

Aplicação de métodos bioinformáticos na análise de séries de temperaturas

Afonso, João Miguel - A71874, Moreira, David A. Cardoso - A64287

Resumo—Os algoritmos e metodologias de análise de sequências biológicas podem ser adaptados em diversas temáticas. Neste trabalho vamos explorar a sua extensão para outras áreas, recorrendo a um conjunto de dados com as temperaturas da superfície terrestre. Para tal, procedeu-se à sua análise estatística e discretização, seguindo-se aplicação de um leque de algoritmos de alinhamento e agrupamento de sequências, procura de padrões, entre outros.

Keywords—séries temporais, sequências biológicas, temperaturas, análise de dados.

I. CASO DE ESTUDO

A análise de sequências é apresentada pela bioinformática como a aplicação de um conjunto variado de métodos analíticos sobre sequências biológicas, com a finalidade de obter informações relevantes sobre as suas características.

Estas sequências, tais como o ADN ou as proteínas, são caracterizadas por serem relativamente longas, mas constituídas por um alfabeto de tamanho reduzido (por exemplo, 4 símbolos para o ADN e 20 para as proteínas). Muitos dos algoritmos para análise de sequências biológicas foram desenvolvidos tendo em conta essas mesmas características. No entanto, estes algoritmos podem ainda ser utilizados noutros domínios, onde os dados gerados tenham natureza sequencial.

O objetivo deste projeto é desenvolver uma análise dos dados de séries temporais de temperatura da superfície terrestre, através da aplicação de métodos bioinformáticos. É ainda esperado que esta análise possa vir a gerar conclusões pertinentes acerca dos padrões de alteração climática.

II. TECNOLOGIAS UTILIZADAS

Um dos objetivos deste trabalho é a sua total implementação na linguagem de programação *python*. Como tal, vão ser utilizadas várias bibliotecas desta linguagem, destacando-se o *Scipy*, *Numpy* e o *Pandas*. Não sendo de interesse para este trabalho uma exploração exaustiva dos métodos providenciados por cada uma das bibliotecas enunciadas, a sua utilização será referida e explicada sempre se considere necessário. No que diz respeito às interações de *input* e *output*, estas serão feitas através da leitura e escrita de ficheiros (*csv* para a leitura de dados e *png* / *pdf* para exportar os resultados obtidos).

III. METODOLOGIA E RESULTADOS

A. Dados de entrada

Os conjuntos de dados utilizados na realização deste trabalho, provenientes da página web *kaggle*[2], são compostos por vários ficheiros em formato *csv*. Neste trabalho será utilizado

um subconjunto do ficheiro que contém as temperaturas para as principais cidades, uma vez que possui todas as informações necessárias à realização do projeto, sendo cerca de trinta vezes inferior ao ficheiro com os dados das mais de três mil cidades.

Com a finalidade de poder utilizar estes dados nos vários algoritmos, estes vão ser carregados em memória, sendo mantidos num dicionário em que as chaves correspondem a cada uma das cidades e os valores são listas com as temperaturas correspondentes.

A título de exemplo, vão ser utilizadas as temperaturas das seguintes cidades no decorrer do documento:

- 1) Europa: Londres, Madrid, Moscovo, Paris
- 2) Ásia: Nova Deli, Pequim, Tóquio
- 3) América do Norte: Los Angeles, Nova Iorque, Toronto
- 4) América do Sul: Lima, Rio de Janeiro
- 5) África: Cairo, Luanda
- 6) Oceânia: Sidney

B. Análise estatística

Partindo do dicionário já preenchido, é agora interessante utilizar um conjunto de métricas estatísticas, por forma a melhor interpretar os dados a ser utilizados. Podemos efetuar essas métricas em dois níveis principais, analisando quer o conjunto global das cidades, quer os dados de cada uma delas.

Sobre o total de dados propomos que sejam calculadas apenas as temperaturas mínima, máxima e média, servindo estas de enquadramento global para o problema proposto. Sobre o conjunto de dados selecionado na alínea anterior, obtivemos os seguintes valores:

- 1) Temperatura mínima: -19.37 °C
- 2) Temperatura máxima: 36.34 °C
- 3) Temperatura média: 13.54 °C

Referindo-nos já às séries temporais, não vamos ainda retirar conclusões sobre as temperaturas que representam. Assim, serão apenas efetuadas medições básicas, correspondendo ao número de cidades, bem como à média, moda, mediana e alcance do tamanho das séries de cada cidade. Foram calculados os seguintes valores para o conjunto de dados referido:

- 4) Número de séries (cidades): 15
- 5) Média: 2498.9 registos
- 6) Mediana: 2394 registos
- 7) Moda: 3166 registos
- 8) Alcance: 1645 (3166 – 1521)

A partir dos dados obtidos podemos observar que existe uma grande variação no número de registos de temperaturas entre as várias cidades, chegando esta a ser superior ao número de medições para as cidades com os menores conjuntos de dados, realidade que terá de ser tida em conta no decorrer do processo de discretização.

C. Visualização

Antes de se começarem a efetuar alterações ao conjunto de dados, criaram-se métodos para facilitar a sua visualização. Para clarificar o trabalho realizado, selecionaram-se os registos de temperaturas da cidade de Londres, sendo estes colocados no gráfico da Figura 1. [Mais exemplos no Anexo A]

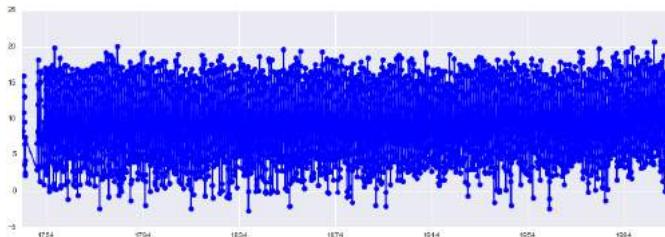


Figura 1. Registo de temperaturas 1743-2013 (Londres)

Como podemos observar, devido ao elevado número de pontos no gráfico, este mostra-se quase imperceptível, esboçando-se apenas um ligeiro aumento das temperaturas a partir do início do século XX. Supondo que o elevado ruído da imagem se deva à sobreposição das linhas representantes das variações anuais de temperaturas, foi gerado o gráfico contendo apenas os valores dos últimos dez anos em registo, encontrando-se o resultado expresso na Figura 2

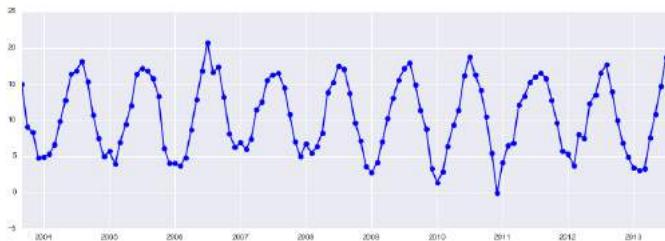


Figura 2. Registo de temperaturas 2003-2013 (Londres)

Uma vez que as variações mensais são já completamente perceptíveis, podemos confirmar a hipótese proposta. No entanto, o nosso objetivo é analisar todo o conjunto de dados, assim sendo, resolvemos representar graficamente apenas um dos meses para cada ano, tendo sido selecionado o mês de Julho. Os resultados obtidos encontram-se na Figura 3

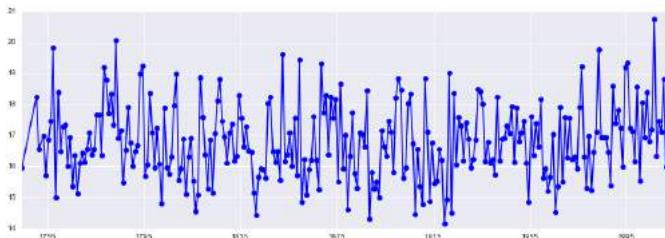


Figura 3. Registo de temperaturas Julho 1743-2013 (Londres)

A seleção de um só mês acentuou as variações já verificadas na Figura 1, uma vez que permitiu reduzir para metade o alcance dos valores das temperaturas (eixo vertical).

