The application of noisy-channel coding techniques to DNA barcoding (Early structure/ideas)

Izaak van Dongen

April 2, 2018

Contents

Listings

1 Introduction

The premise of this project is to investigate the different types of error-correcting codes, and how these might be applied to DNA barcoding. The challenge in this comes from the fact that most error-correcting codes are designed in base-2 (binary) whereas DNA strings are fundamentally base-4 (quaternary). The applicability of this project is that in oligonucleotide synthesis, some samples may need to be identified later on using a subsection of the sample (a barcode). These could just be linearly assigned codes, but this would leave them very susceptible to mutation.

Here is an example: say that we're given a barcode of length four, to encode two different samples. If we worked methodically up from the bottom (using the ordering ACGT - orderings will be discussed further later on) we might end up with the codes AAAA and AAAC. However, either string would only require a single mutation (where we say a mutation is the changing of a single base) to become identical to the other

one. Therefore, in this case, it would clearly be far more optimal to make a choice like, for example, AAAA and CCCC.

There have been a few assumptions and glossed over definitions here:

- What constitutes a mutation?
- What is the best way to represent DNA mathematically?

There are also a number of parameters to the problem, and as they change the problem becomes very much nontrivial:

- What if the barcode size changes?
- What if we want more codes than two?
- What if rather than number of codes and barcode size, the parameters are set to barcode size and maximum number of mutations that can occur?

All of these will be further explored in this dissertation.

2 The Hamming distance

The Hamming distance is a measure of "string distance". String distance is a way to define how different two string are. Coding-theoretically, this can be used to quantify the amount that a string has been changed by transmission (or an oligonucleotide has been mutated).

The Hamming distance between any two equally long strings S and R is given by the number of characters at identical position that differ. For example, the distances

$$d(S, R) = 1$$
$$d(S, T) = 2$$

where

$$S = \texttt{abcde}$$
$$R = \texttt{abcfe}$$
$$T = \texttt{axcze}$$

Note that for any S, $d(S, S) = 0$. This means that there is no "distance" from a string to itself.

In terms of DNA, the Hamming distance can be used to determine the number of bases that have mutated.

3 Parity codes

A simple but inefficient parity encoding scheme is a column/row wise encoding. Take the slightly contrived data string "0100000101010100". This is very tangentially related to DNA - it's the 8-bit ASCII representation of the string "AT", generated by the Python: `"".join("0" + bin(ord(c))[2:] for c in "AT")`

Anyway, the string is then split into a square like so:

$$\begin{matrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 \end{matrix}$$

An extra row and column, including an extra corner piece is appended like so:

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

Each of the extra bits documents the pairty of its row.

4 The Hamming code

The Hamming code is a binary encoding scheme that uses parity to detect, and correct errors. The insertion of "parity bits" is a common practice in basic encoding. Parity refers to the "oddness" or "evenness" of some data. Commonly, this is determined by the sum of the data modulo 2. For example, "00101" results in a parity bit of 0, because the sum of all the bits is 2, which has a remainder of 0 when divided by 2 (is equal to 0 mod 2).

5 Implementing the Hamming code

The script implementing a simple binary Hamming code is as follows:

```
#!/usr/bin/env python3

"""
Hamming encoding framework for binary objects, using even parity.
"""

from itertools import count, takewhile

def powers_to(n):
    return takewhile(lambda x: x < n, (1 << i for i in count()))

def hamming_encode(bin_stream):
    pwr = 1
    out = []

    for bit in bin_stream:
        while len(out) + 1 == pwr:
            pwr <<= 1
            out.append(0)
        out.append(bit)

    for i in powers_to(len(out)):
        out[i - 1] = 1 & sum(out[pbit] for pstart in range(i - 1, len(out), i << 1)
                                       for pbit in range(pstart, pstart + i))
    return out
```

