$

Тоді

$$\hat{f}_{m;a}(S) = \sum_{T\subseteq\{m+1,...,n\}} \hat{f}(S \cup T)(-1)^{\langle a,T\rangle}$$

Твердження 1.4. Нехай $f : \mathbb{F}_2^n \to \{-1, 1\}$. Нехай $A \in \mathbb{F}_2^{n\times n}$ матриця повного рангу і нехай $b \in \mathbb{F}_2^n$. Визначимо $g(x) = f(Ax + b)$. Тоді

$$\hat{f}(S) = (-1)^{\langle b,S\rangle}\hat{g}(A^T S)$$

Таким чином, якщо ми хочемо знайти всі помітні коефіцієнти Фур'є у f, ми можемо знайти їх для $f(Ax+b)$, а потім відновити їх для f. Алгоритм Гольдрайха-Левіна для аналізу Фур'є [**11**] використовує це.

Алгоритм GL_{Furier}

нехай $f : \mathbb{F}_2^n \to \{-1, 1\}$ та нехай $\epsilon > 0$ буде пороговим.

встановити $d = O(\log \frac{1}{\epsilon})$ та $\delta = O(\frac{\epsilon^2}{n})$.

нехай $g(x) = f(Ax + b)$, де $A \in \mathbb{F}_2^{n\times n}$ – рандомна оборотна матриця та $b \in \mathbb{F}_2^n$ вибрані рівномірно.

для $1 \leqslant m \leqslant n$ нехай $g_m : \mathbb{F}_2^n \to \{-1, 1\}$ бути обмеженням g, шляхом фіксації останніх $n-m$ бітів як нулі.

Тобто: $g_m(x_1,...,x_m) = g(x_1,...,x_m,0,...,0)$

для кожного $m = d,...,n$ обчислити список L_m великих коефіцієнтів Фур'є для g_m наступним чином:

якщо $m = d$

встановити $L_d = \mathbb{F}_2^d$.

якщо $m > d$

встановити $L_m = L_{m-1} \times \mathbb{F}_2$.

для кожного $S \in L_m$

оцінити $\hat{g_m}(S)$ з точністю $\epsilon/2$ і похибкою δ.

відкинути всі коефіцієнти Фур'є, для яких оцінка нижче $\epsilon/2$.

нехай $L = \{A^T S : S \in L_n\}$.

повернути L.

По-перше, зауважимо, що оскільки $|L_m| \leqslant O(1/\epsilon^2)$ для всіх m, то за нашим вибором δ алгоритм оцінки Фур'є ніколи не помиляється.

В завершенні, наведемо таку лему [**11**]:

Лема 1.3. З ймовірністю принаймні 99% щодо вибору A,b ми маємо, що $L_m \subset L_{m-1} \times \mathbb{F}_2$ для всіх $m = d,...,n$.

Висновки до розділу 1

Більшість досліджень алгоритму Гольдрайха-Левіна зосереджується саме на його застосуванні для декодування кодів Адамара. У даній роботі зроблено акцент на застосуванні алгоритму для знаходження найбільших коефіцієнтів Фур'є та аналізу ефективності його використання для цієї задачі. Зокрема, алгоритм було використано для кодера $TurboAE$.

2 ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ВИКОРИСТАННЯ АЛГОРИТМУ ГОЛЬДРАЙХА-ЛЕВІНА ДЛЯ ЗНАХОДЖЕННЯ НАЙБІЛЬШИХ КОЕФІЦІЄНТІВ ФУР'Є

Основне завдання в криптографії – це побудова сильніших криптографічних функцій зі слабших. Одним з таких прикладів є теорема Гольдрайха-Левіна, яка використовується для побудови «генератора псевдовипадкових чисел» із «односторонньої перестановки» [**16**]. Ключем до аналізу конструкції Гольдрайха-Левіна є алгоритм навчання. Зокрема, алгоритм навчання Гольдрайха-Левіна вирішує наступне завдання: при заданому доступі до цільової функції $f{:}F_2^n \longrightarrow F_2$ знайти всі лінійні функції, з якими f має кореляцію. Іншими словами, необхідно знайти всі помітно великі коефіцієнти Фур'є функції f. У цьому розділі буде наведено експериментальне дослідження того, як алгоритм Гольдрайха-Левіна може допомогти знайти найбільші коефіцієнти Фур'є, які можна використовувати у нейронному кодері $TurboAE$.