D. Discretização

As sequências biológicas têm normalmente dimensões bastante elevadas. Porém, dependendo se nos estamos a referir ao ADN/ARN ou às sequências proteicas, estas compostas apenas por quatro ou vinte símbolos já definidos.

Assim sendo, uma vez que os valores das temperaturas para as várias cidades se encontram em formato numérico, cria-se a necessidade de proceder à discretização dos dados em questão.

Para tal, de forma a generalizar ao máximo todo o processo, foi criada a classe *DataSet* (declarada em baixo), que pode ser instanciada fornecendo apenas o conjunto original de dados e o alfabeto pretendido para a discretização (lista constituída por símbolos em formato inteiro, textual, ou qualquer outro permitido pela linguagem *python*). Esta classe vai ainda conter vários métodos de discretização, que podem ser posteriormente chamados sobre as suas instâncias, permitindo assim um maior nível de abstração de dados aquando da implementação dos vários algoritmos de análise das sequências.

Definimos também a função *replace*, que recebe os limites dos vários intervalos de discretização, bem como o alfabeto com os símbolos correspondentes a cada intervalo, devolvendo o resultado da substituição dos valores neles contidos pelos símbolos pretendidos.

Posto isto, é necessário implementar os métodos de discretização. Serão abordadas três metodologias distintas, sendo elas a distribuição por amplitude, frequência e agregação simbólica[1]. Para tal, a fim de permitir uma melhor percepção dos vários métodos desenvolvidos, serão utilizados os dados correspondentes aos últimos cinco anos da série temporal. O alfabeto de discretização vai ser constituído por quatro símbolos distintos (de forma similar ao ADN). Para poder sobrepor no mesmo gráfico os dados pré e pós-discretização, os quatro símbolos vão ser representados em formato decimal, correspondendo ao mínimo, 1/3, 2/3 e máximo das temperaturas da série completa.

1) EWD: do inglês *Equal Width Discretization*, este método consiste em calcular os valores máximo e mínimo da série temporal, dividindo o intervalo entre eles pelos vários símbolos do alfabeto fornecido, de forma a que o alcance correspondente a cada sub-intervalo seja o mesmo

discretize.py

```
# Equal Width Discretization (EWD)
def EWD (self):
    scale = self.max - self.min + 1
    step = scale / self.alphabetSize
    intervals = [x * step + self.min for x in range(1,
        self.alphabetSize)]
    intervals.append(self.max+1)
    return self.replace(self.data, intervals)
```

Para melhor ilustrar a sua aplicação, apresenta-se o excerto de código anterior, correspondente à implementação do método referido, seguido-se o resultado por ele produzido (Figura 4), de acordo com as condições inicialmente estabelecidas.

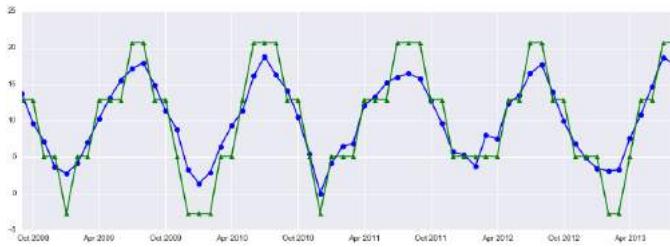


Figura 4. Discretização de igual largura

Como podemos observar no gráfico produzido, este método é demasiado fiel aos dados iniciais, mantendo pequenas variações (ruído), que deviam ser desprezadas. Um exemplo disto é a temperatura mínima do inverno 2011/2012, que em princípio deveria ter encaixado no nível inferior.

2) *EFD*: do inglês *Equal Frequency Discretization*, este método atua de forma semelhante ao anterior, na medida em que apenas divide o conjunto de dados em vários intervalos. Contudo, desta vez os intervalos não vão ter a mesma dimensão, variando esta de forma a que cada um tenha aproximadamente o mesmo número de elementos.

discretize.py

```
# Equal Frequency Discretization (EFD)
def EFD (self):
    step = self.dataSize / self.alphabetSize
    srt = sorted(self.data)
    intervals = []
    for i in [x*step for x in range(1, self.alphabetSize)]:
        intervals.append( srt[round(i)] )
    intervals.append(srt[-1]+1)
    return self.replace(self.data, intervals)
```

Como se pode ver no excerto de código anterior, para calcular os vários intervalos começa-se por calcular o número de elementos da série temporal, dividido pelo número de elementos do alfabeto, sendo o resultado guardado na variável *step*. Posto isto, percorre-se a série temporal ordenada, tomando como fronteiras do intervalo os valores cujos índices são múltiplos do *step* calculado, procedendo-se por fim à substituição dos valores.

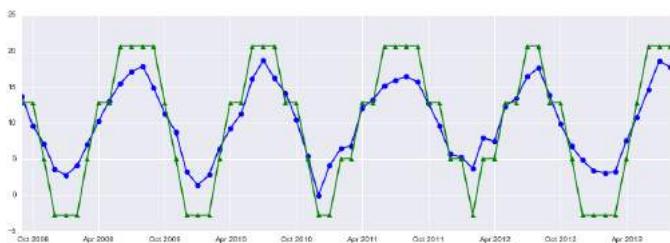


Figura 5. Discretização por frequência

Útil quando se procura uma distribuição equitativa dos dados, a discretização por frequências permite ultrapassar vários dos problemas levantados pela discretização em largura.

3) SAX: do inglês *Symbolic Aggregate Approximation*, este método é especialmente otimizado para a discretização de séries temporais, esperando-se então uma distribuição dos dados melhor orientada para ser utilizada como *input* para os algoritmos apresentados nas próximas secções.

O algoritmo divide-se em duas fases. A primeira, denominada *Piecewise Aggregate Approximation* (PAA), consiste em reduzir o tamanho do conjunto de dados, dividindo o tamanho original da série temporal (*n*) em *k* segmentos de tamanho semelhante, sendo o valor de cada uma dessas partes a média aritmética dos *n/k* elementos que lhe correspondem.

discretize.py

```
# Piecewise Aggregate Approximation (PAA)
def PAA (self, newSize):
    if newSize > self.dataSize:
        newSize=self.dataSize
    delta = self.dataSize / newSize
    PAA=[]
    for i in range( newSize ):
        m=0
        for j in range ( round(delta*i), round(delta*(i+1)) ):
            m += self.data[j]
        PAA.append(m / (round(delta*(i+1))-round(delta*i)) )
    return PAA
```

Este processo de agregação, implementado no excerto de código anterior, pode ser mais facilmente compreendido através da análise da Figura 6. Apesar de estar representado na figura todo o processo do algoritmo de discretização, esta fase é bastante notória, correspondendo à transformação da curva inicial num conjunto de apenas oito valores (representados pelos segmentos de reta).

Nos exemplos seguintes será usado um valor de *k* doze vezes inferior ao tamanho original da série, de forma a que cada valor do novo conjunto seja a média anual das temperaturas, permitindo assim uma remoção do ruído inerente às variações decorrentes durante as várias estações do ano.

