School of Engineering and Applied Science (SEAS)
Ahmedabad University

ECE500: Information Coding Theory

Project Report
Group Number: 01

LZW Compression with Hamming Error Control

Project Mentor: Prof. Ashok Ranade

Group Members:

1. Akash Tike (1741001)
2. Kausha Vora (1741016)
3. Parth Maniyar (1741068)

1. **Synopsis:**

   - **Project Title:** LZW Compression with Hamming Error Control Code

   - **Brief description:**
     We simulated Lempel–Ziv–Welch compression algorithm along with its decompression. Moreover, for error correcting we implemented Hamming code.

     (a) **Lempel–Ziv–Welch Compression:**
     (b) **Hamming Error Correcting Code:**

   - **Programming Language:** Python

   - **User Interface & Result Format:**
     We integrated a graphical interface, **PyQt** to our Python Code.
     Following data/metrics is shown on the GUI after a text file is transmitted:

     (a) **Compressed File Size**
     (b) **Compression Ratio**
     (c) **Decompressed File Size**
     (d) **Compression Speed**
     (e) **Decompression Speed**
     (f) **Compression Time**
     (g) **Decompression Time**
     (h) **Transmission Time without compression**
     (i) **Transmission Time with compression**

     Following data/metrics(for each Code-word) is shown on the terminal screen during the simulation:

     (a) **Information Bits**

(b) **Transmitted Bits**
(c) **Received Bits**
(d) **Detected Error Bits**
(e) **Corrected Error Bits**