2.1 Архітектура кодера TurboAE

Відправною точкою Турбокода є рекурсивний систематичний згортковий (RSC) код, який має оптимальний алгоритм декодування (алгоритм Бала-Кока-Елінека-Равіва (BCJR) [**17**]). Ключовим недоліком коду RSC є те, що алгоритму не вистачає великої кількості пам'яті (оскільки згортковий код працює із ковзним вікном). Як удосконалення пропонувалося ввести довготривалу пам'ять (англ. long-term memory) шляхом створення двох копій вхідних бітів — перша копія проходить через код RSC, а друга копія проходить через перемежувач (англ. interleaver) перед тим, як пройти той самий код. Такий код може бути

декодований за допомогою ітеративного перемежування м'якого декодування (англ. soft decoding) на основі сигналу, отриманого від першої копії, та подальшого використання версії з оберненим перемежуванням попередньої для декодування другої копії. Далі з'явилося ще одне удосконалення – так званий «турбопринцип». «Турбопринцип» належить до ітеративного декодування з послідовним уточненням апостеріорного розподілу бітів, що передаються, на етапах декодування у вихідному порядку і порядку перемежування. Цей код має відмінну продуктивність, і на основі цього у роботі [**18**] розроблено кодер $TurboAE$, що включає як кодер з перемежуванням, так і ітеративний енкодер.

Перемежування широко використовується в системах зв'язку для придушення імпульсних перешкод. Формально перемежувач $x^{\pi} = \pi(x)$ та деперемежувач $x = \pi^{-1}(x^{\pi})$ (обернений перемежувач) пересувають вперед та назад вхідну послідовність x з масивом псевдовипадкового перемежування, відомим як кодеру, так і декодеру відповідно, як показано на рисунку 2.1.

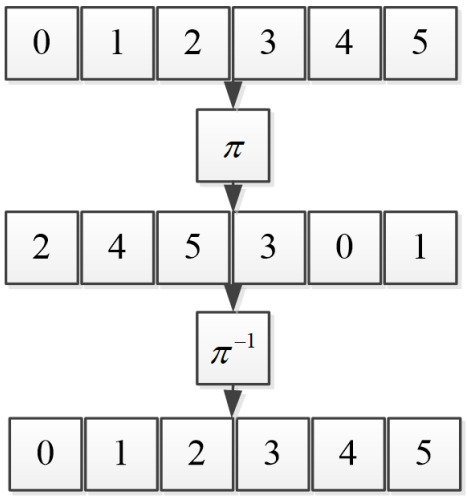

Рисунок 2.1 – Графічне представлення перемежувача та деперемежувача.

У контексті Турбокода та $TurboAE$ перемежування

використовується не для усунення пакетних помилок, а скоріше для додавання довготривалої пам'яті до структури коду.

У дипломній роботі як приклад взято швидкість коду 1/3 для кодувальника з перемежуванням f_θ, який складається з трьох навчальних блоків кодування $f_{i,\theta}(\cdot)$ для $i \in \{1,2,3\}$, де $f_{i,\theta}(\cdot)$ кодує $b_i = f_\theta(u)$, $i \in \{1,2\}$ і $b_3 = f_{3,\theta}(\pi(u))$, b_i — певне неперервне значення. Обмеження потужності канального кодування здійснюється за допомогою блоку обмеження потужності $x_i = h(b_i)$ (рис. 2.2).

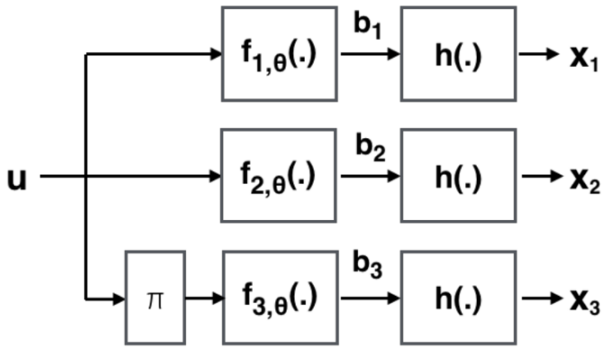

Рисунок 2.2 – Графічне представлення кодера $TurboAE$.

