universidade de aveiro deti departamento de electrónica, telecomunicações e informática

3 de fevereiro de 2021

A03 - AED

Recursively decoding a non-instantaneous binary code

Trabalho realizado por:

Henrique Sousa

NºMec 98324, P4

Índice

Introdução

O objetivo deste trabalho é decodificar recursivamente um código binário não instantâneo. Para isto, é nos fornecido código o qual temos de completar. Este programa gera um código e uma mensagem, ambos, aleatórios e codifica a mensagem. O nosso trabalho é desenvolver a função que descodifica esta mensagem codificada. Foi nos, também, proposta a ideia, opcional, de desenvolver um programa que descodifique a mensagem em tempo real, ou seja, descodifique a mensagem à medida que os bits vão sendo considerados, analisando todas as possibilidades até ao momento.

Abordagem ao problema

O código originalmente fornecido pelos docentes carece da criação de uma função chamada “recursive_decoder”. Esta função tem como objetivo descodificar uma mensagem, anteriormente, codificada pelas restantes funções fornecidas. A descodificação deve ser realizada de forma recursiva e deve recolher algumas informações, tais como, o número de vezes que esta função foi chamada, o número de soluções e o número máximo de bits extra, ou seja, quantos bits a mais foram analisados até o programa entender que a codificação estava errada. Este problema foi selecionado da seguinte forma:

```
static void recursive_decoder(int encoded_idx,int decoded_idx,int
good_decoded_size)
{
  if(decoded_idx == _original_message_size_)
  {
    _number_of_solutions_++;
    return;
  }
  _number_of_calls_++;
  if (decoded_idx - good_decoded_size > _max_extra_symbols_)
    _max_extra_symbols_ = decoded_idx - good_decoded_size;
  for (int i = encoded_idx; i < _max_encoded_message_size_ && i-encoded_idx+1
<= _c_->max_bits; i++)
  {
    char tempMessage[i-encoded_idx+2]; int n = -1;
    while (++n <= i-encoded_idx)
      tempMessage[n] = _encoded_message_[encoded_idx+n];
    tempMessage[n] = '\0';
    for (int j = 0; j < _c_->n_symbols ; j++)
    {
      if (strcmp(_c_->data[j].codeword, tempMessage) == 0)
      {
        _decoded_message_[decoded_idx] = j;
        if (decoded_idx == good_decoded_size && _decoded_message_[decoded_idx]
== _original_message_[decoded_idx])
            recursive_decoder(i+1, decoded_idx+1, good_decoded_size+1);
        else recursive_decoder(i+1, decoded_idx+1, good_decoded_size);
        break;
      }
    }
  }
}
```

A função começa com a verificação para verificar se a variável *“decoded_idx”* é igual à variável *“_original_message_size_”* isto serve para verificar se a mensagem já foi totalmente descodificada, se sim incrementamos o número de soluções e retornamos a função de modo a terminá-la. Incrementamos, também, a variável *“_number_of_calls_”* após essa verificação para garantir que contamos o número de vezes que a função foi chamada, apenas se for necessário descodificar mais bits. A determinação do número máximo de bits extra é realizada antes de qualquer descodificação, caso “*decoded_idx – good_decoded_size”* seja menor que *“_max_extra_symbols_”* alteramos esta última variável para o resultado da subtração, anteriormente, efetuada.

```
if(decoded_idx == _original_message_size_)
{
  _number_of_solutions_++;
  return;
}
_number_of_calls_++;
if (decoded_idx - good_decoded_size > _max_extra_symbols_)
    _max_extra_symbols_ = decoded_idx – good_decoded_size;
```

Após estas verificações e definições iniciais começamos o processo de descodificação. Para isto, devemos começar por analisar a mensagem codificada, analisando bit a bit, caso não exista nenhum símbolo que corresponde ao bit que estamos a analisar, acrescentamos ao mesmo o bit seguinte da mensagem codificada, repetindo este processo até que seja encontrada uma sequência de bits que corresponda ao padrão de um símbolo. Ao encontrar o símbolo que corresponde à sequência de bits podemos atribuir este símbolo à posição *“decoded_idx”* da *“decoded_message”*. Tendo descodificado a sequência de bits devemos chamar novamente a função alterando as variáveis passadas como argumentos. *“encoded_idx”* deve ser incrementada com o número de bits da sequência descodificada mais um, *“decoded_idx”* será sempre incrementada por um em cada chamada e o *“good_decoded_idx”* permancerá o mesmo, ou será incrementado por um caso o bit descodificado corresponda ao da mensagem original, ou seja, tenha sido corretamente descodificado.

```
  for (int i = encoded_idx; i < _max_encoded_message_size_ && i-encoded_idx+1
<= _c_->max_bits; i++)
  {
    char tempMessage[i-encoded_idx+2]; int n = -1;
    while (++n <= i-encoded_idx)
      tempMessage[n] = _encoded_message_[encoded_idx+n];
    tempMessage[n] = '\0';
    for (int j = 0; j < _c_->n_symbols ; j++)
    {
      if (strcmp(_c_->data[j].codeword, tempMessage) == 0)
      {
        _decoded_message_[decoded_idx] = j;
        if (decoded_idx == good_decoded_size && _decoded_message_[decoded_idx]
== _original_message_[decoded_idx])
            recursive_decoder(i+1, decoded_idx+1, good_decoded_size+1);
        else recursive_decoder(i+1, decoded_idx+1, good_decoded_size);
        break;
      }
    }
}
```

De forma a não analisar casos impossíveis de descodificação, caso o número de bits numa sequência for superior ao numero máximo de bits em cada símbolo a função é terminada e as restantes sequências não são analisadas.

Obtenção de Resultados

Scripts

Para obter os resultado executei o script “do_all.sh” fornecido pelos docentes, onde dividi o trabalho, de modo a acelerar o processo, por dois computadores, um com doze cores e outro com quatro. Foi possível obter os resultados até 500 símbolos. Por cada número de símbolos foi criado um ficheiro onde o seu nome representa a quantidade de símbolos.

```
sousa@dev:~/Documents/Universidade/AED/A03/Resultados$ ls *[0-9]*
0003  0019  0035  0051  0067  0083  0099  0115  0131  0147  0163  0179  0195
0004  0020  0036  0052  0068  0084  0100  0116  0132  0148  0164  0180  0196
0005  0021  0037  0053  0069  0085  0101  0117  0133  0149  0165  0181  0197
0006  0022  0038  0054  0070  0086  0102  0118  0134  0150  0166  0182  0198
0007  0023  0039  0055  0071  0087  0103  0119  0135  0151  0167  0183  0199
0008  0024  0040  0056  0072  0088  0104  0120  0136  0152  0168  0184  0200
```

Em cada um destes ficheiros a informação está apresentada desta forma:

Número de símbolos	número de chamadas por símbolo				símbolos extra			
ns	Mínimo	Média	Intermédio	Máximo	Mínimo	Média	Intermédio	Máximo

```
63      5.605     5.745     5.747     5.848     63    91.5    91   146
```

De forma a simplificar a apresentação dos resultados para o apêndice deste trabalho desenvolvi o seguinte script em bash para juntar todos os resultados obtidos dos diferentes “números de símbolos” num único ficheiro:

```
#! /bin/bash

if [ -e results.txt ]; then
  rm results.txt
fi


dir="Resultados"
for file in $(ls $dir | grep [0-9]); do
    ns=$( awk '{print $1}' $dir/$file )
    cMin=$( awk '{print $2}' $dir/$file )
    cMed=$( awk '{print $3}' $dir/$file )
    cInt=$( awk '{print $4}' $dir/$file )
    cMax=$( awk '{print $5}' $dir/$file )
    sMin=$( awk '{print $6}' $dir/$file )
    sMed=$( awk '{print $7}' $dir/$file )
    sInt=$( awk '{print $8}' $dir/$file )
    sMax=$( awk '{print $9}' $dir/$file )
    printf "%s\t%s\t%s\t%s\t%s\t%s\t%s\t%s\t%s\n" "$ns" "$cMin" "$cMed"
"$cInt" "$cMax" "$sMin" "$sMed" "$sInt" "$sMax" >> results.txt
done
```

Este script junta todos os resultados obtidos para os diferentes números de símbolos e cria o ficheiro *“results.txt”* com todos os resultados obtidos.

