Massachusetts Institute of Technology
Department of Electrical Engineering and Computer Science
Department of Mechanical Engineering

Problem Set 2 Solutions

Solution to Problem 1: It's in the Genes

Solution to Problem 1, part a.

How many bits do you need to specify a single codon? Since each codon contains three nucleotides, and there are four nucleotides, there are 4^3 or 64 possible codons. To encode this, we need 6 bits, since $2^6 = 64$.

Solution to Problem 1, part b.

How many bits do you need to encode the sample of DNA? Since each codon needs 6 bits, and there are 500 codons, we need 3000 bits.

Solution to Problem 1, part c.

How may bits do you need to specify an amino acid? 4 bits is enough to encode 16 different possibilities. Five bits is enough for 32. So we need at least five bits.

Solution to Problem 1, part d.

If you represent the information as a sequence of 500 amino acids, how many bits do you need to encode the protein? Each amino acid needs at least five bits, so 2500 bits are necessary.

Solution to Problem 1, part e.

Encoding amino acids directly saves space, in the case, 500 bits. However, encoding with codons, depending on how it is done, may provide some robustness against errors. In fact, each amino acid may have more than one codon which maps to it, or some codons may have no corresponding amino acids. Thus, if we are worried about errors in our encoding or decoding process, which might change codon sequences, we might make use of the codon-code.

Solution to Problem 1, part f.

Consider all cases of 11 or fewer nucleotides per codon, and calculate for each case the "information compression" which we will define here one minus the ratio between the number of bits per amino acid for the new form of DNA and the number of bits/amino acid in Nature's original code, namely 6.

---

nucleotides per codon	bits per codon	sequences	amino acids per codon	bits per amino acid	Compression
3	6	64	1	6	0%
4	8	256	1	8	-33% (expansion)
5	10	1024	2	5	16%
6	12	4096	2	6	0%
7	14	16,384	3	4.66	22.33%
8	16	65,536	3	5.33	11.16 %
9	18	262,144	4	4.5	25%
10	20	1,048,576	4	5	16.66%
11	22	4,194,304	5	4.4	26.66%

Table 2–1: Information Compression of Codon Codes

Solution to Problem 2: Pay per Bit

Solution to Problem 2, part a.

There are 96 characters we need to encode: lower case (26), upper case (26), digits (10), space (1), punctuation marks (32) and ETX (1). The Caltech code uses 28 out of a possible 32 5-bit sequences, and uses all of the 9-bit sequences, of which there are 64 (only 6 of the 9 bits are used, giving $2^6 = 64$ codes). Thus the Caltech code has only 92 spots and is not large enough for all of our characters. The MIT code, on the other hand, uses all 32 of its 6-bit sequences, and has 64 spots left in the 7-bit sequences, giving a total of 96 spots in the code, just enough to fit all of the characters desired. Thus the MIT code works and the Caltech code does not, as one would naturally expect.

Solution to Problem 2, part b.

The Caltech graduate's code does not work. One possible redefinition of it is as follows: There are four unused 5-bit codes 10100, 10101, 10110, and 10111. Suppose we reclaim the next lowest 5-bit code, namely 10011. Now we have 27 5-bit codes to assign. We will assign these to the 26 lower case letters and space (the most commonly occurring characters). Then we will add four bits to the unused 5-bit codes (call these headers), giving us another 80 unique codes (4-bits per header gives 16 unique codes per header, times five headers equals 80 unique codes). This gives us a total of 107 unique codes (27 5-bit and 80 9-bit) which is enough to fit all of the symbols we need, plus extras if we wish. Note that this code has become more complicated to decode, since you have to keep watch for five header bits instead of one.

Solution to Problem 2, part c.

The following table compares the MIT code, ASCII, the 8-bit computer code, and the modified Caltech code presented in previous subsolution . Remember that ASCII code uses 7 bits per character no matter what, and 8-bit computer code is just that, 8-bits per character.

Solution to Problem 2, part d.

To modify the MIT graduate's code to accommodate 12 extra characters, one possible solution is as follows: The 12 additional characters are relatively rare, and so we can afford to use a longer sequence of 8 bits for them. Additionally, to distinguish them from the other sequences in our code, we could specify that they all start with 111. However, this will invalidate 16 of the 7-bit sequences (namely, all of those which start with 111). So we have newly available all the 8-bit codes that start with 111, or 32 codes, minus the 7-bit codes that start with 111, so we have a total of 16 new 8-bit codes. The extra 12 characters can be distributed among these codes.

		# of bits required			
Char. type	# of Chars	MIT Code	ASCII	8-bit	Modified Caltech Code
lower case	300	1800	2100	2400	1500
space	50	300	350	400	250
upper case	30	210	210	240	270
common punc.	10	60	70	80	90
uncommon punc.	20	140	140	160	180
ETX	1	6	7	8	9
Total	411	2516	2877	3288	2299
Average Bits/Char.	–	6.12	7	8	5.59

Table 2–2: Comparison of character codes for Problem 0, part c

Solution to Problem 2, part e.