Оскільки прийняті кодові слова кодуються з вихідного повідомлення u та повідомлення з перемежуванням $\pi(u)$, декодування коду з перемежуванням вимагає ітеративного декодування як у порядку перемежування, так і в порядку зворотного перемежування, як показано на рисунку 2.3.

Нехай y_1, y_2, y_3 позначають зашумлені версії x_1, x_2, x_3 відповідно. Декодер виконує декілька ітерацій, при цьому кожна ітерація містить два декодери $g_{\phi,1}$ та $g_{\phi,2}$ для порядку перемежування та зворотного перемежування на i-й ітерації.

Перший декодер $g_{\phi,1}$ бере прийнятий сигнал y_1, y_2 і деперемежує апріорний сигнал p з формою (K, F), де F — розмір інформаційної ознаки для кожного біта коду, щоб створити апостеріорний q тієї ж

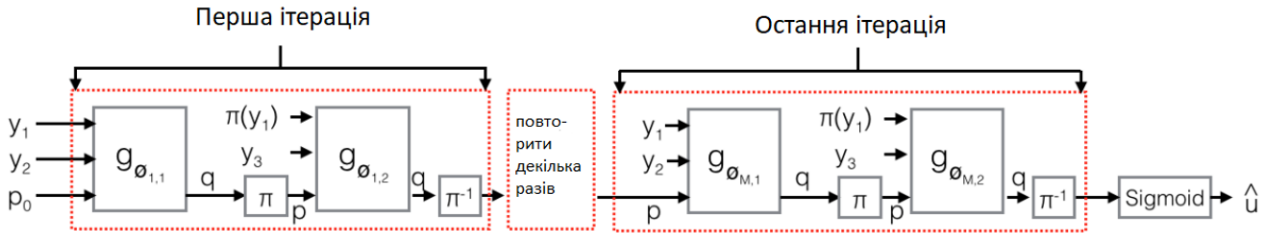

Рисунок 2.3 – Декодер $TurboAE$ на кодовій швидкості 1/3.

форми (K, F). Другий декодер $g_{\phi,2}$ приймає сигнал $\pi(y_1), y_3$ і перемежує попередній апріорний сигнал p для отримання апостеріорного сигналу q. Апостеріорний сигнал попереднього етапу q служить апріорним сигналом наступного етапу p. Перша ітерація приймає 0 як апріорну, а остання апостеріорна ітерація, яка має форму $(K, 1)$, декодується як сигмоїдна функція $\hat{u} = sigmoid(q)$.

І структуру кодера, і структуру декодера можна розглядати як параметризацію турбокода. Як тільки буде параметризовано кодер та декодер, $TurboAE$ можна навчати різними методами машинного навчання, оскільки кодер, канал та декодер диференційовані.

У дипломній роботі буде розглянуто розробку інструментів для інтерпретації процесу навчання для завадостійких кодів за допомогою глибокого навчання, зосередивши увагу на:

1) використанні алгоритму Гольдрайха-Левіна для швидкої інтерпретації навченого кодера;

2) використання коефіцієнтів Фур'є як інструменту для розуміння динаміки навчання та ландшафту втрат.

Входом у мережу є послідовність u зі 100 біт, а виходом кожного блоку $b \in \{1, 2, 3\}$ є послідовність $x_{AE,b} \in \{\pm 1\}^{100}$.

Апроксимаційні вирази для кожного блоку будуть наступними:

Блок 1: $1 \oplus u_1 \oplus u_2 \oplus u_3 \oplus u_4 \oplus u_5$

Блок 2: $u_1 \oplus u_3 \oplus u_4 \oplus u_5$

Блок 3 може приймати значення одного з чотирьох рішень:

Рішення 1: $u_1 \oplus u_2 \oplus u_4$

Рішення 2: $1 \oplus u_1 \oplus u_2 \oplus u_3 \oplus u_4$

Рішення 3: $1 \oplus u_1 \oplus u_2 \oplus u_4 \oplus u_5$

Рішення 4: $1 \oplus u_1 \oplus u_2 \oplus u_3 \oplus u_4 \oplus u_5$

Значення u знаходяться в залежності від вхідних значень x:

$$u_1 = x_{i+2}, u_2 = x_{i+1}, u_3 = x_j, u_4 = x_{i-1}, u_5 = x_{i-2}.$$

Більш детально кодер $TurboAE$ описано в [**19**] та [**20**].