Gráficos

Para criar gráficos com os resultados obtidos nos ficheiros anteriores recorri ao matlab e com o seguinte programa consegui criar gráficos.

```
fileName = 'files.txt';

FID = fopen(fileName);

data = textscan(FID,'%s');
fclose(FID);
stringData = string(data{:});
figure
for i = 1 : length(stringData)
    file = stringData(i);
    data = load(file);
    % numero de chamadas médio
    hold on
    plot(data(1,1), data(1,5), '^', 'color', 'red'); %calls máximas
    plot(data(1,1), data(1,3), '.', 'color', 'blue'); %calls médias
    plot(data(1,1), data(1,2), 'v', 'color', 'black'); %calls minimas
    plot(data(1,1), data(1,4), '-', 'color', 'green'); %calls intermedias
```

```
    set(gca,'yscale','log');
    xlabel("Nº Símbolos");
    ylabel("Nº de chamadas por símbolo médio")
    hold off
end
%números de bits máximo
figure
for i = 1 : length(stringData)
    file = stringData(i);
    data = load(file);
    hold on
    plot(data(1,1), data(1,9), '^', 'color', 'red'); %bits extra máximos
    plot(data(1,1), data(1,7), '.', 'color', 'blue'); %bits extra médios
    plot(data(1,1), data(1,6), 'v', 'color', 'black'); %bits extra mínimos
    plot(data(1,1), data(1,8), '-', 'color', 'green'); %bits extra Intermédios
    xlabel("Nº Símbolos");
    ylabel("Nº de símbolos extra")
    hold off
end
```

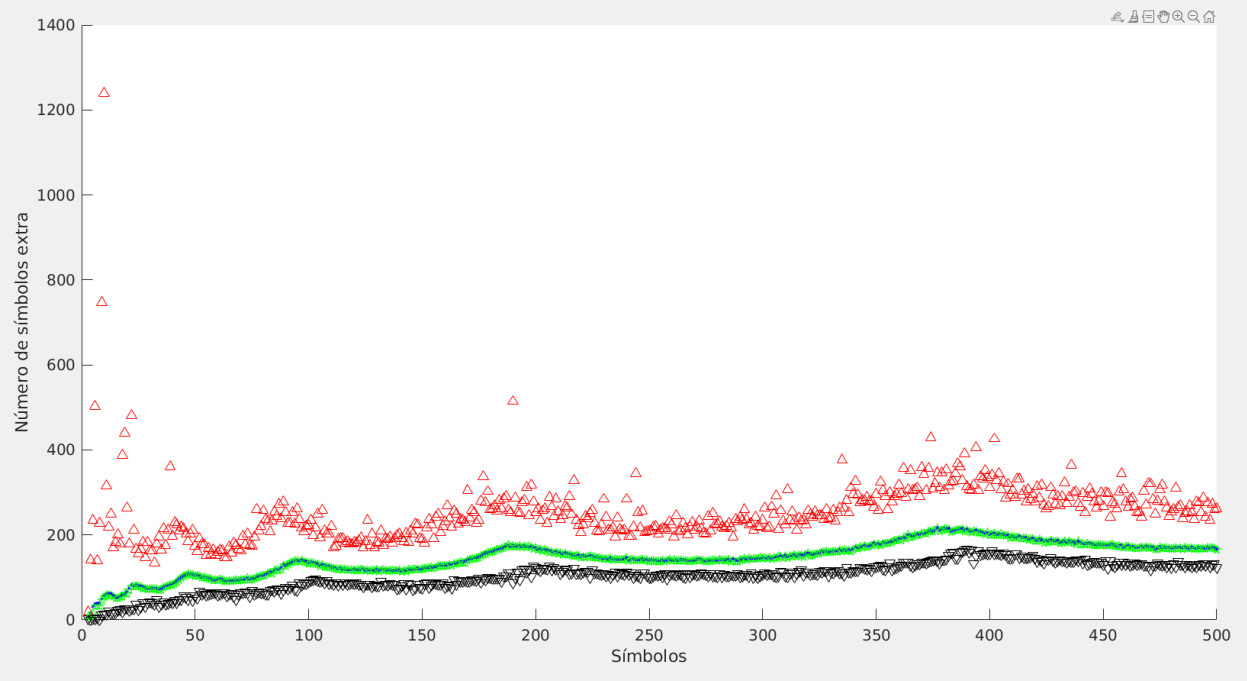
A execução deste programa em Matlab origina duas figuras com valores mínimos, média, intermédios e máximos do número de chamadas à função por símbolo e número de símbolos extra máximos até a função perceber que não está a codificar corretamente.

Resultados

Número de Bits Extra

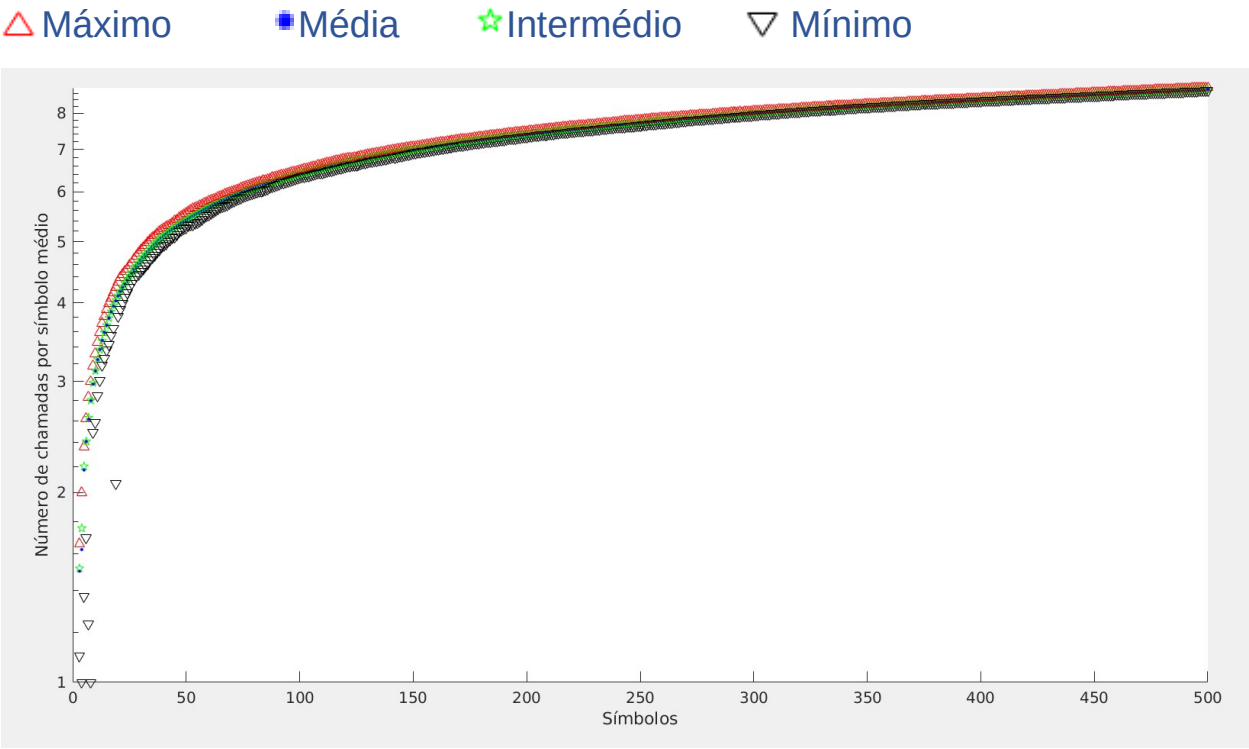
Analisando o gráfico abaixo apresentado podemos concluir que há um caso extremo em que apenas após analisar mais de 1200 símbolos é que o programa percebeu que estava a codificar erradamente. À medida que o número de Símbolos aumenta podemos verificar, também, que o número de símbolos extra se torna mais estável não variando tão drasticamente.

△ Máximo ▪ Média ☆ Intermédio ▽ Mínimo



Chamadas por símbolo

Como podemos verificar pelo gráfico abaixo o número de chamadas por símbolo não difere muito entre as 201 diferentes execuções para cada símbolo e apresenta um crescimento cada vez menor à medida que o número de símbolos aumenta.