2. **Techniques Used:**

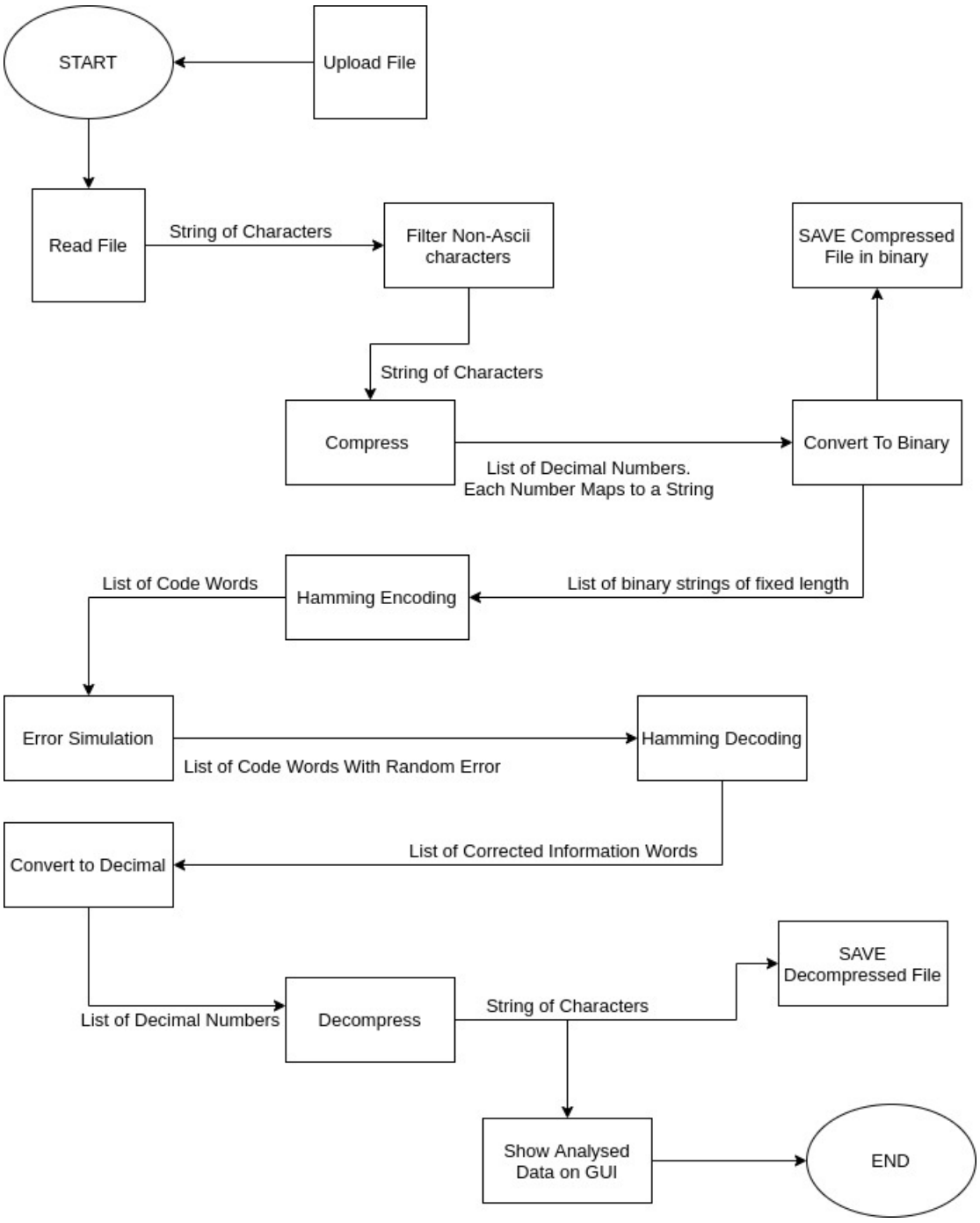
(a) **Lempel–Ziv–Welch Compression:**

LZW is a lossless algorithm, meaning no data is lost while compressing. The idea of LZW is based on the repeating patterns which optimizes the data space. This algorithm is commonly used in Unix file compression and also in the popular GIF image format.

(b) **Hamming Error Correcting Code:**

Hamming code is useful for error detection up to two-bit errors. However, it is also capable of correcting single-bit errors. In Hamming Code, we use extra parity bits to identify the error on the receiving side. Hamming Codes are commonly used in Modems, Embedded Processor, etc. The hamming bound formula used in the project is $2^r \geq m + r + 1$ where r is parity bits and m is information bits and number of errors to be corrected is 1. We have used this formula to calculate number of parity bits while knowing the number information bits.

3. **Flow Chart:**


START
Upload File
Read File
String of Characters
Filter Non-Ascii characters
SAVE Compressed File in binary
String of Characters
Compress
List of Decimal Numbers.
Each Number Maps to a String
Convert To Binary
List of Code Words
Hamming Encoding
List of binary strings of fixed length
Error Simulation
List of Code Words With Random Error
Hamming Decoding
Convert to Decimal
List of Corrected Information Words
List of Decimal Numbers
Decompress
String of Characters
SAVE Decompressed File
Show Analysed Data on GUI
END

4. **User interface:**

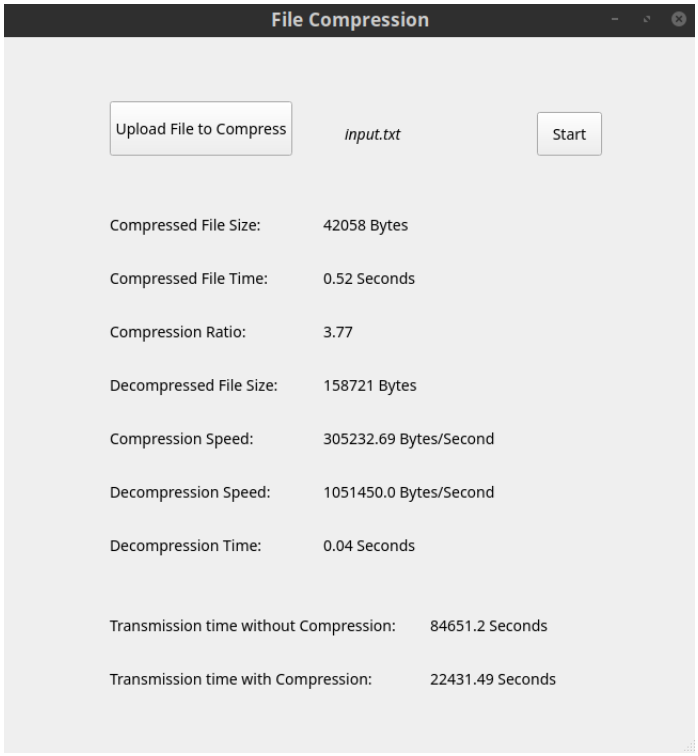


Figure 1: Dialog after the user selects any file(*input.txt* in this case) to upload and hits the start button.

For simulation purposes, we are transmitting one codeword per second. In the above case, it is 20 Bits/Seconds.