Listing 1: Binary Hamming code in Python

This code is accompanied by the following testing scheme:

```
"""
Unit tests for binary_hamming.py
"""

import unittest
```

```
6
7  from binary_hamming import powers_to, hamming_encode
8
9  class BinaryHammingTestCase(unittest.TestCase):
10     def test_powers_to(self):
11         self.assertEqual(list(powers_to(0)), [])
12         self.assertEqual(list(powers_to(1)), [])
13         self.assertEqual(list(powers_to(2)), [1])
14         self.assertEqual(list(powers_to(4)), [1, 2])
15         self.assertEqual(list(powers_to(5)), [1, 2, 4])
16         self.assertEqual(list(powers_to(13)), [1, 2, 4, 8])
17
18     def test_hamming_encode(self):
19         self.assertEqual(hamming_encode([1, 0, 1, 1]), [0, 1, 1, 0, 0, 1, 1])
20
21 if __name__ == "__main__":
22     unittest.main()
```

Listing 2: binary_hamming unit tests

6 The Hadamard code

7 Source

All code and source TEX/LATEX files can be found at `https://github.com/elterminad0r/EPQ`.

References

Assmus, E. F. and Key, J. D. [1992], 'Hadamard matrices and their designs: A coding-theoretic approach', *Transactions of the American Mathematical Society* **330**(1), 269–293.
URL: *http://www.jstor.org/stable/2154164*

Baylis, J. [2010], 'Codes, not ciphers', *The Mathematical Gazette* **94**(531), 412–425.
URL: *http://www.jstor.org/stable/25759725*

Bystrykh, L. V. [2012], 'Generalized dna barcode design based on hamming codes', *PLOS ONE* .
URL: *http://journals.plos.org/plosone/article?id=10.1371/journal.pone.0036852*

del Río, Á. and Rifà, J. [2012], 'Families of hadamard z2z4q8-codes', *CoRR* **abs/1211.5251**.
URL: *http://arxiv.org/abs/1211.5251*

Ehrenborg, R. [2006], 'Decoding the hamming code', *Math Horizons* **13**(4), 16–17.
URL: *http://www.jstor.org/stable/25678619*

Golomb, S. W. and Baumert, L. D. [1963], 'The search for hadamard matrices', *The American Mathematical Monthly* **70**(1), 12–17.
URL: *http://www.jstor.org/stable/2312777*

Guruswami, V. [2010], 'Introduction to coding theory'.
URL: *http://www.cs.cmu.edu/ venkatg/teaching/codingtheory/notes/notes1.pdf*

Hamming, R. W. [1950], 'Error detecting and error correcting codes', *The Bell System Technical Journal* **26**(2), 147–160.
URL: *http://sb.fluomedia.org/hamming/*

Hedayat, A. and Wallis, W. D. [1978], 'Hadamard matrices and their applications', *The Annals of Statistics* **6**(6), 1184–1238.
URL: *http://www.jstor.org/stable/2958712*

Kneale, W. [1956], 'Boole and the algebra of logic', *Notes and Records of the Royal Society of London* **12**(1), 53–63.
URL: *http://www.jstor.org/stable/530792*

Oztas, E. S. and Siap, I. [2013], 'Lifted polynomials over F_{16} and their applications to dna codes', *Filomat* **27**(3), 459–466.
URL: *http://www.jstor.org/stable/24896375*

Petoukhov, S. V. [2008], The degeneracy of the genetic code and hadamard matrices.
URL: *https://arxiv.org/pdf/0802.3366.pdf*

Pless, V. [1978], 'Error correcting codes: Practical origins and mathematical implications', *The American Mathematical Monthly* **85**(2), 90–94.
URL: *http://www.jstor.org/stable/2321784*

Shannon, C. E. [1948], 'A mathematical theory of communication', *The Bell System Technical Journal* **27**, 379–423, 623–656.
URL: *http://affect-reason-utility.com/1301/4/shannon1948.pdf*

Spence, E. [1972], 'Hadamard designs', *Proceedings of the American Mathematical Society* **32**(1), 29–31.
URL: *http://www.jstor.org/stable/2038298*

Trinh, Q. and Fan, P. [2008], 'Construction of multilevel hadamard matrices with small alphabet', **44**, 1250 – 1252.

Yu, Q., Maddah-Ali, M. A. and Avestimehr, A. S. [2017], 'Polynomial codes: an optimal design for high-dimensional coded matrix multiplication', *CoRR* **abs/1705.10464**.
URL: *http://arxiv.org/abs/1705.10464*