Error-Correcting Codes Summative Assessment

George Koulieris

Computational Thinking 2019-20

1 Introduction

During the three practical sessions, you will be studying repetition and Hamming codes. The content necessary to complete this assignment will be taught in Lectures 1 to 4.

You need to submit one deliverable: a **single python file** including all your functions. The deadline to submit is **2020-05-04 at 14:00**. Submission is done by uploading to DUO.

Your python file must run on **Python 3.7.4** (available on MDS). In order to mark your assignments, I will use a script that runs them against different inputs. If I encounter any issue opening your file or running it, you may get a total mark of zero. A script indicating how your work will be marked will be provided in DUO; you should use it to test whether you are using the right function names and data formats.

You should copy the following function into your own python file (its code can be found on DUO): `hammingGeneratorMatrix(r)` returns the generator matrix of the $(2^r - 1, 2^r - r - 1, 3)$-Hamming code. Example:

```
>>> hammingGeneratorMatrix(3)
[[1, 1, 1, 0, 0, 0, 0], [1, 0, 0, 1, 1, 0, 0], [0, 1, 0, 1, 0, 1, 0],
[1, 1, 0, 1, 0, 0, 1]]
```

This function uses the helper function `decimalToVector(n,r)` which you will have to code yourself (see below).

2 Helper function

First, you need to implement decimal to vector conversion:
(10 marks) `decimalToVector(i,l)`, where i and l are two integers ($0 \leq i < 2^l$): returns a vector of l bits representing i. This function is used in `hammingGeneratorMatrix(r)`. Example:

```
>>> i = 25
>>> l = 6
>>> decimalToVector(i,l)
[0, 1, 1, 0, 0, 1]
```

3 Functions for repetition codes

Then, you need to implement an encoder and a decoder for the repetition codes. Again, you **must** use the correct name and the correct inputs for each function (case sensitive); if you do not, I will not be able to use your function and you will be given a mark of 0 for that function. The decoder should either return a vector of length 1 or an empty list in case of failure.

1. **(10 marks)** `repetitionEncoder(m,n)`, where **m** is a vector of length 1 and n a positive integer: encoder for repetition codes. Example:

```
>>> m = [0]
>>> n = 4
>>> c = repetitionEncoder(m,n)
c = [0, 0, 0, 0]
```

2. **(10 marks)** `repetitionDecoder(v)`, where $\mathbf{v}$ is a vector of any positive length: decoder for repetition codes. Example:

```
>>> v = [1, 1, 0, 0]
>>> m = repetitionDecoder(v)
m = []
>>> w = [1, 0, 0, 0]
>>> m = repetitionDecoder(w)
m = [0]
```

4 Functions for Hamming codes

Finally, encoding and decoding for Hamming codes, that have one issue: they can only encode messages of length k, where $k = 2^r - r - 1$ for some $r \geq 2$. In this assignment, you will have to convert any vector into an encodable message. This must be done as follows. Let $\mathbf{a}$ be a vector of length $\ell \geq 1$. Choose $r \geq 2$ as small as possible so that $k - r = 2^r - 2r - 1 \geq \ell$, then let $\mathbf{m} = (m_1, \ldots, m_k)$ such that:

- $(m_1, \ldots, m_r)$ represents ℓ in binary,
- $(m_{r+1}, \ldots, m_{r+\ell}) = \mathbf{a}$,
- $(m_{r+\ell+1}, \ldots, m_k) = (0, \ldots, 0)$.

Here are a few examples:

$\mathbf{a}$	ℓ	r	k	$\mathbf{m}$
(1)	1	3	4	$(0,0,1,1)$
$(0,0,1)$	3	4	11	$(0,0,1,1,0,0,1,0,0,0,0)$
$(0,1,1,0)$	4	4	11	$(0,1,0,0,0,1,1,0,0,0,0)$
$(1,1,1,1,0,1)$	6	4	11	$(0,1,1,0,1,1,1,1,0,1,0)$

You need to implement the following five functions for Hamming codes. You **must** use the correct name and the correct inputs for each function (case sensitive); if you do not, I will not be able to use your function and you will be given a mark of 0 for that function. For each function, an invalid input will make the function return an empty list.

1. **(10 marks)** `message(a)`, where $\mathbf{a}$ is a vector of any positive length: converts it to a message for a Hamming code. Example:

```
>>> a = [0, 1, 1, 0, 1]
>>> m = message(a)
m = [0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0]
```

2. **(10 marks)** `hammingEncoder(m)`, where $\mathbf{m}$ is a vector of length $2^r - r - 1$ for some $r \geq 2$: encoder for Hamming codes. Example:

```
>>> l = [1, 1, 1]
>>> c = hammingEncoder(l)
c = []
```

```
>>>
>>> m = [1, 0, 0, 0]
>>> c = hammingEncoder(m)
c = [1, 1, 1, 0, 0, 0, 0]
```

3. **(20 marks)** `hammingDecoder(v)`, where $\mathbf{v}$ is a vector of length $2^r - 1$ for some $r \geq 2$: decoder for Hamming codes. You may use any of the three decoders seen in Lecture 4. Example:

```
>>> u = [1, 0, 1, 1]
>>> c = hammingDecoder(u)
c = []
>>>
>>> v = [0, 1, 1, 0, 0, 0, 0]
>>> c = hammingDecoder(v)
c = [1, 1, 1, 0, 0, 0, 0]
```

4. **(10 marks)** `messageFromCodeword(c)`, where $\mathbf{c}$ is a vector of length $2^r - 1$ for some $r \geq 2$: recovering the message from the codeword of a Hamming code. Example:

```
>>> b = [1, 1, 0, 1]
>>> m = messageFromCodeword(b)
m = []
>>>
>>> c = [1, 1, 1, 0, 0, 0, 0]
>>> m = messageFromCodeword(c)
m = [1, 0, 0, 0]
```

5. **(20 marks)** `dataFromMessage(m)`, where $\mathbf{m}$ is a vector of length $2^r - r - 1$ for some $r \geq 2$: recovering the raw data from the message. Example:

```
>>> l = [1, 0, 0, 1, 0, 1, 1, 0, 1, 0]
>>> a = dataFromMessage(l)
a = []
>>>
>>> l = [1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 0]
>>> a = dataFromMessage(l)
a = []
>>>
>>> m = [0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0]
>>> a = dataFromMessage(m)
a = [0, 1, 1, 0, 1]
```