

1 Цель работы

Разработать программный модуль, который строит случайный двоичный линейный блочный код для заданных параметров (n, k) . Для построенного кода оценить минимальное расстояние и построить спектр кода. Указать, на сколько полученные параметры далеки от границ Хемминга, Варшамова-Гилберта и Синглтона.

2 Описание алгоритма

На вход программы подается n и k , причем осуществляется проверка корректности исходных данных.

Затем генерируется порождающая матрица G размером $k \times n$, причем матрица имеет вид:

$$G = [I : C]$$

где I – единичная матрица $k \times k$, C – случайно сгенерированная матрица размера $k \times (n - k)$.

Далее, с помощью формулы размещения с повторениями, генерируются двоичные векторы \bar{t} длины k . Полученные векторы умножаются на порождающую матрицу G . Таким образом формируются кодовые слова \bar{c} . Множество кодовых слов состоит из $|c| = 2^k$ кодовых слов.

С помощью перебора определяются минимальное расстояние d :

$$d = \min (wt(c_i))$$

Зная минимальное расстояние d , можно оценить корректирующую способность кода t , т.е. число гарантированно исправляемых ошибок:

$$t = \left\lfloor \frac{d - 1}{2} \right\rfloor$$

Также по множеству кодовых слов строится спектр кода $A(l), l = 0 \dots n$, где $A(l_i)$ – количество кодовых слов весом l_i .

Необходимо так же оценить число кодовых слов, сравнив с границами Хемминга, Варшамова-Гилберта и Синглтона.

Верхняя граница (граница Хэмминга для двоичного кода):

$$2^k \leq \frac{2^n}{\sum_{i=0}^t C_n^i}$$

Верхняя граница (граница Синглтона для двоичного кода):

$$2^k \leq 2^{n-d+1}$$

Нижняя граница (граница Варшамова-Гилберта для двоичного кода):

$$2^k \geq \frac{2^n}{\sum_{i=0}^{d-1} C_n^i}$$

3 Примеры

Рассмотрим пример построения двоичного линейного блочного кода. Пусть $n = 6$, $k = 3$. Тогда число кодовых слов $|C| = 2^k = 2^3$.

Пусть порождающая матрица G имеет вид:

$$G = \begin{bmatrix} 100 & 111 \\ 010 & 110 \\ 001 & 011 \end{bmatrix}$$

Векторы \vec{i} : 000, 001, 010, 011, 100, 101, 110 и 111, умножая эти векторы на порождающую матрицу G , получаем множество кодовых слов:

$$C = \{ 000000, 100111, 110001, 010110, 001011, 101100, 011101, 111010 \}$$

Тогда спектр кода имеет вид:

$$A(0) = 1$$

$$A(1) = 0$$

$$A(2) = 0$$

$$A(3) = 4$$

$$A(4) = 3$$

$$A(5) = 0$$

$$A(6) = 0$$

Тогда минимальное расстояние $d = 3$, а корректирующая способность $t = 1$.

Оценим близость к границам. Верхняя граница (граница Хэмминга для двоичного кода):

$$8 \leq \frac{64}{7} \text{ или } 8 \leq 9.14$$

Верхняя граница (граница Синглтона для двоичного кода):

$$8 \leq 16$$

Нижняя граница (граница Варшамова-Гилберта для двоичного кода):

$$8 \geq \frac{64}{22} \text{ или } 8 \geq 2.9$$

Все границы соблюдены, что говорит о том, что построенный код, хоть и не является совершенным, но при этом является хорошим¹.

4 Примеры работы программы

4.1 Пример 1

Теперь программа будет генерировать случайную матрицу G , а входные параметры n и k будут задаваться пользователем.

```
Type n: 7
Type k: 3
|c| = 8
1 0 0 0 0 1 0
0 1 0 1 0 1 1
0 0 1 0 1 0 1
```

Рисунок 1 - Порождающая матрица

Векторы \vec{i} : 000, 001, 010, 011, 100, 101, 110 и 111, умножая эти векторы на порождающую матрицу G , получаем множество кодовых слов:

¹ В кодировании «хорошими» называют коды, у которых доля d/n не стремится к нулю при $n \rightarrow \infty$ и $R = \text{const}$.