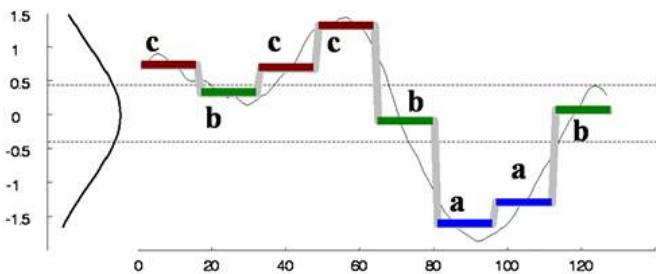


Figura 6. Visualização do método SAX [3]

A segunda fase do algoritmo implica agrupar os dados resultantes da fase anterior, de acordo com a probabilidade da distribuição normal dos dados. A Figura 6 descreve este passo através da divisão no eixos vertical em três símbolos/cores distintos (a/vermelho, b/verde e c/azul).

discretize.py

```
# Symbolic Aggregate Approximation (SAX)
def SAX(self, newSize):
    PAA = self.PAA(newSize)
    u = sum(PAA) / newSize
    o = math.sqrt( sum([(x-u)**2 for x in PAA]) / (newSize-1) )
    intervals=[]
    for i in range(1,self.alphabetSize):
        intervals.append( norm.ppf(i/self.alphabetSize, u, o) )
    intervals.append(self.max+1)
    return self.replace(PAA, changes)
```

O código anterior implementa o algoritmo descrito, começando por calcular a média e o desvio padrão do conjunto de dados, preenchendo finalmente a lista com os limites dos intervalos de discretização, para que lhe seja aplicado o método *replace*, de forma análoga aos outros métodos de discretização.

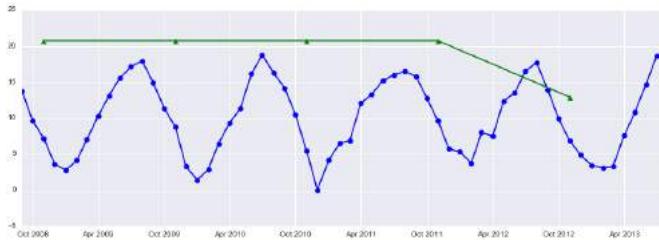


Figura 7. Discretização por agregação simbólica

Como se pode observar na Figura 7, o método SAX é indicado para quando se pretende reduzir a dimensão do conjunto de dados. Assim, é-nos agora possível visualizar os dados já discretizados (Figura 8), retirando informação pertinente que não poderiam ser extraídas da Figura 1. Este método é também o mais adequado dos três para a utilização em *data mining*, classificação e

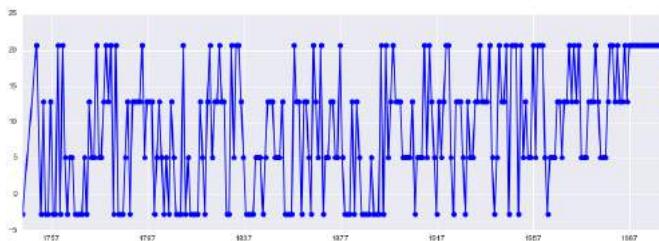


Figura 8. Registo de temperaturas 1743-2013 (SAX)

E. Relação entre cidades

Temos ao nosso dispor um variado conjunto de métodos direcionados à procura de semelhanças entre as várias sequências de temperaturas, todos eles com diversos potenciais.

Neste estágio, ambicionando obter a matriz com as distâncias entre todas as cidades, foi implementado o algoritmo *Needleman-Wunsch*, invertendo as pontuações de cada alinhamento para assim obter a distância entre as séries de temperaturas de cada par de cidades.

Contudo, esta tarefa revelou-se bastante dispendiosa em termos computacionais, uma vez que o algoritmo *per se* já apresenta um custo quadrático, sendo aplicado múltiplas vezes a sequências de dimensões relativamente elevadas. Obtemos então um custo $\theta(N^2 \times C^2)$, para C cidades com um conjunto de dados de tamanho N , que para o caso em questão já implicaria aproximadamente $1000^2 \times 15^2 = 225M$ iterações.

De modo a ultrapassar tal ambição computacional, o algoritmo de *Needleman-Wunsch* foi abandonado, tendo sido substituído pela correlação de *Pearson*, passando a executar em tempo linear para um par de cidades, levando a uma redução do número de iterações em cerca de três ordens de grandeza.

O código que implementa a correlação é bastante simples, uma vez que se serve de *APIs* já existentes para facilitar o processo. Em primeiro lugar, é necessário que os dados de entrada da função (*dataset*) correspondam a um objeto do tipo *Pandas.DataFrame*, ao qual é de imediato aplicado o método *corr()*. Uma vez obtida a matriz com os coeficientes de *Pearson*, esta vai ser transformada de forma a representar as distâncias entre as sequências (de zero a duas unidades), utilizando para tal o método *applymap()*, conforme o excerto de código apresentado em seguida.

pearson.py

```
# Calculates the series distance matrix
def getDistanceMatrix(dataset):
    corr = dataset.corr(method='pearson')
    dist = corr.applymap(lambda x: 1-x)
    return dist
```

Tendo já obtido a matriz de distâncias, é agora necessário desenvolver um método que permita a sua representação numa imagem, de forma a que os dados possam ser analisados mais facilmente. Para tal, foram utilizadas as várias *APIs* listadas em seguida.

pearson.py

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

O processo inicia-se pela criação de uma máscara que vai permitir representar apenas a matriz triangular inferior (com exceção da diagonal). De seguida, gera-se a figura onde vão

ser desenhadas as várias partes do gráfico, define-se o mapa de cores e finalmente colocam-se os valores no gráfico, através do método `seaborn.heatmap()`, como podemos verificar através do código apresentado em seguida.

——— `pearson.py` ———

```
# Stores the series distance matrix on a heatmap
def plotHeatmap(dist,filename):
    mask = np.zeros_like(dist, dtype=np.bool)
    mask[np.triu_indices_from(mask)] = True
    f, ax = plt.subplots(figsize=(9, 8))
    sns.heatmap(dist, mask=mask, cmap='winter_r',
                vmax=2, square=True, linewidths=.5, cbar_kws
                ={'shrink': .5}, ax=ax)
    f.savefig(filename)
```

Os métodos anteriores foram aplicados, de forma a representar graficamente a matriz de distâncias resultante da aplicação do métodos de discretização *SAX*, com um alfabeto de tamanho 4 e uma redução do número de elementos para 1500 na fase de agregação. A Figura 9 representa o resultado obtido.

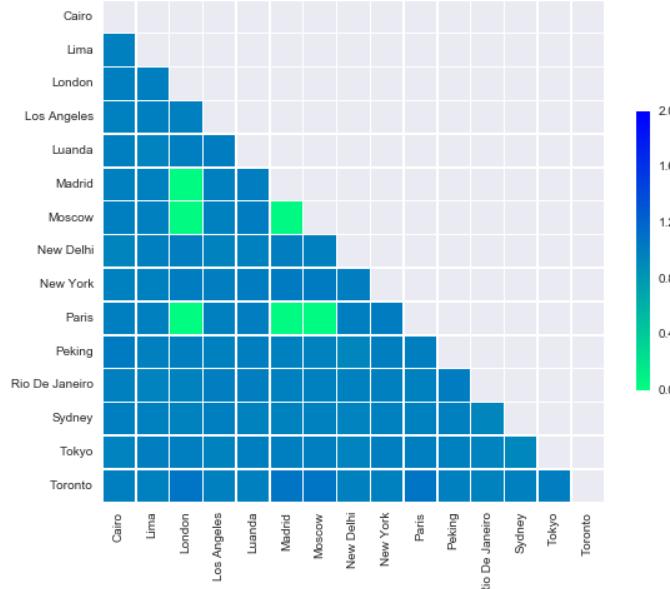


Figura 9. Matriz de distâncias - SAX(4,1500)

Uma breve análise da figura permite perceber que os vários países europeus se comportam de forma semelhante, não sendo possível tirar quaisquer conclusões para os restantes.

Com o objetivo de estabelecer um termo de comparação, repetiu-se o processo anterior, mas desta vez reduzindo o tamanho das sequências para 50. Tal como esperado, a redução drástica do tamanho dos dados aproximou bastante as sequências das várias cidades. No entanto, a imagem gerada não permite a extração de conclusões relevantes, como podemos verificar na Figura 10. [Outras variações estão disponíveis para consulta no Anexo D].

