COMP526 - Assignment 2 - Exam Codes

Rob Lamprell - 201459448

Due Date: 12th May 2021

Introduction

In this assignment we are tasked with creating a binary code which can be communicated between two people in order to maximise the score on an exam. The exam is twenty questions long but, our code is restricted to only 10 bits (true/false). The challenge is in creating a code, which can be encoded and decoded using a prearranged cipher, in order to guarantee the highest worst mark. All algorithms are provided in the file **examSolutions.py**.

Algorithms

Naive Solution

The simplest and most obvious solution is to pass the answers as is. This could be the first ten answers, the last ten answers or some pre-defined ordering. Regardless of the selection, the worst possible outcome will always be ten correct answers. This is implemented under the method, **simple()**. As an aside, it is most probable that, if someone is given ten correct answers and then is asked to complete a further ten questions with a 50% chance of getting each correct they would most likely reach a result of 75%. However when probability is involved, outcomes are not guaranteed.

Rule:	Do not use a one-to-one mapping on all instances as this limits the expressivity of the encoding.

Groupings

This section discusses grouping answers together.

Twos

Given that our exam is length 20 and our code is length 10, it might make sense to combine two answers into one. For clarity see Tables-1, 2 and 3 below:

Index	Exam
0	1
1	1
2	0
3	0
4	1
5	1
...	...
18	0
19	1

Table 1: Exam Answers Grouped in Twos

⇒

Index	Encoding
0	1
1	0
2	1
...	1
9	0

Table 2: Encoded Exam Answers

⇒

Index	Decoding
0	1
1	1
2	0
3	0
4	1
5	1
...	...
18	1
19	1

Table 3: Decoded Solution

Although this removes the guessing element, by having all 20 answers accounted for, it still produces a worst case mark of 10 correct answers as each grouping of 2 has a worst case of 50% accuracy. This can be seen in indexes 18 & 19 of Table-1 and Table-3. The is implementation of this can be found under the method, **sets_of_two()**.

Threes

If grouping on twos leaves a worst case 50% accuracy on each grouping, then grouping in threes may provide a better solution. Because the selection is odd, there cannot be a case of equal representation within a subset. Taking this into account we can assume that the worst possible outcome, grouping to grouping, is 66% ($\frac{2}{3}$ correct).

Index	Exam
0	1
1	1
2	1
3	0
4	0
5	0
...	...
17	0
18	1
19	0

Table 4: Exam Answers Grouped in Threes

⇒

Index	Encoding
0	1
1	0
...	1
6	0

Table 5: Encoded Exam Answers

⇒

Index	Decoding
0	1
1	1
2	1
3	0
4	0
5	0
...	...
17	0
18	0
19	0

Table 6: Decoded Solution

Using the implementation in the method **sets_of_three()**, we get a worst case of 12 correct answers. Therefore, we have improved on both our naive and two-grouping solutions. Thus, we can assume that odd numbered groupings perform better than even numbered groupings. Finally, notice that we also only needed to use 6 of the 10 available bits to fully represent the 20-bit exam - we will come back to this later.

Rule:	Do not use even groupings as there is no dominant representative for the subset - limiting to 50% accuracy.

Fours

Because this is even we are going to have the same issue has Twos - cases where groupings do not have a dominate representation of 1 or 0. Therefore, we will skip this.

Fives

Our aim here is to establish whether or not increasing the set size helps or hinders the worst case performance.

Index	Exam
0	1
1	1
2	1
3	1
4	1
...	...
15	0
16	0
17	0
18	1
19	1

Table 7: Exam Answers Grouped in Fives

⇒

Index	Encoding
0	1
1	1
2	1
3	0

Table 8: Encoded Exam Answers

⇒

Index	Decoding
0	1
1	1
2	1
3	1
4	1
...	...
15	0
16	0
17	0
18	0
19	0

Table 9: Decoded Solution

Using the implementation under the method **sets_of_five()**, we can see that this also produces a worst case of 12 correct answers. This makes sense if we perform the basic calculations. Consider that we have 4 sets containing 5 bits each. The worst case is shown in indexes 15, 16, 17, 18 and 19 of Table-7, where the accuracy is 60%, or $\frac{3}{5}$ correct. Therefore our worst case total can be given as:

$$4 * (3) = 12$$

Notice both that this is the same result as Threes but also requires 2 fewer bits - needing 4 bits to represent the entire exam set. It seems wasteful, in both cases, to leave these bits unused.

Rule:	Higher groupings of answers do not necessarily lead to degradation in the worst case outcome.

Unused bits

Taking the Threes and Fives solutions as test cases, we are going to explore the impact of utilising the remaining bits to provide one-to-one mappings.

For groupings of Threes, we have a spare 4 bits. If we perform the mapping as usual and then use the last 4 bits to map directly to the final four questions of the exam, we can guarantee those four will be correct. This means we will have a worst case of $5(2) + 4$. Which gives us the worst case of 14 - an improvement on the 12 we obtained earlier.