Descodificação em Tempo Real

Raciocínio

Para descodificar a mensagem em tempo real é necessário proceder a um raciocínio diferente. A função já não será recursiva e terá de avaliar diversas possibilidades ao mesmo tempo. Temos de analisar todas as possibilidades para a determinada sequência de bits obtida, sendo que analisaremos bit a bit e veremos quais são as possibilidades. Por exemplo, considerando estes símbolos com os códigos correspondentes:

Simbolo	Código
A	0
B	01
C	011
D	111

Queremos descodificar a seguinte sequência de bits 0110111. De forma a descodificar em tempo real iriamos proceder aos seguintes passos:

1 0 – Possibilidades: [A, B, C]

2 01- Possibilidades: [AD, B, C]

3 011- Possbilidades: [AD, BD, C]

4 0110- Possibilidades: [CA, CB, CC]

5 01101- Possibilidades [CAD, CB, CC]

6 011011- Possibilidades [CAD, CBD, CC]

7 0110111- Possibilidades [CAD]

Código

Passar este racicínio para um algoritmo em C de forma a solucionar esta descodificação em tempo real revelou ser um grande desafio. Primeiramente, achei necessário a criação de um *pseudo-código* para servir de base para a passagem para linguagem de programação C. Após várias tentativas nenhum algoritmo pareceu funcionar, surgiram muitos problemas com utilização de várias variaveis com arrays multidimensionais. No apêndice deste relatório encontra-se a última tentativa e a que acreditei estar mais próxima da solução final para resolver o problema da descodificação em tempo real.

Não querendo desistir já desta solução decidi dar uma chance e tentar solucionar este algoritmo em Python. Para isso fiz algumas alterações ao ficheiro em C e criei um Script em Bash para guardar as variáveis necessárias geradas pelo ficheiro escrito em C num ficheiro para importar, posteriormente, para o algoritmo em python. Tanto o script como o algoritmo completo estão apresentados no Apêndice. O script em bash cria o ficheiro de texto com as informações necessárias e de seguida executa o algoritmo .py:

```
./realtime -t $1 $2 $3 > data.txt

python3 realtime.py
```

Em primeiro lugar, no algoritmo em Python é necessário extrair as informações guardadas no ficheiro “data.txt” criado com a execução do programa em C:

```
data = open('data.txt', 'r')
lines = data.readlines()
symbols = {}
count = 1
for line in lines:
    if count == 1:
        encodedMsg = line.strip()
    elif count == 2:
        originalMsg = line.strip()
    else:
        line = line.split()
        symbols[line[0]] = line[1]
    count+=1
data.close()
```

```
lines = data.readlines()
symbols = {}
count = 1
for line in lines:
    if count == 1:
        encodedMsg = line.strip()
    elif count == 2:
        originalMsg = line.strip()
    else:
        line = line.split()
        symbols[line[0]] = line[1]
    count+=1
data.close()
```

Extraimos, os símbolos e os seus respetivos códigos, armazenando-os num dicionário, a mensagem original (descodificada, que, no final, acabou por não ser necessária) e a mensagem codificada.

O racicíonio explicado acima foi traduzido e solucionado em Python com o seguinte código:

- tempDecode é uma variável extremamente importante para a resolução deste problema. A mesma é uma lista de listas. Ou seja, guarda listas com dois elementos sendo o elemento de indice 0 a mensagem descodificada e no indice 1 a respetiva codificação desta mesma sequência de símbolos.

- Primeiramente, percorremos as possibilidades a ser consideradas (para a primeira iteração consideramos todos os simbolos como sendo, individualmente, uma possibilidade). Caso a solução já tenha sido encontrado a mesma é imprimida e o programa terminado, se a quantidade de bits que estamos a considerar no momento da iteração for superior ao número de bits que podem ser representados pela sequência de simbolos do momento sao criadas n possibilidades novas derivadas da já existente, sendo n o número total de símbolos e a possibilidade considerada terá de ser eliminada. Por último verificamos se cada uma das possibilidades existentes está de acordo com a codificação pretendida.

```
    for j in range(0, rangeOffset):
        if tempDecode[j][1] == encodedMsg:
            tempDecode = [tempDecode[j][0], tempDecode[j][1]]
            print(f"{len(symbols)}\t{ma}\t{encodedMsg}\t{tempDecode[0]}\
t{encodedMsg == tempDecode[1]}")
            exit()
        if (i > len(tempDecode[j][1])):
            for k in range(0, len(symbols)):
                newElement = [tempDecode[j][0] + str(k), tempDecode[j][1] +
symbols.get(str(k))]
                remove.append(j)
                tempDecode.append(newElement)
        if not verify(tempDecode[j][1], i):
            remove.append(j)
```

A lista *"remove"* guarda os elementos que é necessário, posteriormente, remover da lista *"tempDecode".* Para proceder à seleção das novas possibilidades de descodificação usamos outra lista auxiliar *"newDecode"* e efetuamos o seguinte código:

```
    newDecode = list(tempDecode)
    for j in list(set(remove)):
        newDecode.remove(tempDecode[j])
    tempDecode = list(newDecode)
```

Criei também a função verify() para facilitar a verificação das possibilidades:

```
def verify(newElement,i):

    return True if newElement[0:i] == encodedMsg[0:i] else False
```

- Resta-nos agora repetir este processo para valores entre 1 e o número de bits da mensagem codificada, onde este valor representará o número de bits a considerar da mensagem codificada.

O código completo pode ser encontrado no apêndice deste relatório ou neste repositório do GitHub.

(https://github.com/sousaUA/RecursiveAndRealtime-Decoding)

Resultados

De forma a testar o algoritmo criado, apesar de ter testado manualmente, para vários números de simbolos entre 100 e 1000 com diferentes tamanhos de mensagem, bem como, seeds, criei um script “do_all_realtime.sh” para testar o programa para {3..1000} símbolos e tamanho de mensagem {10..20}, para a seed defini sempre 50. O programa revelou alguns bugs para 3 e 4 simbolos em alguns tamanhos de mensagem, mas para cinco ou mais símbolos não revelou qualquer problema mesmo testando para números como 1000 simbolos e tamanho de mensagem de 100, manualmente com o script “do_one_realtime.sh”. Os resultados do script “do_all_realtime.sh” foram guardados no ficheiro “realtimeResults.txt” com o seguinte formato:

NSimbolos	MaxPos	encoded	decoded	encoded==decoded
5	7	111111010101010111000	1113333302	True
5	7	1111110101010101110000110	11133333020	True
5	7	1111110101010101110001101 1	111333330201	True
5	7	11111101010101011100011011110	1113333302010	True
5	7	111111010101010111000110111100 10	11133333020104	True
5	7	111111010101010111000110111100 1001	111333330201043	True
5	7	111111010101010111000110111100 10010101	1113333302010433	True
5	7	111111010101010111000110111100 10010100	11133333020104332	True

Nsimbolos → Número de símbolos considerados

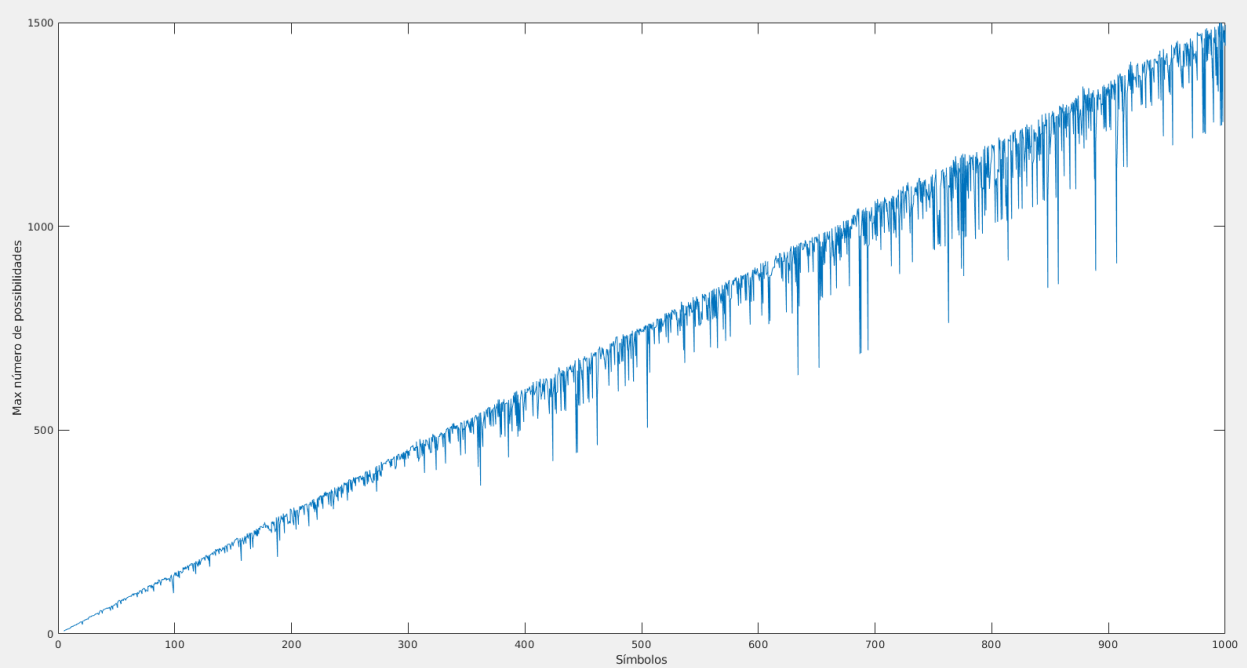
MaxPos → Máximo possibilidades consideradas ao mesmo tempo

Encoded → Mensagem codificada // Decoded→ Mensagem descodificada

encoded==decode → mensagem original codificada é igual à mensagem obtida?