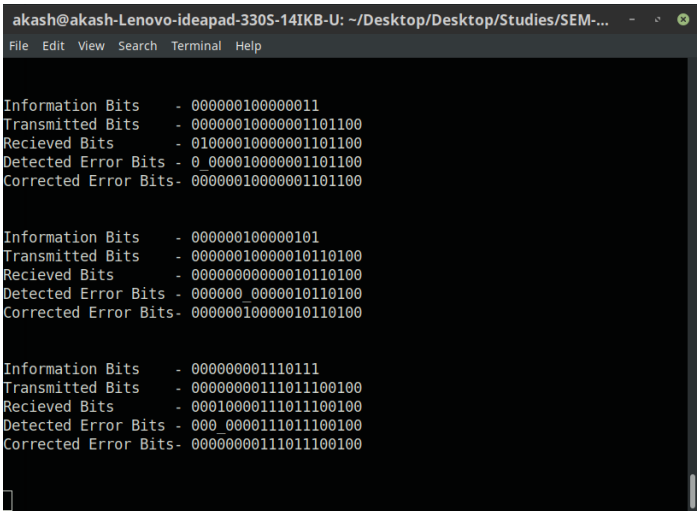


Figure 2: Figure shows 5 metrics for each transmitted codeword.

The Metrics include Information Bits, Transmitted Bits, Received Bits, Detected Error Bits and Corrected Error Bits. The "_" in Detected Error Bits shows the position of error.

5. **Metric Description in the Analysis:**

   (a) **Compressed File Size:** Size of the Compressed File.

   (b) **Compressed File Time:** Time taken to Compress the File.

   (c) **Compression Ratio:**

$$\frac{UncompressedFileSize}{CompressedFileSize}$$

   (d) **Decompressed File Size:** Size of the Decompressed File.

   (e) **Compression Speed:**

$$\frac{UncompressedFileSize}{TimetakentoCompresstheFile}$$

   (f) **Decompressed File Time:** Time taken to Decompress the File.

   (g) **Decompression Speed:**

$$\frac{CompressedFileSize}{TimetakentoDecompresstheFile}$$

   (h) **Transmission Time without Compression:**

$$\frac{UncompressedFileSize}{InformationBitRate}$$

   Taking Transmission speed, Codeword/Second for Simulation

   (i) **Transmission Time with Compression:**

$$\frac{CompressedFileSize}{InformationBitRate} + CompressionTime + DecompressionTime$$