2.2 Використання алгоритму Гольдрайха-Левіна для знаходження коефіцієнтів Фур'є

Переходячи до області $x_i \in \{\pm 1\}$ та вважаючи, що $\chi_s = \Pi_{i \in S} x_i$, кожна булева ($F_2^n \longrightarrow F_2$) та псевдобулева ($F_2^n \longrightarrow R$) функція має єдине представлення Фур'є [**16**]:

$$f(x) = \sum_{S \subseteq \{1,2,\ldots,n\}} \hat{f}(S)\chi_s,$$

де $\hat{f}(S)$ є коефіцієнтами Фур'є для множини S, а $|\hat{f}(S)|^2$ являє собою ваги Фур'є.

Алгоритм Гольдрайха-Левіна (GL) прагне вивести список множин S, для яких $|\hat{f}(S)|$ перевищує заздалегідь заданий поріг γ. Для цього необхідно використовувати наступну теорему.

Теорема 2.1. Для заданого доступу до f:$\{\pm 1\}^n \longrightarrow \{\pm 1\}$, заданих $\gamma, \delta > 0$, існує алгоритм $\left(n, \frac{1}{\gamma} \log \frac{1}{\delta}\right)$, який виводить список $L = \{S_1,\ldots,S_m\}$ при умовах:

(1) якщо $|\hat{f}(S)| \geqslant \gamma$, то $S \in L$;

(2) якщо $S \in L$, то $|\hat{f}(S)| \geqslant \frac{\gamma}{2}$ виконується з ймовірністю $1 - \delta$.

Теорема вимагає, щоб γ було визначено заздалегідь. Якщо γ занадто велике, то нічого не повертається, а якщо занадто мале, то час виконання

збільшується, і повертається більше коефіцієнтів, ніж потрібно.

Алгоритм Гольдрайха-Левіна спрямований на пошук множин S з величиною коефіцієнта Фур'є $|\hat{f}(S)| \geqslant \gamma$. Алгоритм Гольдрайха-Левіна для пошуку коефіцієнтів Фур'є наступний:

Алгоритм GL_{Furier_coef}

Ініціалізувати $k = 0, U = \varnothing$.

Згенерувати випадковий список $L \longleftarrow (\cdot, \cdot, ..., \cdot)$,
який являє собою набір множин $L = \{U \cup T : T \subseteq [n] \backslash [k]\}$.

Для всіх $k \in [1, n]$:

Для всіх $B \in L, B = (a_1, ..., a_{k-1}, \cdot, ..., \cdot)$:

Нехай $B_{ak} = (a_1, ..., a_k, \cdot, ..., \cdot)$ для $a_k = \{0,1\}$.

Оцінити вагу Фур'є $W(B_{ak})$ у межах $\pm\frac{\gamma^2}{4}$ з
ймовірністю не менше $1 - \delta$.

Прибрати значення B зі списку L.

Додати значення B_{ak} до списку L у випадку,
коли розрахована вага $W(B_{ak}) \geqslant \frac{\gamma^2}{2}$.

Вивести значення списку L.

Щоб даний алгоритм працював на практиці, потрібно знати γ й переконатися, що зроблено достатньо запитів. У ході дипломної роботи необхідно знайти набір з найбільшою величиною коефіцієнта Фур'є (або кілька наборів, якщо вони мають приблизно однакові величини коефіцієнта Фур'є). На сьогодні не існує інформації щодо фіксованого значення γ, яке необхідно використовувати для знаходження коефіцієнтів Фур'є. Задача полягає в тому, щоб експериментально вибрати γ й кількість запитів без попереднього знання.

2.3 Практична реалізація знаходження вагів Фур'є для кодера TurboAE

Правильною роботою алгоритму Гольдрайха-Левіна для знаходження коефіцієнтів Фур'є є стабільні вихідні дані у блоках кодера $TurboAE$. Як зазначено на рис. 2.2 та в апроксимаційних виразах для кодера, необхідно щоб один з блоків $TurboAE$ мав стабільні вихідні дані. Стабільність у такому випадку означає, що один і той самий набір вихідних даних створюється послідовно, коли наведений алгоритм Гольдрайха-Левіна запускається декілька разів з різними ініціалізаціями.

