Санкт-Петербургский политехнический университет Петра Великого
Институт компьютерных наук и технологий
Кафедра компьютерных систем и программных технологий

Телекоммуникационные технологии

Отчет по лабораторной работе №7
Помехоустойчивое кодирование

Работу выполнил:
Графов Д.И.
Группа: 33531/2
Преподаватель:
Богач Н.В.

Санкт-Петербург
2019

Содержание

1. Цель работы

- Изучить методы помехоустойчивого кодирования и сравнить их свойства.

2. Задачи работы

- Провести кодирование/декодирование сигнала, полученного с помощью функции randerr кодом Хэмминга 2-мя способами: с помощью встроенных функций encode/decode, а также через создание проверочной и генераторной матриц и вычисление синдрома. Оценить корректирующую способность кода.
- Выполнить кодирование/декодирование циклическим кодом, кодом БЧХ. Оценить корректирующую способность кода.

3. Теоретическая информация

Помехоустойчивое кодирование — кодирование, предназначенное для передачи данных по каналам с помехами, обеспечивающее исправление возможных ошибок передачи вследствие помех. Для обнаружения ошибок используют коды обнаружения ошибок, для исправления — помехоустойчивые коды.

Помехоустойчивое кодирование предполагает введение в передаваемое сообщение, наряду с информационными, так называемых проверочных разрядов, формируемых в устройствах защиты от ошибок (кодерах-на передающем конце, декодерах — на приемном). Избыточность позволяет отличить разрешенную и запрещенную (искаженную за счет ошибок) комбинации при приеме, иначе одна разрешенная комбинация переходила бы в другую.

Помехоустойчивый код характеризуется тройкой чисел (n, k, d0), где n— общее число разрядов в передаваемом сообщении, включая проверочные (r), k=n-r - число информационных разрядов, d0— минимальное кодовое расстояние между разрешенными кодовыми комбинациями, определяемое как минимальное число различающихся бит в этих комбинациях. Число обнаруживаемых (tj и (или) исправляемых (t) ошибок (разрядов) связано с параметром d0 соотношениями.

Коды Хемминга — простейшие линейные коды с минимальным расстоянием 3, то есть способные исправить одну ошибку. Код Хемминга может быть представлен в таком виде, что синдром:

$\vec{s} = \vec{r}H^T$, где $\vec{r}$ — принятый вектор, будет равен номеру позиции, в которой произошла ошибка. Это свойство позволяет сделать декодирование очень простым.

Коды Боуза — Чоудхури — Хоквингема (БЧХ) являются подклассом циклических кодов. Их отличительное свойство — возможность построения кода БЧХ с минимальным расстоянием не меньше заданного.

4. Ход выполнения работы

Данная работа выполнялась на языке Python.

К сожалению, мной не были найдены какие-либо библиотечные решения помехоустойчивого кодирования, поэтому вся работа была проделана на "чистом"Python, за исключением матречных операций с использованием NumPy.

За основу работы был взят материал, полученный на лекции и лабораторном занятии.

С полным кодом можно ознакомиться по адресу:
https://github.com/grafovdenis/telecom/tree/master/lab7/src/

5. Результаты работы

5.1. Код Хэмминга

Рассмотрим функцию encode, реализующую кодирование 4-битного кода в код Хэмминга (7, 4).

Листинг 1. encode

```
6  def encode(num):
7      # Return given 4 bits plus parity bits for bits (0,2,3), (0,1,2) and (0,1,2)
8      b0 = parity(num, [0, 2, 3])
9      b1 = parity(num, [0, 1, 3])
10     b2 = parity(num, [0, 1, 2])
11
12     return [b0, b1, num[3], b2, num[2], num[1], num[0]]  # again saying, works only for 7,4
13
14
15 def parity(s, indexes):
16     return int(s[indexes[0]]) ^ int(s[indexes[1]]) ^ int(s[indexes[2]])
```