   Taking Transmission speed, Codeword/Second for Simulation

6. **Python Program:**

```
import math
import time
import _thread
import random

from pathlib import Path
from bitstring import BitArray
from timeit import default_timer as timer


max_length = 0
max_length_log = 0

def calcRedundantBits(m):

  for i in range(m):
    if(2**i >= m + i + 1):
      return i


def generateCodeWord(arr, r):

  n = len(arr)
  parity = ""
  for i in range(r):
    val = 0
    for j in range(1, n + 1):
      if(j & (2**i) == (2**i)):
        val = val ^ int(arr[j-1])

    parity += str(val)

  return arr + parity


def isascii(s):
  """Check if the characters in string s are in ASCII, U+0-U+7F."""
  return len(s) == len(s.encode())

def SpecialCharacterFilter(string):

  FilteredList = ""

  for i in string:

    if(isascii(i) == True):
      FilteredList += i

  return "".join(FilteredList)

rand_list = []
rand_sol_list = []


def correctedInformationWord(arr, nr):

  n = len(arr)-nr
  res = 0

  for i in range(nr):
    val = int(arr[i+n])
    for j in range(1, n + 1):
      if(j & (2**i) == (2**i)):
        val = val ^ int(arr[j-1])
    res = res + val*pow(2,i)

  arr_list = list(arr)
```

```
  detected_error = arr_list.copy()
  if(res > 0):
    arr_list[res-1] = str(int(arr_list[res-1])^1)
    detected_error[res-1] = "_"

  rand_sol_list.append(res-1)

  detected_error = "".join(detected_error)
  corrected = "".join(arr_list.copy())
  arr_list = arr_list[:-nr]
  return ["".join(arr_list), detected_error, corrected]


def compress(uncompressed):
  """Compress a string to a list of output symbols."""

  global max_length, max_length_log

  # Build the dictionary.
  dict_size = 256
  dictionary = dict((chr(i), i) for i in range(dict_size))
  # in Python 3: dictionary = {chr(i): i for i in range(dict_size)}

  w = ""
  result = []
  for c in uncompressed:
    wc = w + c
    if wc in dictionary:
      w = wc
    else:
      result.append(dictionary[w])
      max_length = max(max_length,dictionary[w])
      # Add wc to the dictionary.
      dictionary[wc] = dict_size
      dict_size += 1
      w = c

  # Output the code for w.
  if w:
    result.append(dictionary[w])
    max_length = max(max_length,dictionary[w])

  max_length_log = int(math.log(max_length,2))
  if(max_length_log*max_length_log != max_length):
    max_length_log += 1

  return result


def decompress(compressed):
  """Decompress a list of output ks to a string."""
  from io import StringIO

  # Build the dictionary.
  dict_size = 256
  dictionary = dict((i, chr(i)) for i in range(dict_size))
  # in Python 3: dictionary = {i: chr(i) for i in range(dict_size)}

  # use StringIO, otherwise this becomes O(N^2)
  # due to string concatenation in a loop
  result = StringIO()
  w = chr(compressed.pop(0))
  result.write(w)
  for k in compressed:
    if k in dictionary:
      entry = dictionary[k]
    elif k == dict_size:
      entry = w + w[0]
    else:
      raise ValueError('Bad compressed k: %s' % k)
```

```
    result.write(entry)

    # Add w+entry[0] to the dictionary.
    dictionary[dict_size] = w + entry[0]
    dict_size += 1

    w = entry

  return result.getvalue()


def convertToBinary(compressed,flag=True):

  result = []

  for num in compressed:
    x = bin(num).replace("0b", "")
    if(flag == True):
      x = "0"*(max_length_log - len(x)) + x
    else:
      x = "0"*(8-len(x)) + x
    result.append(x)

  return result


def convertToDecimal(corrected,flag=True):

  if(flag == True):
    return int(corrected, 2)
  else:
    arr = []
    for data in corrected:
      arr.append(int(data,2))
    return arr


def SaveCompressedFile(s):

  v = int(s, 2)
  b = bytearray()
  while v:
    b.append(v & 0xff)
    v >>= 8

  f = open('../Reciever/compress.txt', 'wb')
  f.write(bytes(b[::-1]))
  f.close()


def DeCompressFile():

  f = open('../Reciever/compress.txt', 'rb')
  arr = []
  for byte in f.read():
    arr.append(byte)

  arr = convertToBinary(arr,False)
  arr = "".join(arr)

  arr1 = []
  for i in range(len(arr)-1,0,-max_length_log):
    arr1.append(str(arr[i-max_length_log+1:i+1]))

  arr1.reverse()
  arr = []
  for data in arr1:
    if(data == ""):
      continue
    arr.append(convertToDecimal(data))
```

```
208
209
210
211 def SimulateError(HammingEncodedList):
212
213   arr_list = []
214
215   for i in range(len(HammingEncodedList)):
216
217     temp = list(HammingEncodedList[i])
