

情報工学実験 2 数理計画法 第二回

学生番号 4617043 神保光洋

2018 年 11 月 13 日

1 実験の要旨

ハミング符号による符号化および符号プログラムの作成

2 実験の目的

誤り率を低くする。

3 実験の原理

$(n, k) = (7, 4)$ ハミング符号を用いる。 $(7, 4)$ ハミング符号とは代表的な誤り訂正符号の一つであり、単一誤りの訂正を可能とする。符号化と呼ばれる操作によって 4 ビットの情報を 7 ビットの符号語に変換し、送信する。

4 実験方法

今回自分は visual studio2013 などという枯れた環境を使うことを嫌い Unix / Linux のような再現可能かつ汎用性の高い環境で実行を行うためあえて MT ではなく rand() 関数を使うこのあたりは評価されたい。ちなみに TA には許可をいただいている。

4.1 実験方法

誤り訂正符号を用いる場合に置いて、 $(7, 4)$ ハミング符号による符号化と符号を行うプログラムを作成する。MT によって発生された乱数により $k = 4$ ビットの $0, 1$ の情報系列 w を生成し、符号化により $n - k = 7 - 4 = 3$ ビットの冗長ビットを算出し、 $n = 7$ ビットの符号語 x を生成する。また $n = 7$ ビットの符号語のうち誤りを付加させるビット位置 (1 か所のみ) を指定できるように、受信系列 y を求める。復号によりどのビット位置が誤っても訂正できることを確認する。その雨に少なくとも複数の誤りパターンに対し実行する

5 実行環境

Mojave 10.14.1 zsh 5.3 (x86_64-apple-darwin18.0) gcc 4.2.1

6 結果

プログラムの実行結果は以下のようになった

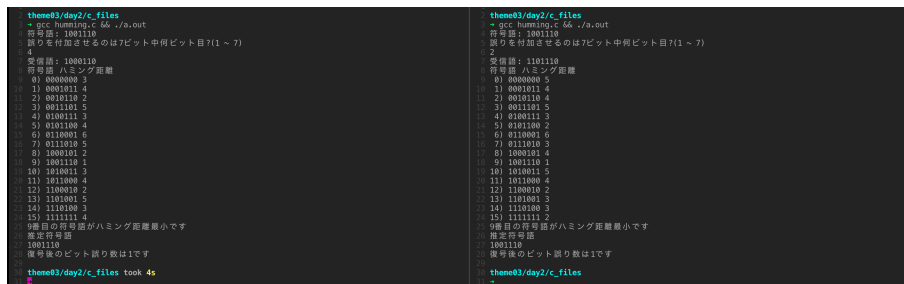


図 1 最急降下法が生成した二次元配列

7 検討事項

7.1 なぜハミング符号は 1 個誤りを訂正できるか

の符号多項式を割り切ることができる $n - k$ 次多項式で、かつ $x^n + 1$ の因数であるものを生成多項式と言う。今回のハミング符号はこれで生成される多項式を割り、そのあまりを求める。これにより符号を分割することができる。誤りが 1 つの場合 16 個の多項式表現を一意に表すことができる、これはハミング符号 1 個誤りを一意に表すことを意味する。

7.2 2 個以上の誤りが発生するとどうなるか。2 個、3 個、... とするとどうなるか。

上にも触れた通り誤りが 1 つの場合 16 個の多項式表現を一意に表すことができるが 2 つ以上であると一意に定めることはできなくなる。しかしどれかどれであると言ったような推定は誤りが少なければできる 2 個、3 個となるほどに受信語 y とのハミング距離は増え誤りの推定値は増える