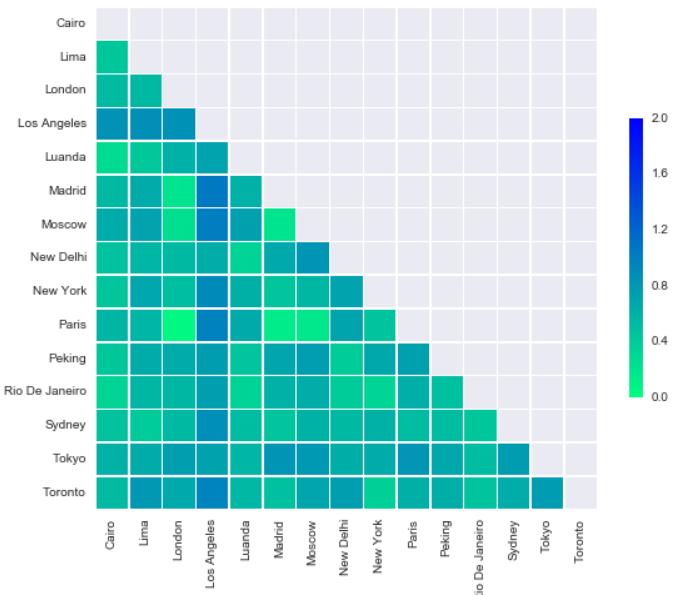


Figura 10. Matriz de distâncias - SAX(4,50)

F. Clustering

As matrizes de distância calculadas na fase anterior podem servir como dados de entrada na aplicação de vários algoritmos de segmentação (*clustering*), que vão permitir uma melhor interpretação dos resultados expostos nos gráficos anteriores.

Começaremos por efetuar uma hierarquização das cidades, uma vez que esta pode ser facilmente realizada a partir da matriz de distâncias.

De seguida, tínhamos como objetivo utilizar o algoritmo k-means para separar as várias cidades por diferentes grupos. No entanto, após alguma investigação, apercebemo-nos que este não pode ser utilizado para agrupar as cidades com base na variação da temperatura ao longo das séries temporais (declive da regressão linear), uma vez que a única métrica que temos disponível neste caso é a matriz de distâncias. Assim sendo, não é possível calcular a média do conjunto de pontos, ponto este que é crucial para a implementação do algoritmo.

A nossa solução para o problema anterior passa pela utilização do algoritmo k-medoids, que para além de ser aplicável ao nosso caso de estudo, é mais robusto do que o método k-means na presença de dados ruidosos, uma vez que os objetos selecionados durante a sua execução são menos influenciáveis por valores extremos do que a média calculada pelo k-means.

1) Hierárquico: este método segue o mesmo raciocínio utilizado para a obtenção de árvores filogenéticas. Utilizando a matriz de distâncias, como critério de segmentação do conjunto de cidades, estas vão sendo agrupadas até formar um diagrama composto por todas elas (por aglomeração).

Em seguida é apresentado um excerto de código, correspondendo a uma versão simplificada daquele que foi utilizado para gerar a representação gráfica das várias hierarquizações.

Como podemos verificar, este processo é implementado recorrendo aos métodos `linkage()` e `dendrogram()` do módulo `scipy.cluster`. Para além disso, recorremos mais uma vez ao módulo `matplotlib.pyplot`, para guardar como imagem os resultados obtidos.

cluster.py

```
# Plots the hierarchical cluster of a given D.M.
def hierarquical(distances, filename):
    cluster=sp.cluster.hierarchy.linkage(distances)
    plt.figure(figsize=(15, 17))
    sp.cluster.hierarchy.dendrogram(
        cluster,
        leaf_rotation=90.,
        color_threshold = 0.001
    )
    plt.savefig(filename)
```

Recorrendo ao código anterior, foram gerados vários dendogramas [presentes no Anexo E]. Em seguida apresentam-se dois exemplos, cuja diferença está apenas relacionada com o tamanho da série discretizada, correspondendo cada um deles à aplicação do algoritmo sobre uma das matrizes de distâncias anteriormente representadas. Na Figura 11 nota-se claramente a semelhança entre os vários países da Europa, facto que nos permite assumir que estes se comportam de forma semelhante, tendo todos o mesmo clima.

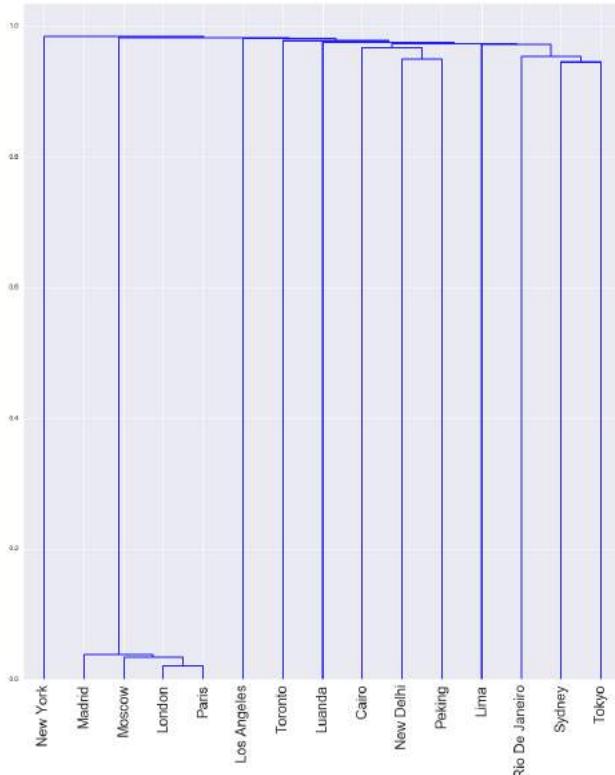


Figura 11. Agrupamento hierárquico, SAX(4,1500)

A Figura 12 foi calculado apenas com cinquenta pontos de temperatura por cidade. Uma análise mais cuidada das várias junções presentes no dendograma permitem perceber que um conjunto tão reduzidos de dados tende a distorcer a informação, agrupando cidades de climas totalmente distintos, que provavelmente estão a aquecer/arrefecer a ritmos diferentes.

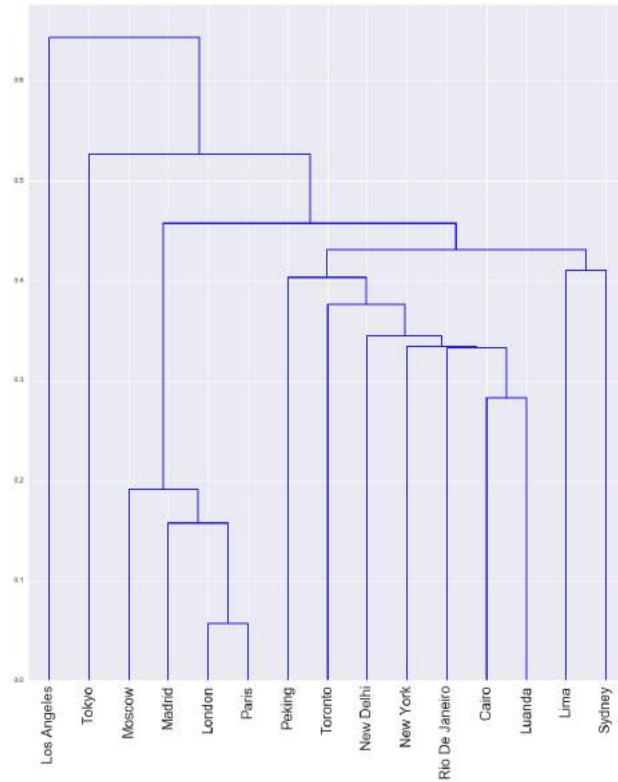


Figura 12. Agrupamento hierárquico, SAX(4,50)

2) *k-medoids*: a aplicação deste método foi efectuada recorrendo ao método `k_medoids()`, obtido *online* [4]. O método consiste na seleção aleatória de tantos grupos de elementos quantos *clusters* se pretendam obter, trocando elementos entre eles ao longo de várias iterações, num processo que visa otimizar as distâncias entre os elementos de cada *cluster*, reduzindo o máximo possível.