У роботі був використаний наступний евристичний підхід: було обрано достатньо велику кількість запитів для проведення експерименту (800 запитів); в залежності від значень γ використовувався алгоритм Гольдрайха-Левіна, описаний у попередньому підрозділі для того, щоб знайти найбільший коефіцієнт Фур'є.

На рис. 2.4 наведено результати роботи алгоритму Гольдрайха-Левіна в залежності від значень γ для різних блоків кодера $TurboAE$.

На рис. 2.4 синім кольором показано результати для блоку 1 кодера $TurboAE$, помаранчевим кольором для блоку 2, зеленим – для блоку 3, червоним – помилкові значення, коли $W(B_{ak}) < \frac{\gamma^2}{2}$.

Як видно з рис. 2.4, при порогових значеннях $\gamma > 0.5$, тільки блоки 1 та 2 мають стабільні вихідні значення. Далі було знижено порогове значення γ. Для цього за експериментальні були взяті значення $\gamma \in \{0.25, 0.375, 0.435, 0.45\}$. У випадку, коли вага Фур'є $W(B_{ak}) < \frac{\gamma^2}{2}$, такі результати розглядалися як помилка і виділені на рис. 2.4 червоним кольором. Тільки у випадку, коли $\gamma > 0.435$ можна казати про стабільні вихідні значення блоку 3.

Щоб дослідити вплив кількості запитів на збіжність ваги Фур'є, також було проведено тестування алгоритму Гольдрайха-Левіна із різною кількістю запитів (10, 25, 50, 100, 200, 400, 800). Результати такого

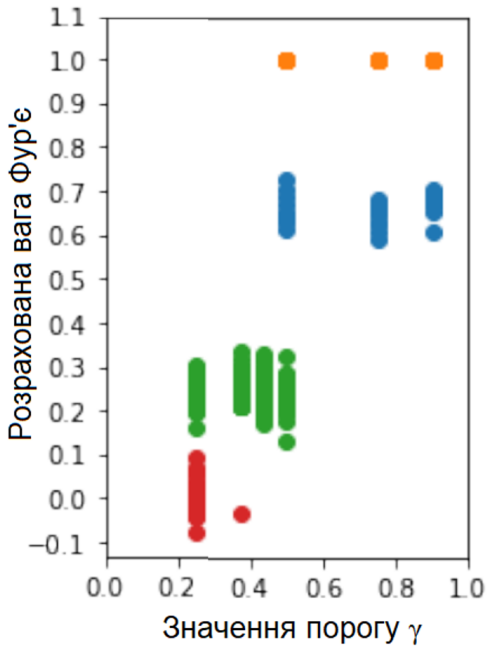

Рисунок 2.4 – Графічне представлення значень вагів Фур'є в залежності від порогових значень.

експерименту наведено на рис. 2.5.

На рис. 2.5 видно, що блок 3 (зелений колір) не має вихідних даних при невеликій кількості запитів, блок 2 (помаранчевий колір) сходиться швидко, а блок 1 (синій колір) сходиться приблизно після 200 запитів. Коли кількість запитів замала, для блоку 2 відображається список виведення помилок (позначається червоними крапками). Швидка збіжність блоку 2, ймовірно, пов'язана з тим, що справжня функція є парною і, отже, має великий коефіцієнт Фур'є. У таблиці 2.1 узагальнено отримані в ході експерименту результати.

Як принципово вибрати γ та мінімальну кількість запитів – цікаве відкрите питання. У табл. 2.1 представлені експериментальні результати. Кількість запитів стосується кількості оцінок функції для оцінки одного очікування.