Máximo número de possibilidades

O número máximo de possibilidades consideradas em simultâneo variou com o número de simbolos, como foram encontrados problemas para três e quatro símbolos apenas considerei números superiores a cinco, inclusive. Estes foram os resultados obtidos traduzidos num gráfico:



Analisando o gráfico, os resultados obtidos e alguns cálculos efetuados cheguei à conclusão que apesar de algumas discrepâncias, o número máximo de simbolos aparenta ser, aproximadamente, 1.5x o número de símbolos que consideramos.

Conclusão

Neste trabalho descodificamos recursivamente um código binário não instântaneo. Para esta solução calculamos o número de vezes que a função foi chamada para cada número de símbolos de três até quinhentos e com duzentas e uma seeds diferentes para cada. O verdadeiro desafio foi sim a descodificação em tempo real, ou seja, descodificar a mensagem à medida que vão sendo adicionados bits codificados. Para esta solução tive de recorrer ao Python, pois revelou ser uma linguagem muito mais apelativa para solucionar este algoritmo mais complexo. Foi notória a diferença de tempo de execução da versão original para esta versão em tempo real. Permitiu, também, recolher informação do número máximo de possibilidades consideradas ao mesmo tempo para cada número de símbolos.

Apêndice

Todos os resultados obtidos, bem como, ficheiros utilizados podem ser encontrados neste repositório do GitHub.

https://github.com/sousaUA/RecursiveAndRealtime-Decoding

Tentativa em C

Versão final da tentativa em C para a resolução de descodificação em tempo real:

```
static long realtime_decoder()
{
  long potencia = pow(2.0,(double) _c_->n_symbols);
  char tempDecode[potencia], newDecode[potencia],
string[_max_decoded_message_size_];
  int encoded_idx = 0, decoded_idx = 0;
  while (decoded_idx < _original_message_size_)
  {
    char tempMessage[encoded_idx+2], tempSymb[encoded_idx+2]; int n = -1, c =
-1, h = 0;
    while (++n <= encoded_idx)
      tempMessage[n] = _encoded_message_[decoded_idx + n];
    tempMessage[n] = '\0';
    for (int j = 0; j < _c_->n_symbols ; j++){
      n = -1;
      while (++n <= encoded_idx)
          tempSymb[n] = _c_->data[j].codeword[n];
      tempSymb[n] = '\0';
      //printf("%s = %s = %s\n", tempSymb, _c_->data[j].codeword,
tempMessage);
      n = -1;
      if (strcmp(tempSymb, tempMessage) == 0){
          for (int k = 0; k <= nPossibilities; k++)
          {
            strcpy(string, tempDecode[k]);
            if (verifySequence(string, tempMessage, decoded_idx, encoded_idx))
            {
              tempDecode[k] = newArray(tempDecode[k], j);
              nPossibilities++;
            }
            else nPossibilities--;
          }
          //tempDecode = newDecode;
          if (nPossibilities > maxPossibilities)
```

```
            maxPossibilities = nPossibilities;
        }
      }
    }
    return nPossibilities;
}
char* newArray (char* message, int j){
    char newArray[_max_decoded_message_size_];
    int n = -1;
    while (++n < _max_decoded_message_size_)
      newArray[n] = message[n];
    newArray[n] = j;
    n++;
    newArray[n] = '\0';
    return newArray;
}
bool verifySequence(char* decodedSequence, char* tempEncoded, int decoded_idx,
int encoded_idx){
    char encoded[_max_encoded_message_size_]; int n = 0;
    for (int i = 0; i <= decoded_idx+1; i++){
      for (int j = 0; j < _c_->n_symbols ; j++){
        if (strcmp(decodedSequence[i], _c_->data[j].codeword) == 0)
          encoded[n++] = _c_->data[j].codeword;
      }
    }
    if (strcmp(encoded, tempEncoded) == 0) return true;
    return false;
}
```

Script em Bash “do_one_realtime.sh”

Este Script em Bash cria um ficheiro com as informações necessárias para o algoritmo em Python e executa o ficheiro .py

```
#! /bin/bash
#!/usr/bin/env python3
ulimit -s 16384
if [ -e data.txt ]; then
  rm data.txt
fi

./realtime -t $1 $2 $3 > data.txt
python3 realtime.py
rm data.txt
```

Script em Bash “do_all_realtime.sh”

Este Script em Bash executa o algoritmo .py para diferentes combinações de número de simbolos e tamanho de mensagem:

```
#! /bin/bash
#!/usr/bin/env python3
ulimit -s 16384
if [ -e data.txt ]; then
  rm data.txt
fi
printf "%s\t%s\t%s\t%s\t%s\n" "NSimbolos" "MaxPos" "encoded" "decoded" "==?" >
realtimeResults.txt
for n in {3..1000}; do
    for i in {10..20}; do
        ./realtime -t $n $i 50 > data.txt
        python3 realtime.py >> realtimeResults.txt
        rm data.txt
    done
    echo "$n done"
done
```

Algoritmo em Python

Program em Python que descodifica uma mensagem em tempo real:

```
def verify(newElement,i):

      return True if newElement[0:i] == encodedMsg[0:i] else False

#Get all data
data = open('data.txt', 'r')
lines = data.readlines()
symbols = {}
count = 1
for line in lines:
    if count == 1:
        encodedMsg = line.strip()
    elif count == 2:
        originalMsg = line.strip()
    else:
        line = line.split()
        symbols[line[0]] = line[1]
    count+=1
data.close()
"""
encodedMsg -> Mensagem codificada
originalMsg -> Mensagem original (descodificada)
```

```
symbols -> dicionario com todos os simbolos e respetivos codigos
"""
tempDecode = [[str(i), symbols.get(str(i))] for i in range(0,len(symbols))]
ma = 0
for i in range(1, len(encodedMsg)+1):
    j = 0
    remove = [] #guarda os indices a remover da lista de possibilidades
    rangeOffset = len(tempDecode) #numero de possibilidades no inicio da
iteração
    if rangeOffset > ma: #guardar o numero maximo de possibilidades
consideradas ao mesmo tempo
            ma = rangeOffset
    for j in range(0, rangeOffset):
        if tempDecode[j][1] == encodedMsg: #Verificar se já foi encontrada a
solução
            tempDecode = [tempDecode[j][0], tempDecode[j][1]]
            print(f"{len(symbols)}\t{ma}\t{encodedMsg}\t{tempDecode[0]}\
t{encodedMsg == tempDecode[1]}")
            exit()
        if (i > len(tempDecode[j][1])):
            for k in range(0, len(symbols)):
                newElement = [tempDecode[j][0] + str(k), tempDecode[j][1] +
symbols.get(str(k))] #elemento a adicionar
                remove.append(j) #adicionar à lista de indices a eliminar mais
tarde
                tempDecode.append(newElement)
        if not verify(tempDecode[j][1], i):
            remove.append(j)
    newDecode = list(tempDecode)
    for j in list(set(remove)): #remover os elementos anteriormente definidos
        newDecode.remove(tempDecode[j])
    tempDecode = list(newDecode)
    remove=[]
print(f"ERRO\t{len(symbols)}\t{ma}\t{encodedMsg}\t{tempDecode}") #Caso não
cheguemos a uma solução
```

Resultados obtidos para cada número de símbolos em tempo real

O ficheiro com estas informações tem mais de dez mil linhas, logo para não ser demasiado extensivo para o relatório o mesmo pode ser consultado neste link.