Apesar de este método ser adequando para conjuntos de dados de dimensões reduzidas, verificámos algumas variações nos resultados produzidos quando repetímos a execução do algoritmo utilizando as mesmas condições iniciais, facto este que pode estar relacionado com a proximidade dos valores contidos na matriz de distâncias.

Depois de separar as várias cidades por diferentes grupos, era necessário definir um método para a representação dos resultados obtidos. Assim, decidimos utilizar uma projeção cilíndrica equidistante do planeta terra, de forma a que traduzir diretamente do sistema de coordenadas fornecido (latitude e longitude), para um dado local da imagem, sobre o qual vamos desenhar um círculo com a cor do *cluster*.

Com o objetivo de efetuar essa conversão, a latitude (0° a 90° , Norte ou Sul) e a longitude (0° a 180° , Este ou Oeste), foram convertidas para uma percentagem. Para tal, os valores numéricos foram colocados numa escala contida no intervalo [0%,50%], sendo posteriormente subtraídos ao ponto médio (50%), no caso do Norte e Oeste, ou somados, no caso do Sul e Este. Para melhor clarificar este exemplo, segue-se o código utilizado para calcular a latitude de uma cidade.

cluster.py

```
# Scales a given latitude to the [0,1] interval
def latToPercent(lat):
    g = float( lat[:-1].strip() )
    p = g / (90*2)
    if "N" in lat:
        return 0.5 - p
    elif "S" in lat:
        return 0.5 + p
    return -1
```

Para agrupar todos os conceitos anteriores, foi desenvolvido o método `drawMap()`. Este método lê a imagem com o planisfério e coloca nela um conjunto de círculos correspondentes às várias cidades, cada um deles com a cor do grupo a que pertence.

cluster.py

```
# Plots the k-medoids cluster of a given D.M.
def drawMap(cluster, filename):
    color=["#0000ff", "#00ff00", "#ffff00", "#ff0000", "#00ffff"]
    img = plt.imread("../map.jpg", format="jpeg")
    fig,ax = plt.subplots(1)
    ax.set_aspect('equal')
    plt.axis('off')
    ax.imshow(img, interpolation='nearest')
    for k,v in cluster.items():
        for city in v:
            city = (
                lngToPercent(city[1])*img.shape[1],
                latToPercent(city[0])*img.shape[0]
            )
            circ = Circle(city,25,color=color[k])
            ax.add_patch(circ)
    plt.savefig(filename, dpi=250)
```

Recorrendo ao código anterior, bem como à matriz de distâncias calculada sobre o *dataset* com 1500 pontos, para um alfabeto de quatro letras, foi obtida a Figura 13 com o resultado da segmentação em quatro classes [ver Anexo F].

Como podemos verificar, também aqui a Europa está representada como tendo um comportamento idêntico para todas as cidades estudadas.

Podemos ainda levantar a hipótese de que a maior influência na variação das temperaturas é a latitude (ver grupos azul e vermelho), mas que a longitude também tem uma certa importância no processo (ver grupos amarelo e vermelho).

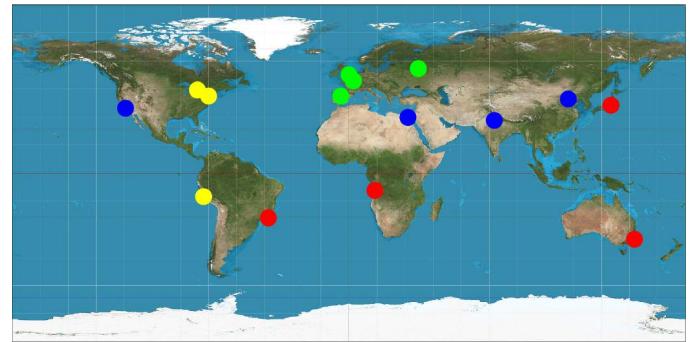


Figura 13. Resultado do k-medoids, SAX(4,1500), 4 grupos

G. Procura de padrões

A bioinformática dispõe de várias soluções algorítmicas para efetuar a procura de padrões (*motifs*) em sequências biológicas. Neste estágio do projeto vamos aplicar algumas dessas metodologias às séries de temperaturas obtidas na fase de discretização dos dados.

Porém, antes de implementar os métodos de procura, é preciso decidir de que forma os resultados vão ser representados, para que possam ser facilmente analisados. Neste sentido, decidimos implementar os gráficos conhecidos como *sequence logos*, uma vez que permitem a agregação das características de todas as sequências e são característicos da bioinformática.

Para tal, devido à complexidade dos gráficos em questão, foi utilizada a ferramenta weblogo[6], que recebe o conjunto das sequências em formato *fasta* e gera a sua representação gráfica, em formato *postscript*.

De seguida, foi utilizado o comando `convert`, disponibilizado pela ferramenta *ImageMagick*[7], que transforma o resultado anterior numa imagem em formato *PNG*.

Segue-se o script utilizado para gerar as representações gráficas de todas as sequências [Anexo G].

seqLogos.sh

```
#!/bin/bash
cd ../outputs/seqLogos
for file in fasta/*
do
    filename=$(basename "$file")
    filename=${filename%.}
    if [[ $filename == 4_* ]]
    then
        python3 ../../src/weblogo/weblogo -c classic
        -n 50 < fasta/$filename.fasta > eps/
        $filename.eps
    else
        python3 ../../src/weblogo/weblogo -c
        chemistry -n 50 < fasta/$filename.fasta
        > eps/$filename.eps
    fi
    convert -flatten -density 500 eps/$filename.eps
    png/$filename.png
done
```

A ferramenta utilizada permite a parametrização de várias características, como o número de elementos por linha do gráfico, o esquema de cores utilizado, as posições inicial e final, entre outras.

A Figura 14 representa a aplicação do algoritmo para linhas de 25 elementos, utilizando a totalidade de uma sequência de tamanho 50, com o esquema clássico de cores.

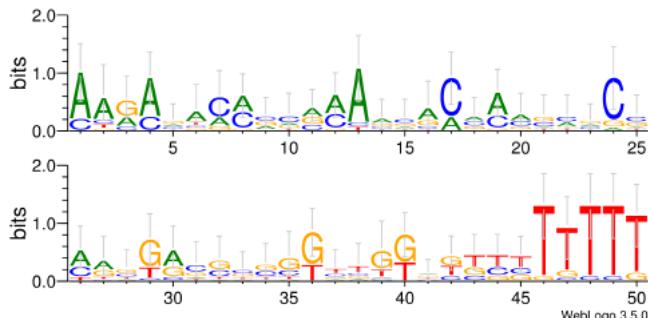


Figura 14. Sequence logo, SAX(4,50)

Contudo, podemos perceber que uma sequência comum, facilmente contendo várias centenas (ou mesmo milhares) de elementos, poderia gerar gráficos bastante grandes e, portanto, de difícil interpretação.

Assim, decidimos criar uma representação mais abstrata da matriz PWM, que pudesse ser utilizada para procurar as zonas de maior interesse, servindo de índice para a representação anterior.

Para tal, desenvolvemos um gráfico em que o eixo horizontal representa a posição na sequência e o eixo vertical as percentagens de cada letra do alfabeto na posição respetiva, de acordo com o código apresentado em seguida.

A chromatogram showing four distinct peaks labeled A, C, G, and T, each enclosed in a blue oval. Below the chromatogram, a legend identifies the colors: A (yellow), C (orange), G (red), and T (dark red).

Figura 15. Representação da matriz PWM, SAX(4,50)

A Figura 15 corresponde à aplicação do código anterior, recorrendo aos mesmos dados de entrada utilizados para gerar a Figura 14. Como podemos ver nas anotações acrescentadas, são identificáveis algumas zonas idênticas na maioria das sequências.