Для роботи використовувалась готова реалізація кодеру $TurboAE$, що можна розглянути за посиланням: https://github.com/yihanjiang/turboae. Код до алгоритму, наведеного у

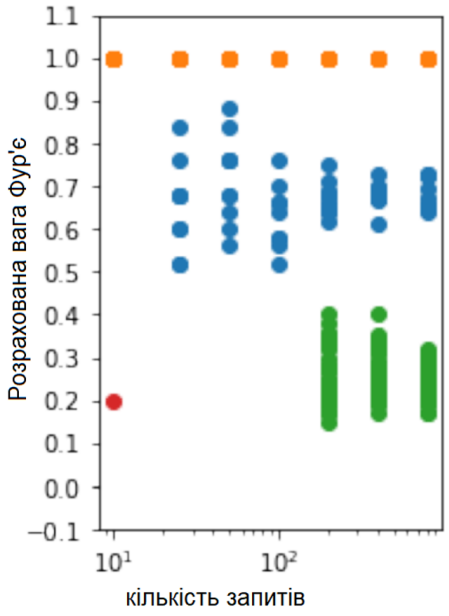

Рисунок 2.5 – Знаходження ваги Фур'є в залежності від кількості запитів для алгоритму Гольдрайха-Левіна.

попередньому розділі, було написано мовою програмування MATLAB, а імплементацію наведено у додатку А.

Висновки до розділу 2

У цьому розділі було наведено експериментальне дослідження того, як алгоритм Гольдрайха-Левіна може допомогти знайти найбільші коефіцієнти Фур'є, які можна використовувати у нейронному кодері *TurboAE*. Було наведено псевдокод до алгоритму Гольдрайха-Левіна для пошуку коефіцієнтів Фур'є, а також результати роботи алгоритму в залежності від значень γ для різних блоків кодера *TurboAE*. Було досліджено вплив кількості запитів на збіжність ваги Фур'є і проведено тестування алгоритму Гольдрайха-Левіна із різною кількістю запитів.

Номер блоку	Набори апроксимацій	Мінімальна кількість запитів	Значення коефіцієнтів Фур'є
1	$1 \oplus u_1 \oplus u_2 \oplus u_3 \oplus u_4 \oplus u_5$	200	0.81
2	$u_1 \oplus u_3 \oplus u_4 \oplus u_5$	25	1.0
3	Рішення 1: $u_1 \oplus u_2 \oplus u_4$ Рішення 2: $1 \oplus u_1 \oplus u_2 \oplus u_3 \oplus u_4 \oplus u_5$ Рішення 3: $1 \oplus u_1 \oplus u_2 \oplus u_4 \oplus u_5$ Рішення 4: $1 \oplus u_1 \oplus u_2 \oplus u_4 \oplus u_5$	800	0.4982 0.5004 0.5014 0.4995
Значення u	$u_1 = x_{i+2}, u_2 = x_{i+1}, u_3 = x_j, u_4 = x_{i-1}, u_5 = x_{i-2}$		

Таблиця 2.1 – Апроксимація Гольдрайха-Левіна для кодера *TurboAE*.

ВИСНОВКИ

У дипломній роботі продемонстровано практичну значущість алгоритму Гольдрайха-Левіна для аналізу шифросистем на надійність. Для практичної реалізації обрано кодер TurboAE, для якого наведено результати використання зазначеного алгоритму.

У ході дипломної роботи були виконані наступні завдання:

1) Проведено огляд літератури щодо алгоритмів Гольдрайха-Левіна, які нині використовуються на практиці;

2) Показано інтерпретації алгоритмів та особливості їх використання в залежності від того, чи використовуються вони тільки для задач кодування та декодування чи для знаходження коефіцієнтів Фур'є.

3) Наведено імплементацію алгоритму Гольдрайха-Левіна для знаходження коефіцієнтів Фур'є, що надалі буде використано для кодера $TurboAE$.

4) Проведено аналіз отриманих результатів імплементації алгоритму Гольдрайха-Левіна в залежності від кількості запитів, які необхідно пройти для визначення результатів по кожному з блоків кодера $TurboAE$.

ПЕРЕЛІК ПОСИЛАНЬ

1. Venkatesan Guruswami. List Decoding of Error-Correcting Codes, 2001 [Електронний ресурс]. — Режим доступу: https://www.cs.cmu.edu/~venkatg/pubs/papers/frozen.pdf.

2. Логачёв О.А., Сальников А.А., Ященко В.В. Булевы функции в теории кодирования и криптологии, 2004.