(https://raw.githubusercontent.com/sousaUA/RecursiveAndRealtime-Decoding/main/A03/realtimeResults.txt)

Resultados obtidos para cada número de símbolos

Nº símbolos	número de chamadas por símbolo				símbolos extra			
ns	Mín	Méd	Int	Máx	Mín	Méd	Int	Max
3	1.100	1.501	1.516	1.660	2	8.6	9	19
4	1.000	1.624	1.756	1.999	0	16.5	8	141
5	1.367	2.172	2.201	2.362	3	31.3	15	235
6	1.695	2.410	2.410	2.620	3	37.6	32	502
7	1.239	2.613	2.629	2.830	4	39.0	33	139
8	1.000	2.799	2.801	2.999	0	40.2	32	228
9	2.493	2.973	2.986	3.176	11	49.1	46	747
10	2.585	3.118	3.121	3.318	13	57.1	54	1239
11	2.845	3.251	3.254	3.462	13	60.6	58	315
12	3.011	3.373	3.371	3.590	20	61.3	58	219
13	3.182	3.486	3.481	3.709	16	61.5	57	249
14	3.267	3.589	3.588	3.801	16	56.6	55	169
15	3.363	3.688	3.685	3.900	17	53.0	51	182
16	3.430	3.784	3.784	3.987	23	52.7	51	201
17	3.552	3.870	3.871	4.069	24	55.9	54	179
18	3.645	3.952	3.959	4.145	19	59.1	55	387
19	2.067	4.029	4.040	4.235	26	62.1	62	439
20	3.808	4.103	4.110	4.314	27	66.2	61	263

21	3.901	4.171	4.178	4.384	22	74.4	70	179
22	3.986	4.236	4.242	4.439	26	82.1	78	481
23	4.079	4.297	4.304	4.488	26	80.3	79	211
24	4.153	4.354	4.357	4.526	36	81.9	80	166
25	4.233	4.411	4.415	4.585	24	80.2	79	156
26	4.289	4.467	4.468	4.630	30	79.4	78	166
27	4.346	4.522	4.525	4.686	40	77.5	74	183
28	4.411	4.575	4.577	4.740	32	75.3	74	146
29	4.441	4.626	4.631	4.800	40	74.1	72	183
30	4.487	4.676	4.680	4.848	43	73.9	73	162
31	4.527	4.723	4.728	4.896	41	74.5	73	163
32	4.577	4.770	4.772	4.936	37	73.8	72	134
33	4.618	4.815	4.818	4.986	47	73.7	72	178
34	4.679	4.860	4.865	5.028	32	72.1	70	195
35	4.720	4.903	4.908	5.071	36	73.5	71	164
36	4.762	4.944	4.951	5.108	42	76.6	76	214
37	4.809	4.984	4.988	5.142	37	81.0	78	175
38	4.856	5.024	5.027	5.181	39	81.0	80	211
39	4.908	5.062	5.068	5.215	42	83.9	82	361
40	4.927	5.098	5.104	5.251	41	84.9	82	196
41	4.963	5.132	5.137	5.276	46	88.7	85	229
42	4.998	5.166	5.170	5.296	41	92.4	89	224

43	5.035	5.199	5.204	5.318	47	97.1	94	209
44	5.074	5.232	5.235	5.352	47	100.9	99	219
45	5.092	5.263	5.266	5.391	56	103.6	103	214
46	5.134	5.294	5.296	5.416	53	105.7	105	202
47	5.180	5.324	5.328	5.454	47	108.9	108	184
48	5.214	5.353	5.353	5.488	56	107.2	106	203
49	5.240	5.381	5.384	5.516	46	106.3	106	211
50	5.262	5.408	5.406	5.548	56	105.3	103	173
51	5.301	5.436	5.435	5.576	51	105.0	104	161
52	5.314	5.464	5.462	5.613	67	103.0	103	192
53	5.322	5.490	5.491	5.635	60	100.3	99	177
54	5.346	5.517	5.519	5.656	64	100.6	100	177
55	5.369	5.543	5.547	5.674	58	99.0	98	151
56	5.397	5.570	5.572	5.706	60	97.7	97	168
57	5.428	5.595	5.595	5.723	62	96.1	96	154
58	5.463	5.621	5.622	5.747	63	94.5	93	164
59	5.494	5.647	5.647	5.773	63	94.0	94	151
60	5.521	5.671	5.672	5.797	58	95.6	94	159
61	5.549	5.696	5.697	5.816	60	96.4	96	165
62	5.577	5.720	5.721	5.835	63	94.3	93	162
63	5.605	5.745	5.747	5.848	63	91.5	91	146
64	5.633	5.768	5.770	5.874	57	92.2	91	147

65	5.663	5.791	5.791	5.896	60	93.0	92	171
66	5.685	5.814	5.813	5.917	62	92.8	91	158
67	5.704	5.836	5.837	5.939	55	93.0	93	181
68	5.722	5.859	5.860	5.963	48	94.3	94	176
69	5.749	5.879	5.878	5.982	61	93.2	92	167
70	5.773	5.901	5.900	5.996	64	95.6	94	163
71	5.799	5.922	5.921	6.016	63	95.4	94	197
72	5.825	5.944	5.942	6.037	65	95.3	94	177
73	5.849	5.963	5.961	6.053	64	98.3	99	202
74	5.866	5.983	5.982	6.080	55	95.9	94	176
75	5.888	6.003	6.002	6.101	69	98.1	95	173
76	5.911	6.021	6.021	6.116	60	97.6	97	195
77	5.933	6.040	6.039	6.139	63	101.4	100	259
78	5.950	6.058	6.058	6.153	66	103.1	100	210
79	5.965	6.077	6.075	6.169	64	105.0	104	221
80	5.982	6.094	6.093	6.183	64	104.6	104	257
81	5.993	6.112	6.113	6.198	61	108.0	105	237
82	6.011	6.129	6.128	6.212	60	110.1	109	233
83	6.018	6.146	6.144	6.226	70	112.9	111	207
84	6.033	6.162	6.160	6.244	71	111.7	109	233
85	6.053	6.179	6.180	6.257	68	117.7	117	254
86	6.068	6.196	6.196	6.275	73	118.9	118	243

87	6.095	6.211	6.210	6.293	71	119.3	116	272
88	6.118	6.228	6.225	6.314	62	123.2	119	238
89	6.135	6.243	6.242	6.331	71	129.8	128	277
90	6.155	6.258	6.258	6.338	77	127.7	126	253
91	6.169	6.274	6.274	6.355	73	131.3	131	227
92	6.193	6.289	6.289	6.370	76	134.6	135	228
93	6.201	6.304	6.304	6.383	82	137.3	136	252
94	6.221	6.318	6.317	6.397	71	140.2	139	226
95	6.238	6.332	6.330	6.417	69	139.3	139	261
96	6.250	6.348	6.345	6.425	87	139.1	136	237
97	6.268	6.362	6.359	6.443	82	140.8	140	216
98	6.284	6.376	6.375	6.450	86	138.1	137	207
99	6.301	6.390	6.388	6.463	86	137.7	135	221
100	6.320	6.404	6.401	6.481	88	134.1	132	222
101	6.334	6.418	6.418	6.498	93	136.2	134	234
102	6.346	6.432	6.431	6.514	94	134.7	134	198
103	6.355	6.446	6.444	6.526	96	133.3	131	217
104	6.362	6.459	6.455	6.536	93	131.8	131	249
105	6.379	6.473	6.469	6.551	90	127.7	126	194
106	6.395	6.486	6.484	6.567	93	129.8	130	258
107	6.412	6.500	6.497	6.582	87	127.5	124	209
108	6.422	6.513	6.510	6.607	92	126.2	125	202

109	6.419	6.527	6.525	6.614	85	126.2	124	203
110	6.435	6.540	6.538	6.629	87	123.1	123	220
111	6.452	6.553	6.551	6.646	83	122.3	121	170
112	6.466	6.566	6.564	6.658	88	121.6	120	173
113	6.480	6.579	6.578	6.674	82	120.5	118	189
114	6.486	6.592	6.589	6.686	88	121.2	120	191
115	6.504	6.605	6.604	6.705	84	119.1	118	192
116	6.517	6.617	6.616	6.714	88	119.7	119	181
117	6.527	6.630	6.629	6.728	84	118.0	118	185
118	6.542	6.643	6.640	6.739	86	118.3	117	179
119	6.548	6.656	6.654	6.754	87	118.0	118	178
120	6.573	6.668	6.668	6.762	86	119.2	117	182
121	6.582	6.681	6.681	6.774	87	118.0	116	180
122	6.592	6.693	6.693	6.781	84	117.7	117	187
123	6.611	6.705	6.703	6.785	83	119.2	119	193
124	6.614	6.716	6.716	6.796	78	119.2	118	170
125	6.617	6.729	6.728	6.805	84	116.0	116	185
126	6.636	6.740	6.740	6.815	83	117.9	117	235
127	6.644	6.752	6.752	6.827	84	117.1	115	176
128	6.661	6.764	6.765	6.845	78	117.4	117	203
129	6.671	6.775	6.775	6.854	81	116.0	116	169
130	6.683	6.787	6.786	6.870	80	119.2	118	182

