# Project Description and Objectives

Taha can you also write anything here and in conclusion, if you can’t it’s ok I left them till the end cause they’re the easiest. Also The parts I wrote some still need to be fixed and I need to write a description for implementation. I will do that tomorrow, I just wanted to get done with annoying figures and flowcharts today and description should be done quickly. So don’t mind the parts below for now. Just fill the missing other parts if u can.

# Universal Lempel-Ziv Encoder Version 1

## Introduction

The first implementation of the Lempel-Ziv encoder is the fixed window (w), fixed matches length (n), encoder. The encoder looks at each n bits and compares them to the matches in the fixed window, if encoder finds a match then it sets the FLAG bit to 1 followed by the pointer to the match in widow. This implementation of the encoder looks at matches in the window at each n bits; X0, Xn, X2n, …,Xw-n; making the size of pointer in encoding equal to log2(w/n). If the encoder doesn’t find a match, then it sets the FLAG bit to 0, and copies the not encoded n bits in encoded string.

An example of the implementation of this version of encoder is shown in **Figure X.**  This example has n as 3, with window size w equal to 12. The size of the pointer values is shown in **equation (x)**:

log2(w/n) = log2(12/3) = log2(4) = 2 bits **(x)**

The window size for i.i.d source in this example with probability of zero,P0 = 0.5, and probability of one, P1= 0.5, can be calculated using **equation (x)**, with H(X) of this binary source given in **equation (x)**:

H(X) = P0log2() + P1log2() = 0.5log2() + 0.5log2() = 1 **(x)**

w = n22nH(X) = (3)223x1 = 72 **(x)**

The example doesn’t use window size of 72 for illustrative reasons. It also important to note this window size w and length of match n are not optimal and, as shown in example, will not necessarily yield in compression. Simulations run on varying parameters of w, and n in **Section X** will show the optimal values that will yield in best compression ratio.

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

Pointer values: 0 1 2 3

FLAG: 1 1 0 0

Pointer value: 2 3 - -

Encoded string: 110 111 0101 0110

Figure ‎1.1: Illustrative example of LZ encoder version 1 with n=3 and w=12

The encoded string of this example from version 1 of encoder is 11011101010110. The expected length, E(L), of this compression can be calculated using **equation (x)**, where Pmatch is the probability of finding a match in the window (i.e. probability FLAG bit is 1) and Pno-match is the probability that no match is found in the window (i.e. probability FLAG bit is 0):

E(L) = Pmatch × (+ Pno-match × (= 0.5 × (+ 0.5 ×(

= 1.695 bits/pixel

## Implementation

All versions of the Lempel-Ziv encoder were implemented using MATLAB. The flowchart in **Figure X** shows the implementation logic of first version of LZ encoder.



# Universal Lempel-Ziv Encoder Version 2

## Introduction

The second implementation of the Lempel-Ziv encoder is the fixed window (w), fixed matches length (n), encoder. The encoder looks at each n bits and compares them to the matches in the fixed window, if encoder finds a match then it sets the FLAG bit to 1 followed by the pointer to the match in widow. This implementation of the encoder looks at matches in the window at each bit; X0, X1, …, Xn, Xn+1, Xn+2 …, X2n, …,Xw-n, Xw-(n+1), Xw–(n+2), …, Xw; making the size of pointer in encoding equal to log2(w). If the encoder doesn’t find a match, then it sets the FLAG bit to 0, and copies the not encoded n bits in encoded string.

An example of the implementation of this version of encoder is shown in **Figure X.**  This example has n as 3, with window size w equal to 12. The size of the pointer values is shown in **equation (x)**:

log2(w) = log2(12) = 3.584962 = 4 bits **(x)**

The window size for i.i.d source in this example with probability of zero,P0 = 0.5, and probability of one, P1= 0.5, can be calculated using **equation (x)**, with H(X) of this binary source given in **equation (x)**:

H(X) = P0log2() + P1log2() = 0.5log2() + 0.5log2() = 1 **(x)**

w = n22nH(X) = (3)223x1 = 72 **(x)**

The example doesn’t use window size of 72 for illustrative reasons. It also important to note this window size w and length of match n are not optimal and as shown in example will not necessarily yield in compression. Simulations run on varying parameters of w, and n in **Section X** will show the optimal values that will yield in best compression ratio.

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

Pointer values: 0 1 2 3 4 5 6 7 8 9 10 11

FLAG: 1 1 1 0

Pointer value: 4 9 1 -

Encoded string: 10100 11001 10001 0110

Figure ‎2.2: Illustrative example of LZ encoder version 3 with n= 3 and w=12

The encoded string of this example from version 1 of encoder is 1010011001100010110. The expected length, E(L), of this compression can be calculated using **equation (x)**, where Pmatch is the probability of finding a match in the window (i.e. probability FLAG bit is 1) and Pno-match is the probability that no match is found in the window (i.e. probability FLAG bit is 0):

E(L) = Pmatch × (+ Pno-match × (= 0.5 × (+ 0.5 ×(

= 1.9591 bits/pixel

## Implementation

All versions of the Lempel-Ziv encoder were implemented using MATLAB. The flowchart in **Figure X** shows the implementation logic of second version of LZ encoder.



# Universal Lempel-Ziv Encoder Version 3

## Introduction

The third implementation of the Lempel-Ziv encoder is the sliding window (w), fixed matches length (n), encoder. The encoder looks at each n bits and compares them to the matches in the sliding window, if encoder finds a match then it sets the FLAG bit to 1 followed by the pointer to the match in widow. This implementation of the encoder looks at matches in the window at each bit; X0, X1, …, Xn, Xn+1, Xn+2 …, X2n, …,Xw-n, Xw-(n+1), Xw–(n+2), …, Xw; making the size of pointer in encoding equal to log2(w). If the encoder doesn’t find a match, then it sets the FLAG bit to 0, and copies the not encoded n bits in encoded string.

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

Pointer values: 0 1 2 3 4 5 6 7 8 9 10 11

FLAG: 1 1 1 0

Pointer value: 4 9 1 -

Encoded string: 10100 11001 10001 0110

Figure ‎2.3: Illustrative example of LZ encoder version 3 with n=3 and w=12

An example of the implementation of this version of encoder is shown in **Figure X.**  This example has n as 3, with window size w equal to 12. The size of the pointer values is shown in **equation (x)**:

log2(w) = log2(12) = 3.584962 = 4 bits **(x)**

The encoded string of this example from version 1 of encoder is 1010011001100010110. The expected length, E(L), of this compression can be calculated using **equation (x)**, where Pmatch is the probability of finding a match in the window (i.e. probability FLAG bit is 1) and Pno-match is the probability that no match is found in the window (i.e. probability FLAG bit is 0):

E(L) = Pmatch ×(+ Pno-match ×(= 0.5 × (+ 0.5 ×(

= 1.9591 bits/pixel

## Implementation

All versions of the Lempel-Ziv encoder were implemented using MATLAB. The flowchart in **Figure X** shows the implementation logic of third version of sliding window LZ encoder.



# Results and Testing

In order to evaluate the accuracy of the developed encoders and decoders a basic strategy was followed to ensure their accuracy. After the development process, some basic tests were ran through each encoder and decoder to ensure they execute as expected. For these tests, small sized and simple input files were passed through the algorithms and the output from encoder/decoder was also computed by hand to ensure conformity. The algorithms were also stepped through in debug mode during this process to check if each step of the function was properly executed. After this step, some random samples of large file (10^4) were passed through the algorithm and the output from the decoder was compared with the initial input to ensure they are both the same.

Simulations

# Conclusion

# Appendix