The following two MATLAB programs implement an encoder and decoder for a select few letters of the MIT graduate's code. The first program is 'encode', and it takes a string of alphanumeric characters and outputs a sequence of 1's and 0's ordered according to the MIT Code into the variable `encodedmessage`. If it encounters a character it does not recognize, it inserts a space.

```
% Start of file, filename encode.m
% MIT Code Encoder
%

%%% Get string to encode
message = input('Input a string you would like encoded
(please use only the first seven letters, space, period,
and semicolon):','s');

messagelength=length(message);

encodedmessage=[];

%%% Encode each letter and append it to encodedmessage for i=1:messagelength
  switch message(i)
    case {'a'}
      newcharacter=0;
      newcharacter=de2bi(newcharacter,6, 'left-msb');
    case {'b'}
      newcharacter=1;
      newcharacter=de2bi(newcharacter,6, 'left-msb');
    case {'c'}
      newcharacter=2;
      newcharacter=de2bi(newcharacter,6, 'left-msb');
    case {'d'}
      newcharacter=3;
      newcharacter=de2bi(newcharacter,6, 'left-msb');
    case {'e'}
      newcharacter=4;
      newcharacter=de2bi(newcharacter,6, 'left-msb');
    case {'f'}
      newcharacter=5;
```

```
      newcharacter=de2bi(newcharacter,6, 'left-msb');
    case {'g'}
      newcharacter=6;
      newcharacter=de2bi(newcharacter,6, 'left-msb');
    case {' '}
      newcharacter=27;
      newcharacter=de2bi(newcharacter,6, 'left-msb');
    case {'.'}
      newcharacter=28;
      newcharacter=de2bi(newcharacter,6, 'left-msb');
    case {';'}
      newcharacter=29;
      newcharacter=de2bi(newcharacter,6, 'left-msb');
    case {'A'}
      newcharacter=65;
      newcharacter=de2bi(newcharacter,7, 'left-msb');
    case {'B'}
      newcharacter=66;
      newcharacter=de2bi(newcharacter,7, 'left-msb');
    case {'C'}
      newcharacter=67;
      newcharacter=de2bi(newcharacter,7, 'left-msb');
    case {'D'}
      newcharacter=68;
      newcharacter=de2bi(newcharacter,7, 'left-msb');
    case {'E'}
      newcharacter=69;
      newcharacter=de2bi(newcharacter,7, 'left-msb');
    case {'F'}
      newcharacter=70;
      newcharacter=de2bi(newcharacter,7, 'left-msb');
    case {'G'}
      newcharacter=71;
      newcharacter=de2bi(newcharacter,7, 'left-msb');
    otherwise
      newcharacter=27;
      newcharacter=de2bi(newcharacter,6, 'left-msb');
    end
  encodedmessage=cat(2,encodedmessage,newcharacter);
end

%%% Add the ETX character encodedmessage=cat(2,encodedmessage, de2bi(31,6,'left-msb'))

%%% End of file
```

The following is the decoder implementation. It takes a vector of 1's and 0's in the `encodedmessage` variable and outputs the alphanumeric translation of that string, according to the MIT code.

```
% Start of file, filename decode.m
% MIT Code Decoder
%
```

```
%%% Initialize variables
decodedcharacter=[];
decodedmessage=[];
codelength=0;
i=1;

messagelength=length(encodedmessage);

while i < messagelength,
  %%% Determine if it is a 6-bit or 7-bit code

  if encodedmessage(i) == 0
    codelength=6;
  else
    codelength=7;
  end

  %%% Read out the bits associated with the code
  decodedcharacter=[];
  for k=1:codelength
  decodedcharacter=cat(2,decodedcharacter,encodedmessage(i+k-1));
  end

  %%% Decode the character
  decodedcharacter=bi2de(decodedcharacter, 'left-msb');

  switch decodedcharacter
    case 0
      decodedcharacter='a';
    case 1
      decodedcharacter='b';
    case 2
      decodedcharacter='c';
    case 3
      decodedcharacter='d';
    case 4
      decodedcharacter='e';
    case 5
      decodedcharacter='f';
    case 6
      decodedcharacter='g';
    case 27
      decodedcharacter=' ';
    case 28
      decodedcharacter='.';
    case 29
      decodedcharacter=';';
    case 65
      decodedcharacter='A';
    case 66
      decodedcharacter='B';
```

```
    case 67
      decodedcharacter='C';
    case 68
      decodedcharacter='D';
    case 69
      decodedcharacter='E';
    case 70
      decodedcharacter='F';
    case 71
      decodedcharacter='G';
    case(31)
      decodedcharacter='(ETX)';
    otherwise
      decodedcharacter=' ';
  end
  decodedmessage=cat(2,decodedmessage,decodedcharacter);

  %%% Increment the position counter for the encoded message
  i=i+codelength;

end

%%% Output decoded message
fprintf(decodedmessage)
fprintf('\n')

%%% End of file
```