However, in this instance we are still being wasteful. Consider the final grouping of three at indexes 15, 16 and 17. We are overwriting two of these with our uncompressed answers, meaning index 15 is a grouping of three with only a single unchanged value. We may as well claim this bit back and also have it be an uncompressed mapping. This way our worst case possible is $5(2) + 5 = 15$. An example of this is shown below in Figure-1.

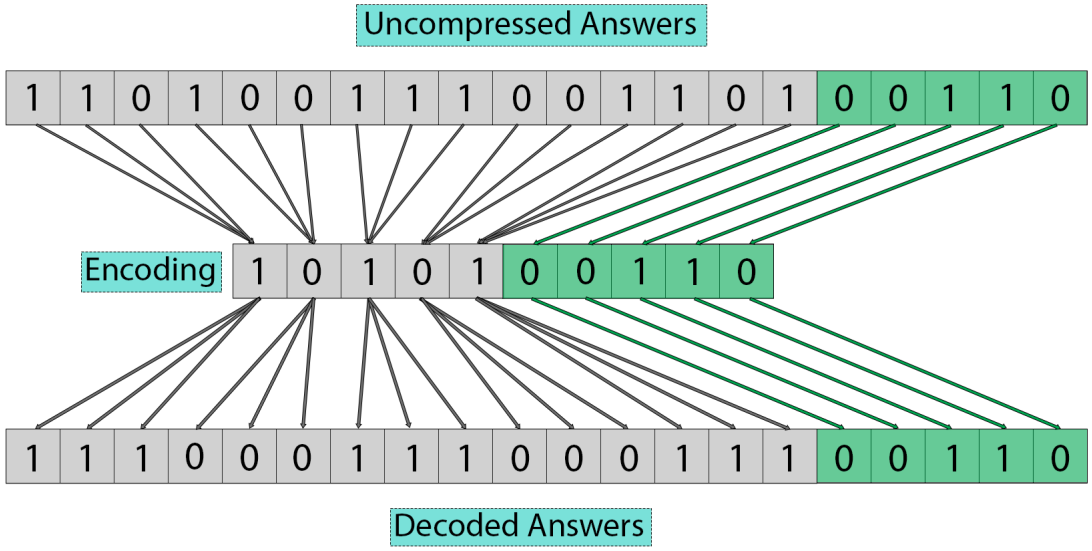

Figure 1: Three Group Mapping With Five Uncompressed Exam Answers - method implementation **sets_of_5threes_fill()**.

Now we will do the same with groupings of five. Recall we had used 4 bits of memory to store all 20 questions leaving 6 bits spare. Though in actuality, we now only need to use 3 bits as the fourth set is defined by the uncompressed mappings. This gives us 7-bits. However, if we calculate this, we are in excess of our 20-bits:

$$3(5) + 7 = 22(bits)$$

This is still wasteful, as the third set of five is now using losing 2-bits to uncompressed mappings, but it still using them determine its value. If we instead remove our seven uncompressed mappings from the 20-bit output, we have 13 bits requiring representation - for the three remaining code bits.

$$20 - 7 = 13(bits)$$

The best way to split the remaining 13 bits, is with two sets of five, one set of three. Putting this together we get full bit utilisation:

$$2(5) + 1(3) + 7 = 20(bits)$$

This mapping is described in Figure-2 below. And should produce a worst case outcome of:

$$2(3) + 1(2) + 7 = 15$$

This is implemented under the methods named **sets_of_2fives_1three_fill()** and provides a worst case of 15, as predicted. Therefore, we can say that increasing the size of the groupings does not necessarily decrease the worst case result. Which allows more space for one-to-one mappings of correct answers.

Rule:	Maximise the number of one-to-one mappings.