Листинг 2. Результат работы encode

```
1   Source  |  Encoded
2  [0 1 0 0] [0, 1, 0, 1, 0, 1, 0]
3  [0 0 1 1] [0, 1, 1, 1, 1, 0, 0]
4  [1 1 0 0] [1, 0, 0, 0, 0, 1, 1]
5  [0 0 1 1] [0, 1, 1, 1, 1, 0, 0]
6  [1 1 0 1] [0, 1, 1, 0, 0, 1, 1]
7  [0 0 1 1] [0, 1, 1, 1, 1, 0, 0]
8  [0 1 1 0] [1, 1, 0, 0, 1, 1, 0]
9  [1 0 0 1] [0, 0, 1, 1, 0, 0, 1]
10 [0 0 0 0] [0, 0, 0, 0, 0, 0, 0]
11 [1 1 1 1] [1, 1, 1, 1, 1, 1, 1]
12 [0 0 1 0] [1, 0, 0, 1, 1, 0, 0]
13 [0 0 1 0] [1, 0, 0, 1, 1, 0, 0]
14 [0 0 0 1] [1, 1, 1, 0, 0, 0, 0]
15 [1 0 1 1] [1, 0, 1, 0, 1, 0, 1]
16 [1 1 0 0] [1, 0, 0, 0, 0, 1, 1]
17 [0 0 0 1] [1, 1, 1, 0, 0, 0, 0]
```

Рассмотрим кодирование конкретного примера 1011 и вычисление синдрома.

Листинг 3. Вспомогательные функции для вычисления синдрома, нахождения и исправления ошибки

```
19 def get_syndrome_num(num):
20     s0 = syndrome(num, [0, 2, 4, 6])
21     s1 = syndrome(num, [1, 2, 5, 6])
22     s2 = syndrome(num, [3, 4, 5, 6])
23     result = s0 + s1 * 2 + s2 * 4
24     return result
25 
26 
27 def syndrome(num, indexes):
28     return int(num[indexes[0]]) ^ int(num[indexes[1]]) ^ int(num[indexes[2]] ^ int(num[indexes[3]]))
29 
30 
31 # recover encoded message with error in (syndrome) digit
32 def recover(num, syndrome):
33     if syndrome == 0:
34         return num
35     result = num
36     result[syndrome] = 0 if result[syndrome] == 1 else 1
37     return result
```

Листинг 4. Запуск примера

```
52     num = np.array([1, 0, 1, 1])
53     encoded = encode(num)
54     print("Syndrome for correct message:", get_syndrome_num(encoded))
55     print("Syndrome example for:", num)
56     print(encoded)
57     print("Let error be in 6 digit:")
58     encoded[6] = 0 if encoded[6] == 1 else 1
59     print(encoded)
60     print("Error in", get_syndrome_num(encoded) - 1, "digit")
61     print("Recovered message:")
62     print(recover(encoded, get_syndrome_num(encoded) - 1))
63     print("\n\n")
```

Листинг 5. Результат работы

```
1 Syndrome for correct message: 0
2 Syndrome example for: [1 0 1 1]
3 [1, 0, 1, 0, 1, 0, 1]
4 Let error be in 6 digit:
5 [1, 0, 1, 0, 1, 0, 0]
6 Error in 6 digit
7 Recovered message:
8 [1, 0, 1, 0, 1, 0, 1]
```

Далее рассмотрим пример с пораждающей матрицей.