3. Yuan Zhou. Lecture 14: Hamming and Hadamard Codes. CSCI-B609: A Theorist's Toolkit, Fall 2016 [Електронний ресурс]. — Режим доступу: http://yuanz.web.illinois.edu/teaching/B609fa16/L14.pdf.

4. Ryan O'Donnell. Lecture 10: Error-correcting Codes. A Theorist's Toolkit: CMU 18-859T, Fall 2013 [Електронний ресурс]. — Режим доступу: http://www.cs.cmu.edu/~odonnell/toolkit13/lecture10.pdf.

5. И. В. Агафонова. КОДЫ РИДА–МАЛЛЕРА: ПРИМЕРЫ ИСПРАВЛЕНИЯ ОШИБОК, 2012 [Електронний ресурс]. — Режим доступу: http://dha.spb.ru/PDF/ReedMullerExamples.pdf.

6. Swastik Kopparty. Lecture 1: Decoding Polynomial Codes, 2017 [Електронний ресурс]. — Режим доступу: https://sites.math.rutgers.edu/~sk1233/courses/topics-S17/lec4.pdf.

7. Luca Trevisan. Some Applications of Coding Theory in Computational Complexity, 2004.

8. Luca Trevisan. Notes for Lecture 14 v0.9. CS278: Computational Complexity 2002 [Електронний ресурс]. — Режим доступу: https://lucatrevisan.github.io/cs278-04/notes/lecture14.pdf.

9. I. Dumer, G. Kabatiansky, C. Tavernier. The Goldreich-Levin algorithm with reduced complexity, 2012.

10. Chapter 3: Spectral structure and learning theory. CSE 291: Fourier analysis [Електронний ресурс]. — Режим доступу: https://cseweb.ucsd.edu/~slovett/teaching/WI17-CSE291/3-spectral_structure_and_learning.pdf.

11. David Morin. Chapter 3: Fourier analysis. CSE 291: Fourier analysis,

2009 [Електронний ресурс]. — Режим доступу: https://scholar.harvard.edu/files/david-morin/files/waves_fourier.pdf.

12. Nicolas Resch. List-Decodable Codes (Randomized) Constructions and Applications, 2020 [Електронний ресурс]. — Режим доступу: http://reports-archive.adm.cs.cmu.edu/anon/2020/CMU-CS-20-113.pdf.

13. Michel Goemans. 18.310 lecture notes: Chernoff bounds, and some applications, 2015 [Електронний ресурс]. — Режим доступу: https://math.mit.edu/~goemans/18310S15/chernoff-notes.pdf.

14. O. Goldreich and L. A. Levin, "A hard-core predicate for all one-way functions", 1989.

15. F. J. MacWilliams and N. J. A. Sloane, The Theory of Error-Correcting Codes, 1977.

16. Ryan O'Donnell. Analysis of Boolean Functions: Cambridge University Press, 2014. — 419 p.

17. L. Bahl, J. Cocke, F. Jelinek, and J. Raviv, "Optimal decoding of linear codes for minimizing symbol error rate (corresp.)," IEEE Transactions on information theory, vol. 20, no. 2, pp. 284–287, 1974.

18. J. Hagenauer, "The turbo principle: Tutorial introduction and state of the art," in Proc. International Symposium on Turbo Codes and Related Topics, 1997, pp. 1–11.

19. Jiang, Yihan, et al. "Turbo autoencoder: Deep learning based channel codes for point-to-point communication channels." Advances in neural information processing systems32 (2019).

20. Devroye, N., et al. "Interpreting Training Aspects of Deep-Learned Error-Correcting Codes." arXiv preprint arXiv:2305.04347 (2023).

ДОДАТОК А ТЕКСТИ ПРОГРАМ

А.1 Програма 1

```
k = 0;
n = 800;
L = randn(1, n);
threshold = 0.5;
delta = 0.99;
for k = 1:n
    a = unifrnd(0, 0.1, 1, k);
    idx = randperm(numel(a),
    fix(numel(a) / 3));
    sbs = ind2sub(size(L), idx);
    S = false(size(L));
    S(sbs) = 1;
    Ys = L .* S;
    W = abs(ifft(Ys));
    gamma = (threshold^2) / 4;
    if any(W > gamma)
        L = [L, W(W > gamma)];
    end
end
L
```