218     randomNum = random.randint(0, max_length_log-1)
219     if(temp[randomNum] == '0'):
220       temp[randomNum] = '1'
221     else:
222       temp[randomNum] = '0'
223     rand_list.append(randomNum)
224
225     arr_list.append("".join(temp))
226
227   return arr_list
228
229 def HammingEncoding(compressed):
230
231   HammingEncodedList = []
232
233   m = max_length_log # information bits (k)
234   r = calcRedundantBits(m)
235
236   for data in compressed:
237
238     arr = generateCodeWord(data,r)
239     HammingEncodedList.append(arr)
240
241   return HammingEncodedList
242
243
244 def HammingDecoding(HammingEncodedList):
245
246   HammingDecodedList = []
247   DecodedErrorList = []
248   CorrectedErrorList = []
249
250   m = max_length_log
251   r = calcRedundantBits(m)
252
253   for data in HammingEncodedList:
254
255     [corrected,detected_error,corrected_error_with_parity] = correctedInformationWord(
      data, r)
256     HammingDecodedList.append(corrected)
257     DecodedErrorList.append(detected_error)
258     CorrectedErrorList.append(corrected_error_with_parity)
259
260   return [HammingDecodedList,DecodedErrorList,CorrectedErrorList]
261
262
263 def SaveDecompressedFile(DecompressedString):
264
265   f = open("../Reciever/Decompressed.txt","w")
266
267   f.write(DecompressedString)
268
269   f.close()
270
271
272 start = end = start_decompress = end_decompress = 0
273
274 def Start(fname):
275
276   global start,end,start_decompress,end_decompress
```

```
277
278   start = timer()
279
280   f = open(fname,"r")
281
282   FilteredString =  SpecialCharacterFilter(str(f.read()))
283
284   compressed = compress(FilteredString)
285
286   binary_compressed = convertToBinary(compressed)
287
288   compressed_str = "".join(map(str, binary_compressed))
289
290   SaveCompressedFile(compressed_str)
291
292   end = timer()
293
294   HammingEncodedList = HammingEncoding(binary_compressed)
295   # 0th element transmitted
296
297   RecievedList = SimulateError(HammingEncodedList)
298   # RecievedList = HammingEncodedList
299
300   [HammingDecodedList,DecodedErrorList,CorrectedErrorList] = HammingDecoding(
        RecievedList)
301   # 0th element recieved
302
303   start_decompress = timer()
304
305   DecimalList = convertToDecimal(HammingDecodedList,False)
306
307   DecompressedString = decompress(DecimalList)
308
309   SaveDecompressedFile(DecompressedString)
310
311   end_decompress = timer()
312
313   print(DecompressedString)
314
315   return [binary_compressed,HammingEncodedList,RecievedList,DecodedErrorList,
        CorrectedErrorList]
316
317 def test(self,InformationBits,TransmittedBits,RecievedBits,DetectedErrorBits,
        CorrectedErrorBits,var):
318
319   print("Information Bits per Second = ",str(int(len(InformationBits[0])/var)),"Bits/
        Sec")
320   print("Transmitted Bits per Second = ",str(int(len(TransmittedBits[0])/var)),"Bits/
        Sec",end="\n\n")
321
322   for i in range(len(InformationBits)):
323
324     print("Information Bits    -", str(InformationBits[i]))
325     print("Transmitted Bits    -" ,str(TransmittedBits[i]))
326     print("Recieved Bits       -" ,str(RecievedBits[i]))
327     print("Detected Error Bits -" ,str(DetectedErrorBits[i]))
328     print("Corrected Error Bits-" ,str(CorrectedErrorBits[i]))
329     print("\n")
330
331     time.sleep(var)
332
333 # LZW Encoding
334 # Convert to Binary
335   # Information Bits
336 # Hamming encoding
337   # Transmitted Bits
338   # Update Percentage of file transferred
339 # Simulate Random error
340   # Recieved Bits
341 # Hamming decoding
```

```
  # Error Bits or Position
  # Corrected Bits
# Convert to ASCII
# LZW Decoding

from PyQt5 import QtCore, QtGui, QtWidgets
from PyQt5.QtWidgets import *
import sys
from pyqtgraph import PlotWidget, plot
import pyqtgraph as pg

g1 = []
b1 = []
g2 = []
b2 = []

class Second(QtGui.QMainWindow):
  def __init__(self,parent=None):

    super(Second, self).__init__(parent)
    self.graphWidget = pg.PlotWidget()
    self.setCentralWidget(self.graphWidget)

    pen = pg.mkPen(color=(255, 255, 255))
    self.graphWidget.plot(b1, g1,pen=pen)
    pen = pg.mkPen(color=(255, 255, 0))
    self.graphWidget.plot(b2, g2,pen=pen)

class MainWindow(QtWidgets.QMainWindow):

  def __init__(self, *args, **kwargs):

    super(MainWindow, self).__init__(*args, **kwargs)
    window = Ui_mainWindow()
    window.setupUi(self)


class Ui_mainWindow(object):