Quando, por exemplo, se discretizam as sequências de forma a que cada uma tenha 500 caracteres de comprimento, podemos obter como resultado a Figura 16, que compõe numa área bastante reduzida um gráfico que ocuparia a totalidade da página se fosse representado como um *sequence logo* [Anexo H, Anexo I].

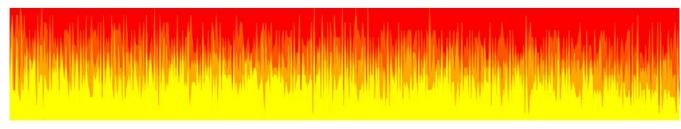


Figura 16. Representação da matriz PWM, SAX(4,500)

Estando já munidos de todas as ferramentas necessárias à representação adequada de motifs, implementámos o algoritmo de procura. Este recebe como parâmetros um dicionário com as várias sequências, o tamanho (W) das motifs a procurar e a percentagem mínima (P) de sequências que têm de conter as motifs desejadas.

Foram implementadas duas versões distintas do algoritmo, sendo que apenas uma linha de código difere entre elas. A primeira (versão comentada) obriga a que a motif só seja considerada no caso de estar contida nas mesmas posições em cada uma das sequências, permitindo que sejam aplicados os métodos de visualização anteriores para melhor analisar as sequências que, apesar de não conterem a motif, podem ou não ter a maioria dos seus caracteres.

A segunda versão (não comentada) permite que sejam consideradas motífs quaisquer subsequências que respeitem os valores W e P. A sua aplicação pode ser útil, por exemplo, para verificar se alguma cidade está a seguir as variações de outra(s).

Apresenta-se em seguida o código correspondente à implementação de ambos os algoritmos.

```
motifs.py
```

```
#Plots the PWM matrix representantion
def plotAreaChart(counts, filename):
    df = pd.DataFrame.from_dict (
        counts,
        orient='columns',
        dtype=None
    )
    ax = df.plot (
        kind='area',
        figsize=(30,5),
        stacked=True,
        colormap='autumn_r'
    )
    ax.legend (
        loc='upper_center',
        bbox_to_anchor=(0.5, -0.05),
        fancybox=True,
        shadow=True,
        ncol=20,
        prop={'size':14}
    )
    fig = ax.get_figure()
    fig.savefig(filename)
```

 motifs.py

```
def findMotifs(data, W, P):
    if P>1 or P<0 or W<1: return []
    seqs=clone(data)
    num_seqs = len(seqs)
    res=[]
    while len(seqs) >= num_seqs*P:
        city, seq_0 = seqs.popitem()
        for i in range (len(seq_0) - W + 1):
            count = 0
            motif = "".join(seq_0 [i:i+W])
            for k,seq in seqs.items():
                #if motif == seq[i:i+W]:
                if motif in "".join(seq):
                    count += 1
            if count/(num_seqs-1) >= P:
                if motif not in res:
                    res.append(motif)
                break
    return res
```

Utilizando o algoritmo anterior para procurar motifs de tamanho 5, contidas em pelo menos 50% das sequências num *dataset* produzido através da função SAX(4,100), foram encontradas as seguintes motifs: **TTGTT**; **TTTTT**. É interessante verificar que as únicas cadeias que verificam as condições acima indicadas correspondem a valores de temperaturas no nível mais elevado.

IV. CONCLUSÃO

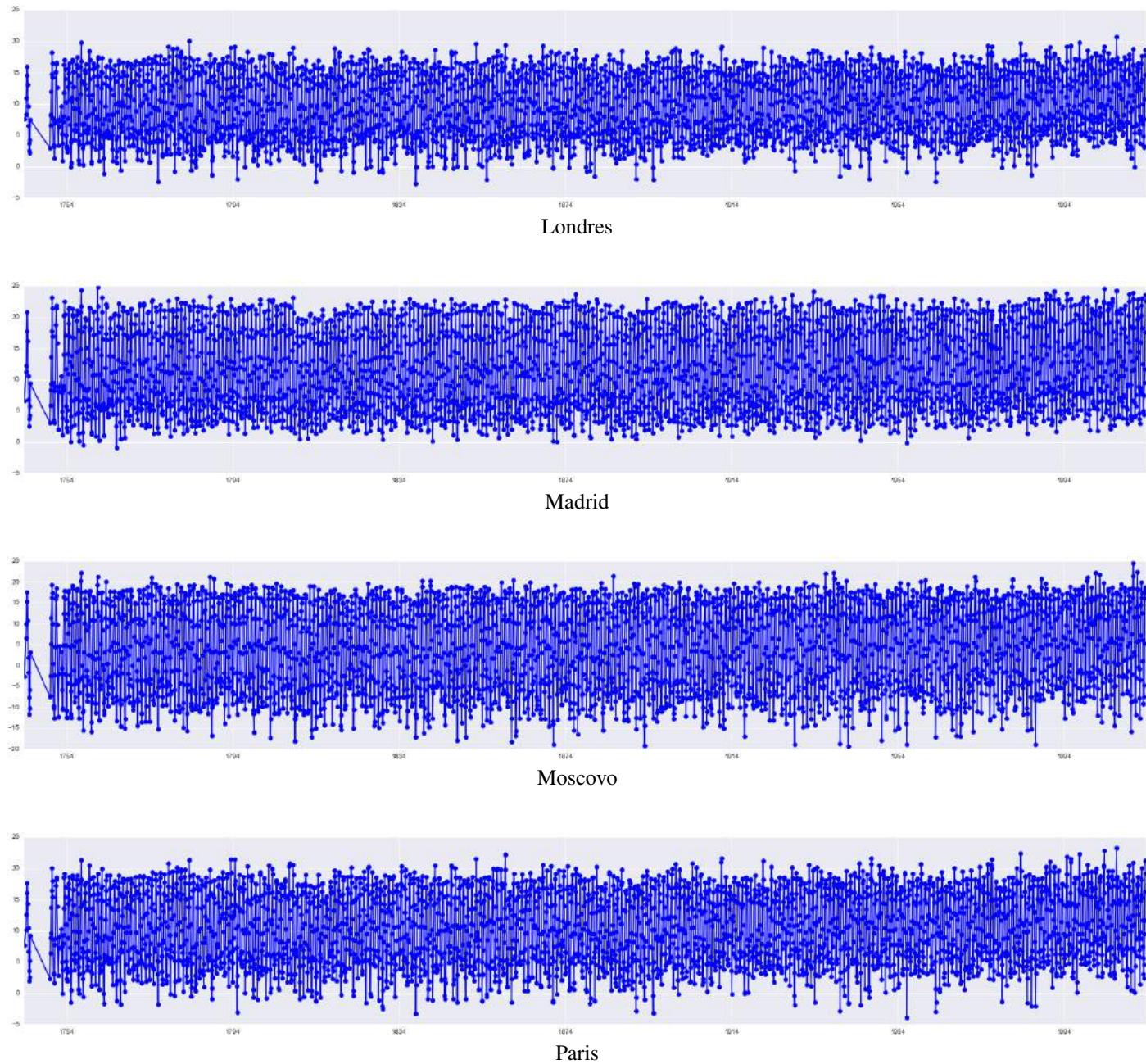
A análise de dados é uma área relativamente vasta, com possibilidade para explorar múltiplas metodologias para a resolução dos problemas apresentados. Neste trabalho foram utilizados algoritmos bioinformáticos para elaborar um pequeno estudo sobre uma base de dados de temperaturas, tendo já sido apresentadas algumas conclusões. Acreditamos que um estudo mais exaustivo neste sentido permitiria um melhor ajuste dos algoritmos bioinformáticos para a análise de dados não biológicos, levando à extração de conclusões relevantes que permitissem validar, questionar ou mesmo refutar resultados obtidos através de aproximações mais comuns.