0	0	0	0	0	0	0
0	0	1	0	1	0	1
0	1	0	1	0	1	1
0	1	1	1	1	1	0
1	0	0	0	0	1	0
1	0	1	0	1	1	1
1	1	0	1	0	0	1
1	1	1	1	1	0	0

Рисунок 2 - Кодовые слова

Тогда спектр кода имеет вид:

```
A[0] = 1
A[1] = 0
A[2] = 1
A[3] = 1
A[4] = 2
A[5] = 3
A[6] = 0
A[7] = 0
```

Рисунок 3 - Спектр кода

Тогда минимальное расстояние $d = 2$, а корректирующая способность $t = 0$.

```
d_min = 2
t = 0
```

Рисунок 4 - Минимальное расстояние и корректирующая способность

Оценим близость к границам.

```
Upper bound = 128.0 true
Bottom line = 16.0 false
singleton bound = 64.0 true
```

Рисунок 5 - Границы мощности кода

Не все границы соблюдены, что говорит о том, что построенный код не является хорошим, т.к. при заданных параметрах n , k и d существует код с большей мощностью.

Все результаты, полученные в ходе работы программы совпадают с теоретическими, что говорит о корректной работе программы.

4.2 Пример 3

Программа генерирует случайную матрицу G , а входные параметры n и k будут задаваться пользователем.

```
Type n: 7
Type k: 3
|c| = 8
1 0 0 0 1 1 1
0 1 0 1 0 0 1
0 0 1 0 0 1 1
```

Рисунок 6 - Порождающая матрица

Векторы \vec{i} : 000, 001, 010, 011, 100, 101, 110 и 111, умножая эти векторы на порождающую матрицу G , получаем множество кодовых слов:

```
0 0 0 0 0 0 0
0 0 1 0 0 1 1
0 1 0 1 0 0 1
0 1 1 1 0 1 0
1 0 0 0 1 1 1
1 0 1 0 1 0 0
1 1 0 1 1 1 0
1 1 1 1 1 0 1
```

Рисунок 7 - Кодовые слова

Тогда спектр кода имеет вид:

```
A[0] = 1
A[1] = 0
A[2] = 0
A[3] = 3
A[4] = 2
A[5] = 1
A[6] = 1
A[7] = 0
```

Рисунок 8 - Спектр кода

Тогда минимальное расстояние $d = 3$, а корректирующая способность $t = 1$.

```
d_min = 3
t = 1
```

Рисунок 9 - Минимальное расстояние и корректирующая способность

Оценим близость к границам.

```
Upper bound = 16.0 true
Bottom line = 4.413793103448276 true
singleton bound = 32.0 true
```

Рисунок 10 - Границы мощности кода

Все границы соблюдены, что говорит о том, что построенный код, хоть и не является совершенным, но при этом является хорошим².

Все результаты, полученные в ходе работы программы совпадают с теоретическими, что говорит о корректной работе программы.

5 Выводы

Таким образом, в ходе выполнения лабораторной работы был разработан программный модуль, который строит случайный двоичный линейный блочный код для заданных параметров (n, k) . Для построенного кода оценить минимальное расстояние и

² В кодировании «хорошими» называют коды, у которых доля d/n не стремится к нулю при $n \rightarrow \infty$ и $R = \text{const}$.

построить спектр кода. Указать, на сколько полученные параметры далеки от границ Хемминга, Варшамова-Гилберта и Синглтона.

Сравнивая теоретические результаты для заданных входных значений с полученными, был сделан вывод о корректной работе программы, а также была дана оценка построенному коду, путем оценки корректирующей способности, спектра кода и сравнения мощности кода с границам.