  def setupUi(self, mainWindow):
    mainWindow.setObjectName("mainWindow")
    mainWindow.resize(650, 670)

    self.fname = ""
    self.centralwidget = QtWidgets.QWidget(mainWindow)
    self.centralwidget.setObjectName("centralwidget")
    self.pushButton = QtWidgets.QPushButton(self.centralwidget)
    self.pushButton.setGeometry(QtCore.QRect(100, 60, 171, 51))
    self.pushButton.setObjectName("pushButton")
    self.pushButton.clicked.connect(lambda:self.getfile())
    self.label = QtWidgets.QLabel(self.centralwidget)
    self.label.setGeometry(QtCore.QRect(320, 70, 181, 41))
    font = QtGui.QFont()
    font.setItalic(True)
    self.label.setFont(font)
    self.label.setObjectName("label")
    self.pushButton_2 = QtWidgets.QPushButton(self.centralwidget)
    self.pushButton_2.setGeometry(QtCore.QRect(500, 70, 61, 41))
    self.pushButton_2.setObjectName("pushButton_2")
    self.pushButton_2.clicked.connect(lambda:self.start(mainWindow))

    self.label_2 = QtWidgets.QLabel(self.centralwidget)
    self.label_2.setGeometry(QtCore.QRect(100, 150, 161, 51))
    self.label_2.setObjectName("label_2")
    self.label_3 = QtWidgets.QLabel(self.centralwidget)
    self.label_3.setGeometry(QtCore.QRect(100, 200, 161, 51))
    self.label_3.setObjectName("label_3")
    self.label_4 = QtWidgets.QLabel(self.centralwidget)
    self.label_4.setGeometry(QtCore.QRect(100, 300, 161, 51))
    self.label_4.setObjectName("label_4")
```

```
    self.label_5 = QtWidgets.QLabel(self.centralwidget)
    self.label_5.setGeometry(QtCore.QRect(100, 250, 161, 51))
    self.label_5.setObjectName("label_5")
    self.label_6 = QtWidgets.QLabel(self.centralwidget)
    self.label_6.setGeometry(QtCore.QRect(100, 350, 161, 51))
    self.label_6.setObjectName("label_6")
    self.label_7 = QtWidgets.QLabel(self.centralwidget)
    self.label_7.setGeometry(QtCore.QRect(100, 400, 161, 51))
    self.label_7.setObjectName("label_7")
    self.label_8 = QtWidgets.QLabel(self.centralwidget)
    self.label_8.setGeometry(QtCore.QRect(100, 450, 161, 51))
    self.label_8.setObjectName("label_8")

    font = QtGui.QFont()
    font.setPointSize(12)
    self.label_11 = QtWidgets.QLabel(self.centralwidget)
    self.label_11.setGeometry(QtCore.QRect(100, 530, 281, 41))
    self.label_11.setObjectName("label_11")
    self.label_12 = QtWidgets.QLabel(self.centralwidget)
    self.label_12.setGeometry(QtCore.QRect(100, 580, 281, 41))
    self.label_12.setObjectName("label_12")


    self.label_2_val = QtWidgets.QLabel(self.centralwidget)
    self.label_2_val.setGeometry(QtCore.QRect(300, 150, 261, 51))
    self.label_2_val.setObjectName("label_2_val")
    self.label_3_val = QtWidgets.QLabel(self.centralwidget)
    self.label_3_val.setGeometry(QtCore.QRect(300, 200, 261, 51))
    self.label_3_val.setObjectName("label_3_val")
    self.label_4_val = QtWidgets.QLabel(self.centralwidget)
    self.label_4_val.setGeometry(QtCore.QRect(300, 300, 261, 51))
    self.label_4_val.setObjectName("label_4_val")
    self.label_5_val = QtWidgets.QLabel(self.centralwidget)
    self.label_5_val.setGeometry(QtCore.QRect(300, 250, 261, 51))
    self.label_5_val.setObjectName("label_5_val")
    self.label_6_val = QtWidgets.QLabel(self.centralwidget)
    self.label_6_val.setGeometry(QtCore.QRect(300, 350, 261, 51))
    self.label_6_val.setObjectName("label_6_val")
    self.label_7_val = QtWidgets.QLabel(self.centralwidget)
    self.label_7_val.setGeometry(QtCore.QRect(300, 400, 261, 51))
    self.label_7_val.setObjectName("label_7_val")
    self.label_8_val = QtWidgets.QLabel(self.centralwidget)
    self.label_8_val.setGeometry(QtCore.QRect(300, 450, 261, 51))
    self.label_8_val.setObjectName("label_8_val")
    font = QtGui.QFont()
    font.setPointSize(12)
    self.label_11_val = QtWidgets.QLabel(self.centralwidget)
    self.label_11_val.setGeometry(QtCore.QRect(400, 530, 300, 41))
    self.label_11_val.setObjectName("label_11_val")
    self.label_12_val = QtWidgets.QLabel(self.centralwidget)
    self.label_12_val.setGeometry(QtCore.QRect(400, 580, 300, 41))
    self.label_12_val.setObjectName("label_12_val")

    mainWindow.setCentralWidget(self.centralwidget)
    self.statusbar = QtWidgets.QStatusBar(mainWindow)
    self.statusbar.setObjectName("statusbar")
    mainWindow.setStatusBar(self.statusbar)
    self.dialogs = list()
    self.retranslateUi(mainWindow)
    QtCore.QMetaObject.connectSlotsByName(mainWindow)

  def retranslateUi(self, mainWindow):