131	6.699	6.797	6.797	6.882	79	116.0	114	180
132	6.709	6.809	6.810	6.894	90	119.0	117	210
133	6.721	6.821	6.821	6.902	81	118.2	118	175
134	6.724	6.832	6.832	6.914	84	116.8	116	192
135	6.734	6.843	6.842	6.916	82	117.4	115	189
136	6.747	6.854	6.854	6.926	85	117.3	116	195
137	6.757	6.865	6.865	6.938	83	115.9	116	180
138	6.764	6.876	6.875	6.947	83	117.0	116	192
139	6.777	6.886	6.886	6.965	73	116.5	116	198
140	6.787	6.897	6.896	6.975	75	117.3	117	202
141	6.806	6.907	6.906	6.987	80	115.0	113	192
142	6.802	6.918	6.917	6.997	86	116.3	116	185
143	6.821	6.928	6.929	7.009	72	115.5	114	204
144	6.834	6.938	6.936	7.014	85	119.7	119	183
145	6.841	6.948	6.948	7.019	84	118.1	117	210
146	6.858	6.958	6.956	7.031	71	119.9	119	202
147	6.864	6.968	6.967	7.048	75	120.1	119	225
148	6.880	6.979	6.977	7.049	83	121.6	119	224
149	6.891	6.989	6.987	7.060	77	119.1	118	192
150	6.902	6.998	6.997	7.071	75	120.2	120	189
151	6.910	7.008	7.006	7.075	85	122.7	121	181
152	6.925	7.018	7.015	7.087	87	123.1	121	217

153	6.932	7.027	7.025	7.095	76	122.7	122	236
154	6.941	7.037	7.035	7.106	82	124.4	124	223
155	6.951	7.046	7.046	7.116	87	123.4	121	192
156	6.959	7.055	7.054	7.121	86	125.6	124	241
157	6.965	7.064	7.065	7.133	83	127.7	127	249
158	6.982	7.074	7.074	7.144	84	129.0	127	205
159	6.993	7.083	7.083	7.152	88	128.0	126	220
160	7.001	7.091	7.092	7.159	77	129.1	127	231
161	7.010	7.101	7.101	7.172	80	128.5	127	269
162	7.027	7.110	7.111	7.180	81	131.5	130	239
163	7.033	7.118	7.118	7.191	75	130.5	128	219
164	7.044	7.127	7.126	7.197	94	134.1	132	256
165	7.050	7.135	7.133	7.210	84	132.4	132	249
166	7.062	7.144	7.144	7.215	93	136.0	133	226
167	7.071	7.153	7.154	7.225	90	136.9	136	236
168	7.078	7.161	7.162	7.232	90	135.8	134	234
169	7.089	7.169	7.169	7.240	93	138.7	134	229
170	7.096	7.178	7.178	7.252	91	142.1	140	304
171	7.099	7.186	7.186	7.258	90	143.7	142	239
172	7.112	7.194	7.195	7.267	94	145.0	144	258
173	7.120	7.202	7.202	7.270	94	146.8	143	252
174	7.123	7.210	7.210	7.277	93	146.7	145	227

175	7.140	7.218	7.218	7.285	98	148.2	147	235
176	7.143	7.226	7.227	7.293	88	151.5	151	279
177	7.156	7.234	7.233	7.301	98	155.3	153	337
178	7.167	7.241	7.242	7.309	94	157.9	155	276
179	7.173	7.249	7.251	7.320	99	159.1	159	302
180	7.180	7.256	7.258	7.327	99	161.3	159	259
181	7.191	7.264	7.263	7.332	97	160.9	161	261
182	7.198	7.272	7.271	7.342	98	164.3	162	269
183	7.204	7.279	7.278	7.351	98	166.4	166	256
184	7.212	7.287	7.286	7.363	98	167.1	166	282
185	7.220	7.294	7.294	7.369	90	168.5	168	262
186	7.230	7.301	7.301	7.373	104	171.3	170	285
187	7.232	7.309	7.309	7.383	103	173.1	172	291
188	7.241	7.315	7.316	7.387	111	175.9	177	258
189	7.250	7.323	7.323	7.400	110	172.1	172	253
190	7.251	7.330	7.330	7.408	87	174.7	174	514
191	7.264	7.337	7.337	7.419	100	173.4	172	287
192	7.278	7.344	7.345	7.422	113	173.4	172	253
193	7.283	7.351	7.353	7.431	94	174.6	174	251
194	7.290	7.358	7.358	7.438	118	172.2	171	276
195	7.295	7.365	7.366	7.450	120	174.5	173	281
196	7.307	7.372	7.371	7.452	113	172.9	171	312

197	7.312	7.379	7.380	7.459	117	171.7	173	246
198	7.321	7.386	7.385	7.468	101	172.1	171	317
199	7.329	7.393	7.393	7.474	127	170.4	167	278
200	7.328	7.400	7.399	7.480	124	169.2	170	253
201	7.345	7.407	7.405	7.488	119	165.4	165	261
202	7.347	7.414	7.413	7.492	115	166.9	166	235
203	7.358	7.421	7.420	7.501	121	165.3	162	257
204	7.362	7.428	7.426	7.508	121	165.2	165	262
205	7.369	7.435	7.433	7.513	119	163.6	162	228
206	7.380	7.442	7.442	7.524	127	162.3	159	288
207	7.382	7.448	7.449	7.527	117	161.3	160	266
208	7.387	7.456	7.455	7.537	123	159.8	157	244
209	7.393	7.462	7.462	7.545	119	160.2	159	285
210	7.403	7.469	7.468	7.551	120	158.5	156	255
211	7.407	7.476	7.476	7.553	109	157.4	157	236
212	7.415	7.483	7.483	7.559	114	155.4	153	245
213	7.429	7.490	7.488	7.571	119	157.7	156	267
214	7.431	7.496	7.495	7.578	118	155.4	153	263
215	7.441	7.503	7.504	7.586	109	153.4	153	290
216	7.441	7.509	7.509	7.594	111	152.5	151	248
217	7.440	7.516	7.517	7.601	121	152.5	150	328
218	7.450	7.522	7.523	7.608	109	152.2	151	234

219	7.461	7.529	7.530	7.613	111	152.0	151	222
220	7.465	7.536	7.536	7.623	108	149.3	147	206
221	7.471	7.543	7.543	7.628	112	151.3	148	224
222	7.482	7.549	7.549	7.631	115	150.1	147	243
223	7.487	7.556	7.556	7.638	114	150.8	149	238
224	7.493	7.562	7.562	7.637	100	150.8	150	250
225	7.497	7.569	7.569	7.646	118	149.3	147	258
226	7.509	7.575	7.576	7.646	111	147.8	147	228
227	7.508	7.582	7.581	7.659	111	148.2	146	209
228	7.520	7.588	7.589	7.658	108	146.3	146	207
229	7.519	7.595	7.595	7.669	107	145.1	145	224
230	7.530	7.601	7.601	7.670	108	143.8	143	284
231	7.540	7.608	7.608	7.686	104	145.3	144	242
232	7.543	7.614	7.615	7.683	107	142.5	143	212
233	7.550	7.621	7.620	7.689	99	145.7	144	211
234	7.554	7.627	7.627	7.698	106	142.9	141	217
235	7.560	7.633	7.633	7.702	113	143.5	143	195
236	7.566	7.640	7.639	7.712	104	142.0	142	240
237	7.577	7.646	7.645	7.716	111	142.5	142	213
238	7.577	7.652	7.652	7.731	109	143.0	142	210
239	7.591	7.658	7.658	7.734	107	143.0	142	212
240	7.594	7.665	7.665	7.736	112	146.5	146	284

241	7.594	7.671	7.672	7.746	106	139.9	139	197
242	7.594	7.677	7.678	7.758	110	143.3	142	197
243	7.600	7.683	7.684	7.760	99	143.0	142	219
244	7.608	7.690	7.690	7.764	105	141.3	140	345
245	7.616	7.696	7.697	7.764	103	141.0	140	252
246	7.623	7.701	7.702	7.768	100	141.3	141	216
247	7.622	7.708	7.707	7.778	111	143.6	142	256
248	7.633	7.714	7.715	7.783	111	140.6	140	206
249	7.643	7.719	7.719	7.784	103	141.5	141	207
250	7.643	7.726	7.726	7.792	100	140.3	137	209
251	7.654	7.731	7.733	7.800	99	141.4	139	210
252	7.660	7.737	7.739	7.810	104	140.0	139	219
253	7.667	7.744	7.745	7.811	106	140.0	138	221
254	7.677	7.749	7.751	7.817	105	138.9	137	202
255	7.674	7.755	7.757	7.826	107	139.2	138	211
256	7.683	7.761	7.761	7.835	101	140.9	140	215
257	7.690	7.766	7.768	7.836	108	138.6	137	215
258	7.695	7.772	7.773	7.841	96	140.2	140	211
259	7.699	7.778	7.778	7.849	106	142.3	142	231
260	7.705	7.783	7.783	7.850	108	140.4	139	224
261	7.708	7.789	7.790	7.858	107	140.8	141	195
262	7.725	7.795	7.796	7.863	105	141.1	140	246