Листинг 6. Пример с пораждающий матрицей

```
65      print("Generator matrix example:")
66      num_matrix = np.matrix([[0, 1, 0, 1]])
67      print(num_matrix)
68
69      gm = np.matrix([[1, 0, 0, 0, 1, 1, 0],
70                      [0, 1, 0, 0, 1, 0, 1],
71                      [0, 0, 1, 0, 0, 1, 1],
72                      [0, 0, 0, 1, 1, 1, 1]])
73
74      encoded_matrix = num_matrix * gm
75
76      encoded_matrix = encoded_matrix.transpose()
77
78      for i in range(encoded_matrix.__len__()):
79          encoded_matrix[i] = encoded_matrix[i] % 2
80
81      encoded_matrix = encoded_matrix.transpose()
82      print("Encoded matrix:", encoded_matrix)
83
84      damaged = np.matrix([[0, 1, 0, 1, 0, 1, 1]])
85
86      print("Damaged matrix:", damaged)
87
88      b = np.matrix([[0, 1, 1],
89                     [1, 0, 1],
90                     [1, 1, 0],
91                     [1, 1, 1],
92                     [0, 0, 1],
93                     [0, 1, 0],
94                     [1, 0, 0]]
95                    )
96
97      encoded_error = np.flip(encoded_matrix * b)
98      damaged_error = np.flip(damaged * b)
99
100     # transform indexes from [2 4 5 6 0 1 3] to [0 1 2 3 4 5 6]
101     result = transform(encoded_matrix)
102     damaged_result = transform(damaged)
103
104     print("Result matrix :", result)
105     print("Damaged result:", damaged)
106
107     for i in range(damaged_error.__len__()):
108         damaged_error[i] = damaged_error[i] % 2
109
110     for i in range(encoded_error.__len__()):
111         encoded_error[i] = encoded_error[i] % 2
```

```
112
113     dmg_err = damaged_error.item(2) * 4 + damaged_error.item(1) * 2 + damaged_error.item(0)
114     enc_err = encoded_error.item(2) * 4 + encoded_error.item(1) * 2 + encoded_error.item(0)
115
116     print("Syndrome of correct message:", enc_err)
117     print("Syndrome of damaged message:", dmg_err, "(means that mistake is in 3rd digit)")
```

Листинг 7. Результат работы

```
1 Generator matrix example:
2 [[0 1 0 1]]
3 Encoded matrix: [[0 1 0 1 0 1 0]]
4 Damaged matrix: [[0 1 0 1 0 1 1]]
5 Result matrix : [[0 0 1 0 0 1 1]]
6 Damaged result: [[0 1 0 1 0 1 1]]
7 Syndrome of correct message: 0
8 Syndrome of damaged message: 4 (means that mistake is in 3rd digit)
```

5.2. Циклический код

Рассмотрим пример работы с циклическим кодом для числа 1010.
Листинг 8. cyclical.py

```
# https://habr.com/ru/post/357666/
import numpy as np

# k = d * G
d = np.array([1, 0, 1, 0])
print("Cyclical code for:", d)

G = np.matrix([[1, 0, 1, 1, 0, 0, 0],
               [0, 1, 0, 1, 1, 0, 0],
               [0, 0, 1, 0, 1, 1, 0],
               [0, 0, 0, 1, 0, 1, 1]])

k = d * G
for index, el in enumerate(k):
    k[index] = el % 2

print(k)

H = np.matrix([[1, 0, 1],
               [1, 1, 1],
               [1, 1, 0],
               [0, 1, 1],
               [1, 0, 0],
               [0, 1, 0],
               [0, 0, 1]])

s = k * H

for index, el in enumerate(s):
    s[index] = el % 2

print("Syndrome for correct message  :", s)

k = np.matrix([[1, 0, 0, 1, 1, 1, 1]])

s = k * H

for index, el in enumerate(s):
    s[index] = el % 2

print("Syndrome for incorrect message:", s)
```

Листинг 9. Результат работы

```
Cyclical code for: [1 0 1 0]
[[1 0 0 1 1 1 0]]
Syndrome for correct message  : [[0 0 0]]
Syndrome for incorrect message: [[0 0 1]]
```

6. Выводы

В ходе данной работы на языке Python были написаны функции для кодирования в код Хэмминга (7, 4) и вычисления синдома, был рассмотрен пример с пораждающей матрицей, а также с циклическим кодом.

Таким образом мной было рассмотрено помехоустойчивое кодирование сигнала с помощью различных методов.