    _translate = QtCore.QCoreApplication.translate
    mainWindow.setWindowTitle(_translate("mainWindow", "File Compression"))
    self.pushButton.setText(_translate("mainWindow", "Upload File to Compress"))
    self.label.setText(_translate("mainWindow", "No files selected yet"))
    self.pushButton_2.setText(_translate("mainWindow", "Start"))
    self.label_2.setText(_translate("mainWindow", "Compressed File Size:"))
    self.label_3.setText(_translate("mainWindow", "Compressed File Time:"))
    self.label_4.setText(_translate("mainWindow", "Decompressed File Size:"))
```

```
    self.label_5.setText(_translate("mainWindow", "Compression Ratio:"))
    self.label_6.setText(_translate("mainWindow", "Compression Speed:"))
    self.label_7.setText(_translate("mainWindow", "Decompression Speed:"))
    self.label_8.setText(_translate("mainWindow", "Decompression Time:"))
    self.label_11.setText(_translate("mainWindow", "Transmission time without Compression:"))
    self.label_12.setText(_translate("mainWindow", "Transmission time with Compression:"))

  def getfile(self):

    _translate = QtCore.QCoreApplication.translate
    fname = QFileDialog.getOpenFileName(None, 'Open file','.',"Text files (*.txt)")
    fname = fname[0]
    self.fname = fname
    if(fname == ""):
      self.label.setText(_translate("mainWindow", "No files selected yet"))
    else:
      fname = fname.split("/")[-1]
      self.label.setText(_translate("mainWindow", fname))

  def start(self,mainWindow):

    _translate = QtCore.QCoreApplication.translate

    [InformationBits,TransmittedBits,RecievedBits,DetectedErrorBits,CorrectedErrorBits] = Start(self.fname)

    CompressionFileSize = Path('../Reciever/compress.txt').stat().st_size

    CompressedFileTime = round(end-start,2)

    UncompressedFileSize = Path(self.fname).stat().st_size

    CompressionRatio = round(UncompressedFileSize/CompressionFileSize,2)

    DeCompressedFileSize = Path('../Reciever/Decompressed.txt').stat().st_size

    CompressionSpeed = round(UncompressedFileSize/CompressedFileTime,2)

    DeCompressedFileTime = round(end_decompress - start_decompress,2)

    DecompressionSpeed = round(CompressionFileSize/DeCompressedFileTime,2)

    var = 1

    InformationBitsPerSecond = int(len(InformationBits[0])/var)

    TransmissionTimeWithoutCompression = round((UncompressedFileSize*8)/(InformationBitsPerSecond),2)

    TransmissionTimeWithCompression = round(CompressedFileTime + DeCompressedFileTime + (CompressionFileSize*8)/(InformationBitsPerSecond),2)


    self.label_2_val.setText(_translate("mainWindow", str(CompressionFileSize) + " Bytes"))
    self.label_3_val.setText(_translate("mainWindow", str(CompressedFileTime) + " Seconds"))
    self.label_4_val.setText(_translate("mainWindow", str(DeCompressedFileSize) + " Bytes"))
    self.label_5_val.setText(_translate("mainWindow", str(CompressionRatio)))
    self.label_6_val.setText(_translate("mainWindow", str(CompressionSpeed) + " Bytes/Second"))
    self.label_7_val.setText(_translate("mainWindow", str(DecompressionSpeed) + " Bytes/Second"))
    self.label_8_val.setText(_translate("mainWindow", str(DeCompressedFileTime) + " Seconds"))
    self.label_11_val.setText(_translate("mainWindow", str(TransmissionTimeWithoutCompression) + " Seconds"))
```

```
540     self.label_12_val.setText(_translate("mainWindow", str(
        TransmissionTimeWithCompression) + " Seconds"))
541 
542     for bandwidth in range(1,100):
543 
544       TransmittionTimeWithoutCompression = round((UncompressedFileSize*8)/(bandwidth)
        ,2)
545 
546       TransmittionTimeWithCompression = round(CompressedFileTime + DeCompressedFileTime
         + (CompressionFileSize*8)/(bandwidth),2)
547 
548       g1.append(TransmittionTimeWithoutCompression)
549       b1.append(bandwidth)
550 
551       g2.append(TransmittionTimeWithCompression)
552       b2.append(bandwidth)
553 
554     self.showGraph(mainWindow)
555 
556     _thread.start_new_thread( test, (self,InformationBits,TransmittedBits,RecievedBits,
        DetectedErrorBits,CorrectedErrorBits,var) )
557 
558   def showGraph(self,mainWindow):
559 
560     dialog = Second(mainWindow)
561     self.dialogs.append(dialog)
562     dialog.show()
563 
564 def main():
565   app = QtWidgets.QApplication(sys.argv)
566   main = MainWindow()
567   main.show()
568   sys.exit(app.exec_())
569 
570 
571 if __name__ == '__main__':
572   main()
```