263	7.725	7.801	7.802	7.863	108	139.5	139	207
264	7.731	7.807	7.809	7.872	107	141.9	140	212
265	7.741	7.812	7.814	7.883	101	141.0	139	227
266	7.748	7.818	7.818	7.887	100	140.3	139	214
267	7.753	7.824	7.826	7.890	109	139.5	138	200
268	7.758	7.829	7.830	7.895	102	139.6	138	219
269	7.767	7.835	7.835	7.896	107	139.9	139	229
270	7.771	7.840	7.840	7.904	109	139.7	139	212
271	7.782	7.845	7.847	7.910	105	142.2	141	218
272	7.783	7.851	7.851	7.919	106	143.1	140	233
273	7.791	7.856	7.858	7.919	110	141.3	140	214
274	7.801	7.862	7.862	7.931	108	139.0	139	204
275	7.802	7.867	7.867	7.932	106	138.9	137	203
276	7.810	7.873	7.872	7.939	103	140.0	139	243
277	7.813	7.878	7.877	7.937	109	139.9	138	209
278	7.821	7.883	7.884	7.946	103	141.1	141	250
279	7.823	7.889	7.890	7.944	104	140.4	140	223
280	7.831	7.893	7.893	7.955	101	140.0	139	225
281	7.835	7.899	7.900	7.956	109	142.4	142	231
282	7.841	7.905	7.905	7.965	102	141.7	141	219
283	7.849	7.909	7.910	7.972	107	140.6	140	214
284	7.847	7.915	7.916	7.973	103	141.1	139	221

285	7.857	7.920	7.920	7.981	103	142.3	142	227
286	7.860	7.925	7.927	7.983	106	139.8	137	217
287	7.869	7.930	7.931	7.991	105	143.2	140	195
288	7.871	7.935	7.936	7.994	105	140.0	137	237
289	7.878	7.940	7.941	7.992	109	139.9	137	231
290	7.889	7.946	7.946	8.002	102	142.8	142	242
291	7.890	7.950	7.950	8.010	103	142.8	141	235
292	7.895	7.955	7.954	8.013	107	143.9	144	260
293	7.908	7.960	7.961	8.017	107	142.7	141	236
294	7.899	7.965	7.965	8.025	102	145.5	145	224
295	7.911	7.970	7.969	8.023	110	144.3	143	227
296	7.916	7.975	7.974	8.025	100	144.6	144	212
297	7.920	7.980	7.981	8.036	105	143.8	144	242
298	7.925	7.985	7.983	8.038	100	145.5	144	221
299	7.930	7.990	7.990	8.043	103	144.9	146	219
300	7.937	7.994	7.994	8.054	107	146.4	145	235
301	7.942	7.999	7.999	8.051	110	146.1	146	264
302	7.951	8.004	8.003	8.058	107	145.0	144	215
303	7.958	8.009	8.008	8.060	109	146.7	146	219
304	7.967	8.014	8.013	8.066	108	144.6	143	266
305	7.967	8.019	8.019	8.076	104	147.2	146	249
306	7.968	8.023	8.024	8.070	101	149.1	146	292

307	7.975	8.028	8.028	8.080	104	149.6	150	213
308	7.982	8.033	8.033	8.081	115	150.8	149	257
309	7.985	8.038	8.038	8.086	112	148.6	145	233
310	7.991	8.043	8.043	8.090	103	152.0	150	223
311	7.997	8.047	8.048	8.098	101	148.7	146	306
312	8.003	8.052	8.053	8.100	108	151.3	150	234
313	8.010	8.056	8.058	8.101	112	151.3	150	254
314	8.013	8.061	8.061	8.104	102	153.3	153	232
315	8.018	8.066	8.067	8.109	108	150.1	149	211
316	8.025	8.071	8.070	8.113	114	153.6	152	240
317	8.027	8.075	8.075	8.119	118	152.7	152	232
318	8.030	8.079	8.080	8.121	105	151.8	148	235
319	8.042	8.085	8.085	8.127	113	155.9	154	223
320	8.041	8.089	8.089	8.131	113	156.7	157	257
321	8.047	8.093	8.092	8.139	106	154.9	151	247
322	8.047	8.098	8.097	8.142	109	153.6	152	242
323	8.057	8.102	8.102	8.154	109	155.7	155	255
324	8.060	8.107	8.108	8.151	106	158.0	156	245
325	8.068	8.111	8.111	8.159	114	157.3	156	245
326	8.074	8.115	8.116	8.164	112	162.1	159	245
327	8.079	8.120	8.120	8.164	116	160.5	160	236
328	8.079	8.124	8.124	8.173	116	161.5	161	259

329	8.087	8.129	8.130	8.175	104	159.4	159	238
330	8.090	8.133	8.133	8.176	112	160.4	159	240
331	8.094	8.137	8.138	8.181	111	162.8	161	258
332	8.095	8.141	8.143	8.185	115	161.7	161	233
333	8.105	8.146	8.146	8.186	116	162.1	160	253
334	8.109	8.150	8.150	8.190	111	165.8	164	263
335	8.108	8.154	8.154	8.199	111	163.0	162	376
336	8.110	8.159	8.159	8.206	105	165.2	165	264
337	8.124	8.163	8.163	8.207	123	163.3	162	281
338	8.122	8.167	8.166	8.211	113	165.3	163	253
339	8.129	8.171	8.170	8.212	113	164.7	161	311
340	8.133	8.175	8.175	8.216	116	168.8	165	296
341	8.135	8.179	8.179	8.220	115	171.2	169	326
342	8.140	8.183	8.183	8.221	115	170.0	168	294
343	8.143	8.187	8.188	8.227	123	171.5	172	276
344	8.149	8.191	8.192	8.231	124	173.3	170	281
345	8.153	8.196	8.196	8.238	122	175.3	174	279
346	8.160	8.200	8.199	8.239	119	174.0	171	289
347	8.162	8.203	8.205	8.241	122	174.3	173	278
348	8.167	8.207	8.207	8.249	115	176.1	175	280
349	8.169	8.211	8.211	8.252	125	180.8	178	265
350	8.172	8.215	8.215	8.258	125	178.0	175	295

351	8.175	8.219	8.220	8.258	126	179.4	178	257
352	8.182	8.224	8.223	8.263	129	177.5	175	325
353	8.186	8.227	8.227	8.267	119	182.8	179	305
354	8.192	8.231	8.231	8.271	117	181.6	180	291
355	8.199	8.235	8.235	8.272	125	182.3	180	260
356	8.199	8.239	8.239	8.278	134	183.9	181	300
357	8.202	8.243	8.244	8.283	118	183.8	184	277
358	8.207	8.246	8.245	8.285	121	189.0	187	290
359	8.210	8.250	8.250	8.292	123	187.8	187	295
360	8.213	8.254	8.253	8.293	126	191.4	189	318
361	8.214	8.258	8.259	8.295	129	193.0	187	297
362	8.222	8.262	8.262	8.299	136	193.9	192	357
363	8.228	8.266	8.265	8.306	136	193.5	192	289
364	8.227	8.269	8.270	8.306	135	199.9	201	307
365	8.232	8.273	8.273	8.311	128	198.5	194	352
366	8.238	8.277	8.277	8.316	125	196.8	196	304
367	8.242	8.281	8.282	8.319	128	199.4	200	306
368	8.246	8.285	8.285	8.324	132	202.3	199	311
369	8.250	8.288	8.288	8.327	132	201.2	199	290
370	8.254	8.291	8.293	8.329	136	204.8	203	359
371	8.257	8.295	8.297	8.336	136	203.3	204	343
372	8.264	8.298	8.300	8.337	134	204.7	203	304