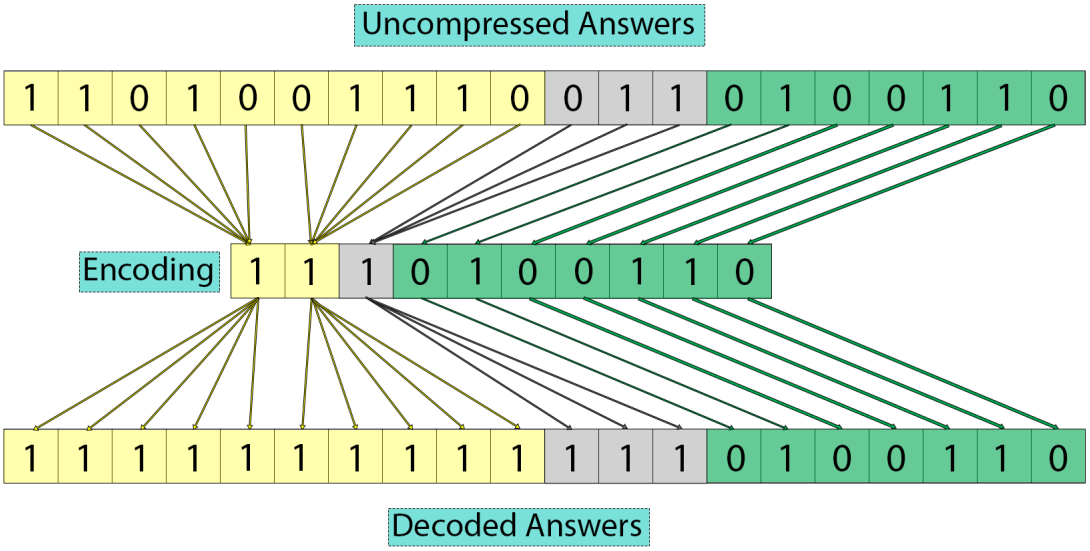

Figure 2: Two Five Group Mappings, One Three Group Mapping and Seven Uncompressed Exam Answers - method implementation **sets_of_2fives_1three_fill()**.

Combining the Rules

We can now attempt to construct a encoding and decoding method using the rules we have established so far:

- (1) Maximise the number of one-to-one mappings.
- (2) Do not use a one-to-one mapping on all instances.
- (3) Do not use an even grouping as there is no dominant representative for the set.
- (4) Higher Groupings of answers do not necessarily lead to a degradation in the minimum outcome.

(1) The maximum number of one-to-one mappings we can achieve without breaking rules (2) is the length of our code minus one (10 - 1 = 9). This leaves us with a subset of 11 remaining questions, which satisfies (3) - odd allowing for a dominant representative of the subset. Our worst case result from this subset would be 6 correct and 5 incorrect. This gives us a minimum mark of:

$$min(score) = len(code) - 1 + \frac{(len(answer) - len(code) - 1)}{2}$$

$$= 9 + \frac{11}{2}$$

$$= 14.5$$

$$= 15 \text{ (impossible to get half a mark, so round up)}$$

This matches our best results so far, combines all previously established rules and its implementation, in the method **max_actuals_largest_set()**, is much simpler than using either a combination of:

- two-five-groupings, one-three-grouping and seven-one-to-one-mappings
- or five-three-groupings and five-one-to-one-mappings

Due to its simplicity, I would choose this one.

Hamming Codes

I feel as though the assignment is hinting towards using Hamming Codes as it explicitly calls out Hamming distance as the metric of measurement. Perhaps involving some form of error correction would provide better results than simply keeping a one-to-one mapping on unused bits. However, due to time constraints I was unable to explore this.

Furthermore, it may be prudent to utilise some form error checking. In case a message is distorted in some way, such as separate passing vehicle using its horn at an inopportune time.

Lower Bounds

The lecture notes discuss lower bounds of bitstring codewords, namely the Hamming bound.

$$2^k \leq \frac{2^n}{\sum_{f=0}^{\frac{d-1}{2}} \binom{n}{f}}$$

$$2^k(\sum_{f=0}^{\frac{d-1}{2}} \binom{n}{f}) \leq 2^n$$

First let's prove the best solution we know of, 15:

$$2^{10} \times (20C0 + 20C1 + 20C2 + 20C3 + 20C4 + 20C5) \leq 2^{20}$$

$$2^{10} \times (1 + 20 + 190 + 1140 + 4845 + 15504) \leq 1,048,576$$

$$1024 \times (21,700) \leq 1,048,576$$

$$22,220,800 \not\leq 1,048,576$$

$$\therefore Possible$$

The condition holds. Next we try 16:

$$2^{10} \times (1 + 20 + 190 + 1140 + 4845) \leq 1,048,576$$

$$1024 \times (6,196) \leq 1,048,576$$

$$6,344,704 \not\leq 1,048,576$$

$$\therefore Possible$$

Now 17:

$$2^{10} \times (1 + 20 + 190 + 1140) \leq 1,048,576$$

$$1024 \times (1,351) \leq 1,048,576$$

$$1,383,424 \not\leq 1,048,576$$

$$\therefore Possible$$

Finally 18:

$$2^{10} \times (1 + 20 + 190) \leq 1,048,576$$

$$1024 \times (211) \leq 1,048,576$$

$$216,064 \leq 1,048,576$$

$$\therefore Impossible$$

Here we have our first deviation. Therefore our lower-bound for the worst score we could potentially guarantee is 17 - two higher than we were able to find. But that is not to say there is definitely a solution which can obtain a worst case of 17, as there may be a way to disprove this possibility - the same may also be true for a result of 16.

Summary

The best solution obtained is 15. Which can be achieved using these functions within **examSolutions.py**:

- sets_of_5threes_fill()
- sets_of_2fives_1three_fill()
- max_actuals_largest_set() – set as default in the python code

There is possibly one or more solutions which are able to attain a result of 17.