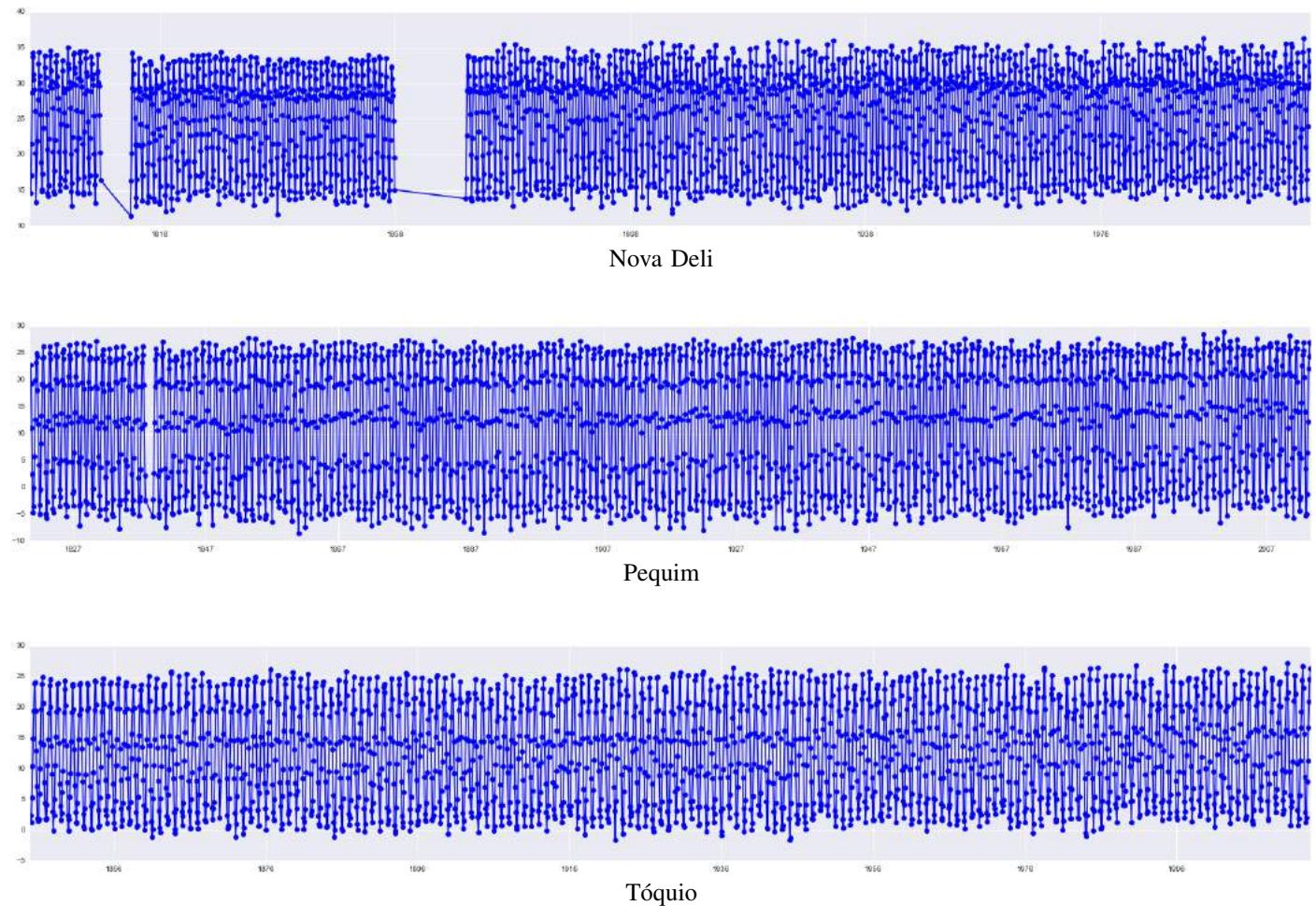
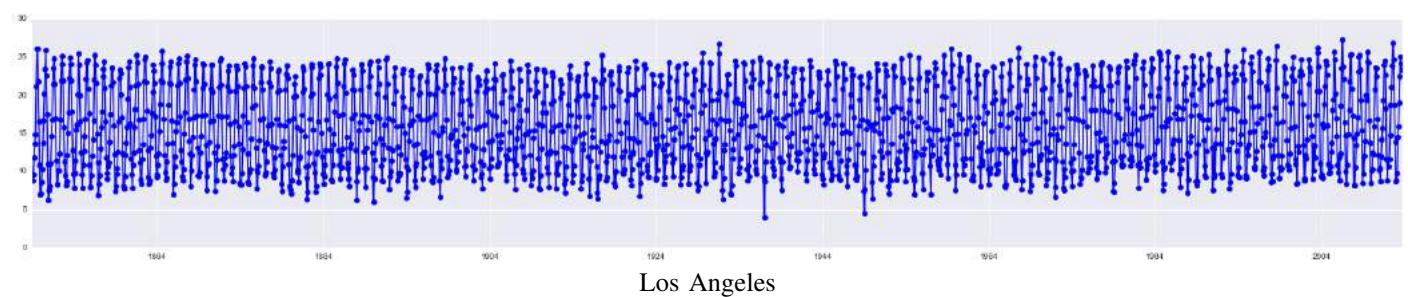
REFERÊNCIAS

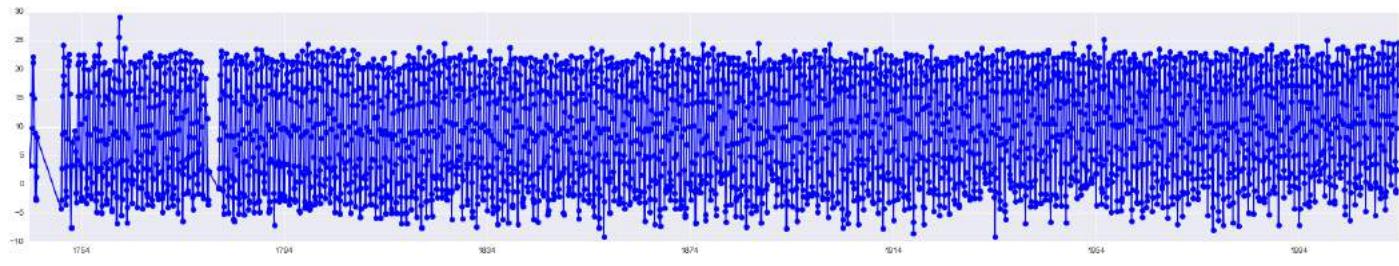
- [1] P. Chaudhari, D. P. Rana, R. G. Mehta, N. J. Mistry, M. M. Raghuwanshi, *Discretization of Temporal Data: A Survey*
- [2] Kaggle, *Climate Change: Earth Surface Temperature Data* [Online] Disponível em: www.kaggle.com/berkeleyearth/climate-change-earth-surface-temperature-data
- [3] Djamel Yagoubi, *Time series discretization with SAX* [Online] Disponível em: gite.lirmm.fr/yagoubi/SAX/blob/73635fc6c595d191dde538a2ea2f89e55c212eb9/src/resources/sax.png
- [4] Bauckhage C. Numpy/scipy Recipes for Data Science: k-Medoids Clustering[R]. Technical Report, University of Bonn, 2015.
- [5] Wikipédia, *Equirectangular projection* [Online] Disponível em: en.wikipedia.org/wiki/Equirectangular_projection
- [6] WebLogo [Online] Disponível em: weblogo.threplusone.com
- [7] ImageMagick® [Online] Disponível em: wwwimagemagick.org

APÊNDICE A
SÉRIES DE TEMPERATURAS SELECIONADAS (SEM ALTERAÇÕES)