373	8.266	8.302	8.303	8.340	137	209.3	208	357
374	8.271	8.306	8.307	8.344	139	210.5	208	429
375	8.275	8.309	8.310	8.345	141	207.1	208	309
376	8.276	8.313	8.314	8.353	142	210.6	211	321
377	8.279	8.317	8.318	8.356	140	216.5	216	346
378	8.284	8.320	8.322	8.362	134	210.3	208	311
379	8.286	8.324	8.325	8.365	141	214.8	211	346
380	8.292	8.328	8.327	8.368	141	215.6	214	305
381	8.292	8.331	8.332	8.367	143	214.8	213	353
382	8.298	8.335	8.335	8.374	152	215.5	213	315
383	8.301	8.338	8.338	8.379	158	208.1	207	324
384	8.305	8.342	8.343	8.387	159	210.2	209	327
385	8.304	8.345	8.346	8.385	145	211.9	211	345
386	8.311	8.349	8.350	8.387	145	213.8	212	368
387	8.315	8.352	8.354	8.392	160	214.9	212	361
388	8.317	8.356	8.356	8.393	161	215.3	213	328
389	8.324	8.359	8.359	8.396	164	210.7	208	391
390	8.328	8.363	8.363	8.398	166	213.1	211	313
391	8.330	8.366	8.367	8.404	159	209.0	208	306
392	8.334	8.370	8.370	8.407	164	209.2	206	304
393	8.330	8.373	8.373	8.412	133	209.4	209	318
394	8.337	8.377	8.377	8.416	151	211.1	208	405

395	8.342	8.381	8.381	8.422	163	206.9	205	305
396	8.345	8.384	8.384	8.424	153	207.4	205	337
397	8.348	8.388	8.389	8.425	156	205.6	204	331
398	8.349	8.391	8.391	8.431	158	203.4	201	351
399	8.354	8.395	8.396	8.433	156	204.8	206	336
400	8.362	8.398	8.399	8.435	155	200.9	199	311
401	8.364	8.401	8.402	8.441	162	203.3	201	340
402	8.367	8.405	8.405	8.445	155	203.6	204	426
403	8.369	8.409	8.409	8.447	157	201.0	199	328
404	8.372	8.412	8.412	8.450	155	201.8	200	344
405	8.378	8.415	8.416	8.454	154	201.6	203	306
406	8.377	8.419	8.420	8.458	157	201.1	199	324
407	8.385	8.422	8.423	8.465	153	199.5	198	286
408	8.387	8.426	8.426	8.470	147	199.5	199	320
409	8.395	8.429	8.430	8.471	145	195.3	193	293
410	8.395	8.433	8.433	8.471	146	196.3	195	304
411	8.397	8.436	8.436	8.481	155	194.7	194	289
412	8.398	8.440	8.440	8.481	155	194.3	194	331
413	8.404	8.443	8.444	8.485	147	197.0	194	332
414	8.410	8.447	8.446	8.489	150	193.8	191	294
415	8.412	8.450	8.450	8.490	147	194.2	192	303
416	8.414	8.454	8.454	8.497	147	193.5	192	278

417	8.419	8.457	8.458	8.498	146	189.7	188	306
418	8.424	8.460	8.460	8.499	149	190.8	190	283
419	8.425	8.464	8.464	8.503	152	192.1	191	295
420	8.427	8.467	8.467	8.509	145	188.3	186	278
421	8.432	8.471	8.470	8.509	138	187.2	186	314
422	8.432	8.475	8.474	8.514	143	189.7	189	287
423	8.436	8.478	8.478	8.519	139	187.4	186	318
424	8.442	8.481	8.481	8.520	133	185.0	181	270
425	8.441	8.485	8.485	8.530	142	186.9	184	262
426	8.445	8.488	8.488	8.531	142	185.8	183	267
427	8.445	8.492	8.492	8.533	147	188.3	187	311
428	8.446	8.495	8.495	8.535	139	186.0	184	294
429	8.456	8.498	8.498	8.538	135	185.9	184	272
430	8.457	8.501	8.501	8.542	140	187.0	185	290
431	8.460	8.505	8.505	8.546	142	181.7	180	268
432	8.468	8.508	8.509	8.550	140	183.8	182	291
433	8.467	8.512	8.512	8.550	139	185.7	185	323
434	8.473	8.515	8.515	8.553	135	182.0	182	274
435	8.478	8.519	8.519	8.558	140	185.5	185	288
436	8.479	8.522	8.522	8.564	137	182.7	182	364
437	8.480	8.525	8.525	8.564	137	182.4	181	299
438	8.486	8.529	8.529	8.569	142	179.3	178	302

439	8.489	8.532	8.531	8.573	140	185.0	185	274
440	8.491	8.535	8.535	8.574	142	180.5	177	265
441	8.491	8.538	8.538	8.578	134	179.8	178	297
442	8.496	8.542	8.542	8.581	143	178.9	177	253
443	8.501	8.546	8.545	8.588	130	179.9	180	288
444	8.500	8.549	8.549	8.586	132	180.2	177	309
445	8.506	8.552	8.551	8.590	136	178.3	177	295
446	8.510	8.555	8.556	8.593	138	176.9	175	277
447	8.514	8.559	8.558	8.599	136	176.5	175	264
448	8.514	8.562	8.561	8.602	135	178.3	177	280
449	8.523	8.565	8.565	8.603	136	177.4	176	299
450	8.521	8.568	8.567	8.611	131	176.8	176	267
451	8.525	8.572	8.571	8.613	139	176.5	175	300
452	8.527	8.575	8.575	8.612	121	176.7	175	298
453	8.536	8.579	8.578	8.621	140	176.3	176	241
454	8.538	8.582	8.580	8.620	129	177.2	177	276
455	8.541	8.585	8.586	8.625	124	175.5	174	272
456	8.544	8.588	8.588	8.627	131	173.2	173	262
457	8.547	8.591	8.591	8.633	136	174.3	174	301
458	8.551	8.595	8.594	8.634	131	175.8	172	345
459	8.553	8.598	8.597	8.640	135	170.4	170	307
460	8.554	8.601	8.600	8.643	131	174.0	173	278

461	8.559	8.604	8.604	8.646	134	174.5	173	265
462	8.561	8.608	8.607	8.650	129	171.5	170	284
463	8.568	8.611	8.610	8.651	130	172.4	170	287
464	8.568	8.614	8.613	8.655	134	170.6	169	266
465	8.572	8.617	8.617	8.658	130	171.0	168	253
466	8.577	8.620	8.620	8.661	132	171.5	170	255
467	8.585	8.623	8.622	8.666	133	173.0	171	242
468	8.581	8.627	8.627	8.668	129	171.0	169	303
469	8.586	8.630	8.630	8.672	128	173.2	172	271
470	8.588	8.633	8.632	8.675	127	171.6	169	321
471	8.593	8.636	8.635	8.677	117	171.3	170	320
472	8.596	8.639	8.638	8.683	128	170.8	169	283
473	8.598	8.642	8.642	8.685	129	166.6	166	249
474	8.601	8.646	8.644	8.687	134	169.2	166	279
475	8.607	8.649	8.649	8.693	131	172.2	171	307
476	8.606	8.652	8.651	8.691	128	169.4	169	318
477	8.609	8.655	8.655	8.700	130	170.7	169	278
478	8.612	8.659	8.658	8.700	125	169.7	168	263
479	8.615	8.661	8.660	8.706	131	170.0	168	243
480	8.618	8.665	8.664	8.709	123	169.1	167	255
481	8.622	8.668	8.667	8.708	132	168.9	168	259
482	8.629	8.671	8.671	8.714	124	170.7	169	310

483	8.628	8.674	8.673	8.722	137	170.4	167	265
484	8.634	8.677	8.676	8.717	130	170.7	169	268
485	8.631	8.680	8.680	8.723	131	169.2	166	238
486	8.640	8.683	8.683	8.725	125	171.1	170	250
487	8.642	8.686	8.686	8.729	130	169.6	168	255
488	8.644	8.689	8.689	8.732	129	168.2	168	268
489	8.648	8.692	8.692	8.732	124	171.0	169	252
490	8.653	8.696	8.695	8.739	129	168.9	167	236
491	8.655	8.698	8.698	8.738	125	168.2	167	278
492	8.659	8.702	8.701	8.742	131	170.1	168	275
493	8.654	8.704	8.704	8.748	128	169.3	167	253
494	8.660	8.708	8.707	8.751	132	169.8	168	287
495	8.665	8.711	8.711	8.750	128	167.8	167	242
496	8.673	8.714	8.714	8.757	133	171.3	170	275
497	8.674	8.716	8.716	8.759	125	169.0	169	234
498	8.674	8.720	8.720	8.760	131	170.7	170	272
499	8.679	8.723	8.722	8.767	133	168.9	168	259
500	8.686	8.725	8.725	8.771	123	166.8	165	261