7. **Results:**

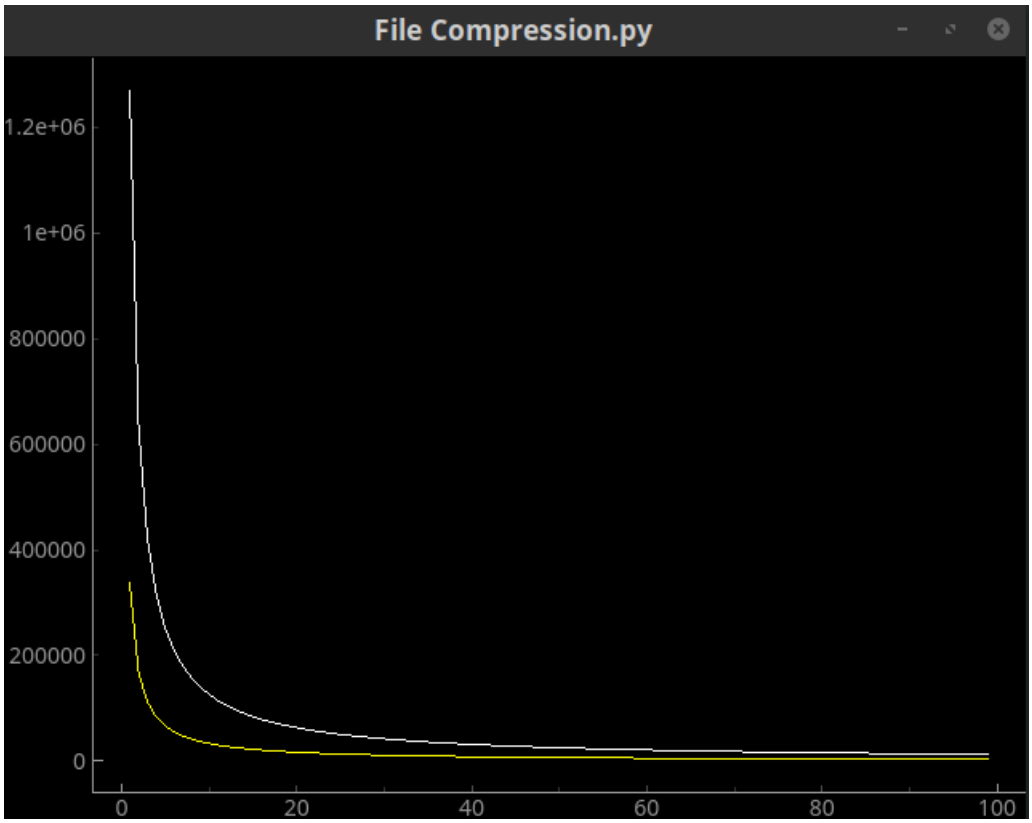


Figure 3: Information Bit Rate(Bits/Second) VS Time Taken to Transmit the File(Seconds)

- Y-axis = Transmission Time(Second)
- X-axis = Information Bit Rate(Bits/Second)
- Yellow Line: Transmission Time taken for a file with Compression
- While Line: Transmission Time taken for a file without Compression
- In the experiment we have taken the input file of 158KB. All the figures are according to the respective file.

8. **Conclusion:**

- As we can see in the result, the time taken for a file to transmit without compression is significantly greater than the time taken for a file to transmit with compression.
- However, for large information bit rate the difference between both would become negligible even though their ratio remains same.
- We experimented with a few set of files(Dummy Data) and found the Average Compression Ratio to be 3.5-4.5.
- Our Error Control Code corrects up to 1 Bit error which is evident in our simulation.