6 Листинг программы

```
package com.suai;

import java.util.Scanner;

public class LinearBlockCode {

    private int n = 6;
    private int k = 3;
    private byte[][] G;
    private byte[][] c;
    private int cIndex;
    private int dMin;
    private int t;

    public LinearBlockCode() throws Exception {
        Scanner input = new Scanner(System.in);
        System.out.print("Type n: ");
        if (input.hasNextInt()) {
            n = input.nextInt();
        } else {
            throw new Exception("Incorrect n");
        }
        System.out.print("Type k: ");
        input = new Scanner(System.in);
        if (input.hasNextInt()) {
            k = input.nextInt();
        } else {
            throw new Exception("Incorrect k");
        }
        if (k >= n) {
            throw new Exception("k must be < n");
        }
        G = new byte[k][n];
        c = new byte[(int) Math.pow(2, k)][n];
        dMin = (int) Math.pow(2, k);
        System.out.println("|c| = " + c.length);

        matrixGenerate();
        //testingMatrix();
        System.out.println();
        codeWords();
        codeSpectrum();
    }
}
```



```

correctivePower();
upperBound();
bottomLine();
singletonBound();

}

private void testingMatrix() {
    G[0][0] = 1;
    G[0][1] = 0;
    G[0][2] = 0;
    G[0][3] = 1;
    G[0][4] = 1;
    G[0][5] = 1;
    G[1][0] = 0;
    G[1][1] = 1;
    G[1][2] = 0;
    G[1][3] = 1;
    G[1][4] = 1;
    G[1][5] = 0;
    G[2][0] = 0;
    G[2][1] = 0;
    G[2][2] = 1;
    G[2][3] = 0;
    G[2][4] = 1;
    G[2][5] = 1;
    for (int i = 0; i < k; i++) {
        for (int j = 0; j < n; j++) {
            System.out.print(G[i][j] + " ");
        }
        System.out.println();
    }
}

private void matrixGenerate() {
    for (int i = 0; i < k; i++) {
        for (int j = 0; j < n; j++) {
            if (i == j && j < k) {
                G[i][j] = 1;
            }
            if (j >= k) {
                G[i][j] = (byte) (Math.random() * 2);
            }
            System.out.print(G[i][j] + " ");
        }
    }
}

```

```

        System.out.println();
    }
}

private void product(byte[] i) {
    for (int a = 0; a < n; a++) {
        for (int b = 0; b < k; b++) {
            c[cIndex][a] = (byte) (c[cIndex][a] ^ (i[b] & G[b][a]));
        }
    }
    cIndex++;
}

private void A(int a, int k, int cur, byte i[]) {
    if (cur == k) {
        for (int b = 0; b < k; b++) {
            System.out.print(i[b] + " ");
        }
        System.out.println();
        product(i);
    } else {
        for (byte b = 0; b < a; b++) {
            i[cur] = b;
            A(a, k, cur + 1, i);
        }
    }
}

private void codeWords() {
    byte[] i = new byte[k];
    A(2, k, 0, i);
    for (int j = 0; j < c.length; j++) {
        for (int l = 0; l < c[0].length; l++) {
            System.out.print(c[j][l] + " ");
        }
        System.out.println();
    }
}

private void codeSpectrum() {
    int[] A = new int[n + 1];
    for (int i = 0; i < c.length; i++) {
        int d = 0;
        for (int j = 0; j < c[0].length; j++) {
            if (c[i][j] == 1)

```

```

        d++;
    }
    if (d < dMin && d != 0)
        dMin = d;
    A[d]++;
}
for (int i = 0; i < A.length; i++)
    System.out.println("A[" + i + "] = " + A[i]);
System.out.println("d_min = " + dMin);
}

private void correctivePower() {
    t = (dMin - 1) / 2;
    System.out.println("t = " + t);
}

public int getFactorial(int f) {
    int result = 1;
    for (int i = 1; i <= f; i++) {
        result = result * i;
    }
    return result;
}

public double C(int n, int k) {
    return (double) getFactorial(n) / ((getFactorial(k)) * getFactorial(n - k));
}

public void upperBound() {
    int sum = 0;
    for (int i = 0; i <= t; i++) {
        sum += C(n, i);
    }
    System.out.print("Upper bound = " + (Math.pow(2, n) / sum));
    if (Math.pow(2, k) <= (Math.pow(2, n) / sum))
        System.out.println(" true");
    else
        System.out.println(" false");
}

public void bottomLine() {
    int sum = 0;
    for (int i = 0; i < dMin; i++) {
        sum += C(n, i);
    }
    System.out.print("Bottom line = " + (Math.pow(2, n) / sum));
    if (Math.pow(2, k) >= (Math.pow(2, n) / sum))

```

```
        System.out.println(" true");
    else
        System.out.println(" false");
}
```

```
public void singletonBound() {
    System.out.print("singleton bound = " + Math.pow(2, (n-dMin+1)));
    if (Math.pow(2,k) <= Math.pow(2, (n-dMin+1)))
        System.out.println(" true");
    else
        System.out.println(" false");
}
```

```
public static void main(String[] args) {
    try {
        LinearBlockCode l = new LinearBlockCode();
    } catch (Exception e) {
        System.out.print(e.getMessage());
        e.printStackTrace();
    }
}
```

```
}
```