7.3 まとめ

ハミング距離は自然言語処理などで使われることは知っていたが情報通信でも使われていることを理解した。

また誤が多いと誤り訂正符号は推定できなくなることを実験を通し学んだ。

8 参考文献

参考文献

- [1] 山本和彦 プログラミング Haskell・オーム社
- [2] 高橋麻奈 やさしい C 言語 (第 5 版)・SB クリエイティブ

9 付録

実験で使ったプログラムは以下のようになる。

ソースコード 1 humming.c

```
#include <stdio.h>
#include <stdlib.h>

void n2b(int n, int bi[], int bi_len) {
    if (n == 0) {
        return;
    }
    else {
        bi[bi_len++] = n % 2;
        n2b(n / 2, bi, bi_len);
    }
}

void show_bi(int bi[], int n) {
    int i;
    for (i = 0; i < n; i++) {
        printf("%d", bi[i]);
    }
    printf("\n");
}

void init_bi(int bi[], int n) {
    int i;
    for (i = 0; i < n; i++) {
        bi[i] = 0;
    }
}
```

```

}

void reverse4(int bi[], int n) {
    int tmp[n];

    int i;
    for (i = 0; i < n; i++) {
        tmp[i] = bi[i];
    }
    init_bi(bi, n);
    bi[3] = tmp[0];
    bi[2] = tmp[1];
    bi[1] = tmp[2];
    bi[0] = tmp[3];
}

void add_c(int bi[]) {
    int c_0 = bi[0] ^ bi[1] ^ bi[2];
    int c_1 = bi[1] ^ bi[2] ^ bi[3];
    int c_2 = bi[0] ^ bi[1] ^ bi[3];

    bi[4] = c_0;
    bi[5] = c_1;
    bi[6] = c_2;
}

void cpy_bi2bits(int bits[16][7], int bi[], int i, int n) {
    int j;
    for(j = 0; j < n; j++) {
        bits[i][j] = bi[j];
    }
}

void make_bits(int bits[16][7]) {
    int n = 7;
    int bi[n];
    int i;

```

```

    for (i = 0; i < 16; i++) {
        init_bi(bi, n);
        n2b(i, bi, 0);
        reverse4(bi, n);
        add_c(bi);
        cpy_bi2bits(bits, bi, i, n);
    }
}

int show_start(int w[]) {
    int i;

    printf("符号語: ");
    for (i = 0; i < 7; i++) {
        printf("%d", w[i]);
    }
    printf("\n");

    int input_num;
    printf("誤りを付加させるのは7ビット中何ビット目?(1 ~ 7)\n");
    scanf("%d", &input_num);
    int e_num = input_num - 1;

    return e_num;
}

void show_received(int y[], int n) {
    printf("受信語: ");
    int i;
    for (i = 0; i < n; i++) {
        printf("%d", y[i]);
    }
    printf("\n");
}

void show_decoded(int min_hum_pattern, int bis[16][7], int min_hum) {

```

```

printf("%d 番目の符号語がハミング距離最小です\n", min_hum_pattern);

printf("推定符号語\n");
int i;
for (i = 0; i < 7; i++) {
    printf("%d", bis[min_hum_pattern][i]);
}
printf("\n");

printf("復号後のビット誤り数は%dです\n", min_hum);
}

void hum(int bis[16][7], int y[]) {
    int min_hum = 100000;
    int min_hum_pattern = 0;
    int i, j;

    printf("符号語 ハミング距離\n");
    for (i = 0; i < 16; i++) {
        int hum_num = 0;

        // ハミング距離計算
        for (j = 0; j < 7; j++) {
            if (bis[i][j] != y[j]) {
                hum_num++;
            }
        }

        if (min_hum > hum_num) {
            min_hum = hum_num;
            min_hum_pattern = i;
        }

        // バイナリパターン表示
        printf("%2d) ", i);
        for (j = 0; j < 7; j++) {
            printf("%d", bis[i][j]);

```

```

    }
    printf(" ");

    // ハミング距離表示
    printf("%d", hum_num);
    printf("\n");
}

// 復号結果表示
show_decoded(min_hum_pattern, bis, min_hum);
}

int main(void) {

    int bis[16][7];
    make_bits(bis);

    srand(100);
    double ran = rand();

    int n = 7;
    int k = 4;

    int w[n];
    int e[n];
    int y[n];

    int i;

    for (i = 0; i < k; i++) {
        double rnd = rand();
        if (rnd / RAND_MAX <= 0.5) {
            w[i] = 1;
        }
        else {

```

```

        w[i] = 0;
    }
}

int c_1 = w[0] ^ w[1] ^ w[2];
int c_2 = w[1] ^ w[2] ^ w[3];
int c_3 = w[0] ^ w[1] ^ w[3];

w[4] = c_1;
w[5] = c_2;
w[6] = c_3;

// エラー番号取得
int e_num = show_start(w);

for (i = 0; i < n; i++) {
    if (i == e_num) {
        e[i] = 1;
    }
    else {
        e[i] = 0;
    }
}

for (i = 0; i < n; i++) {
    y[i] = w[i] ^ e[i];
}

// 受信結果表示
show_received(y, n);

// 推定符号語計算
hum(bis, y);

return 0;
}

```