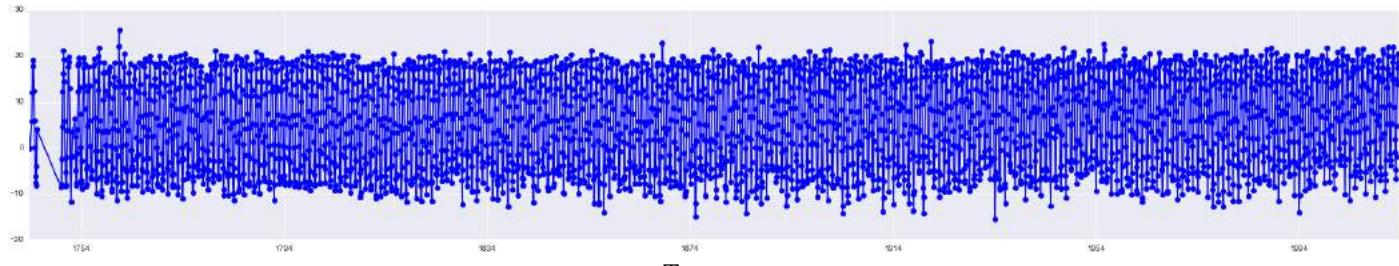
A. Europa



B. Ásia*C. América do Norte*

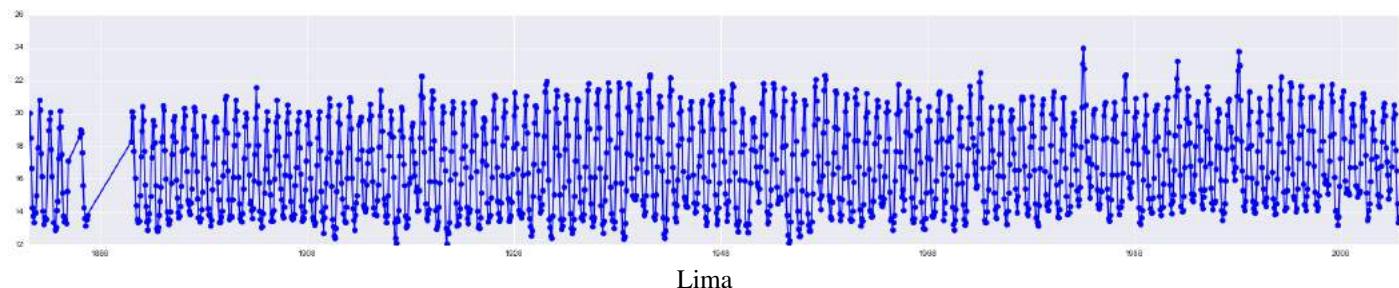


Nova Iorque

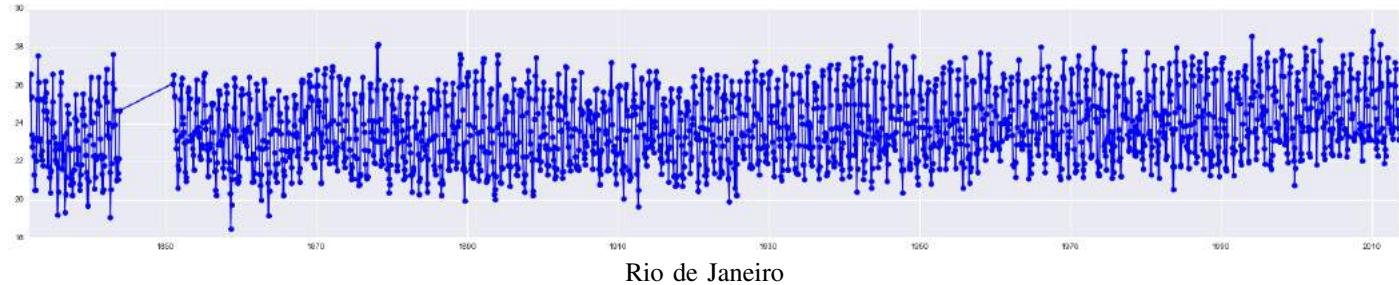


Toronto

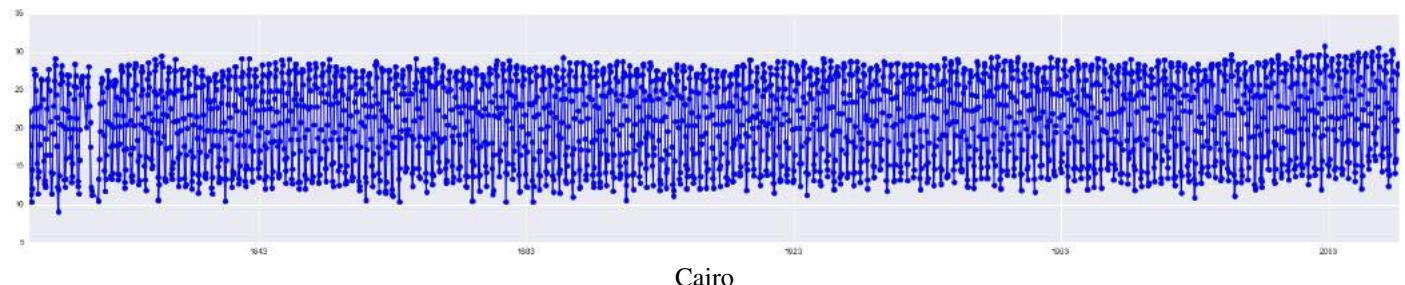
D. América do Sul



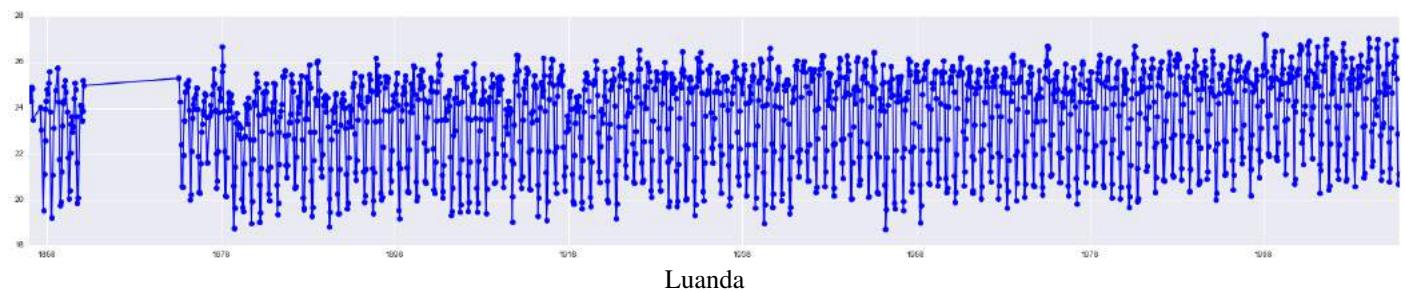
Lima



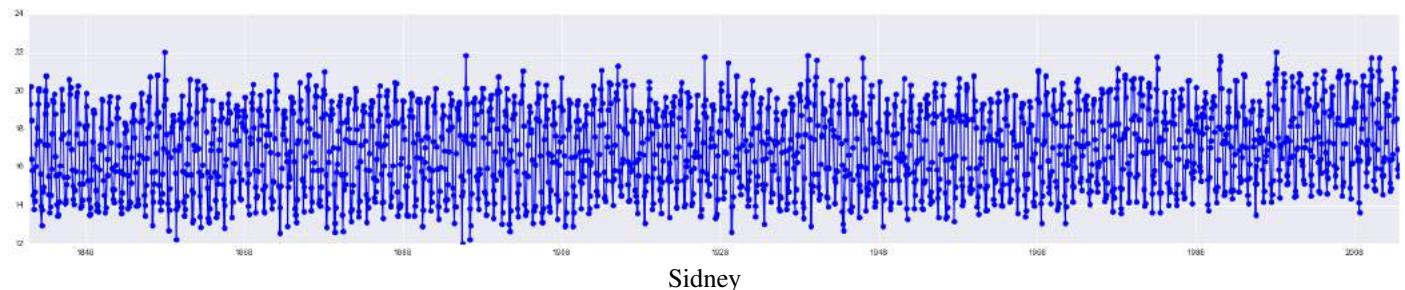
Rio de Janeiro

E. África

Cairo



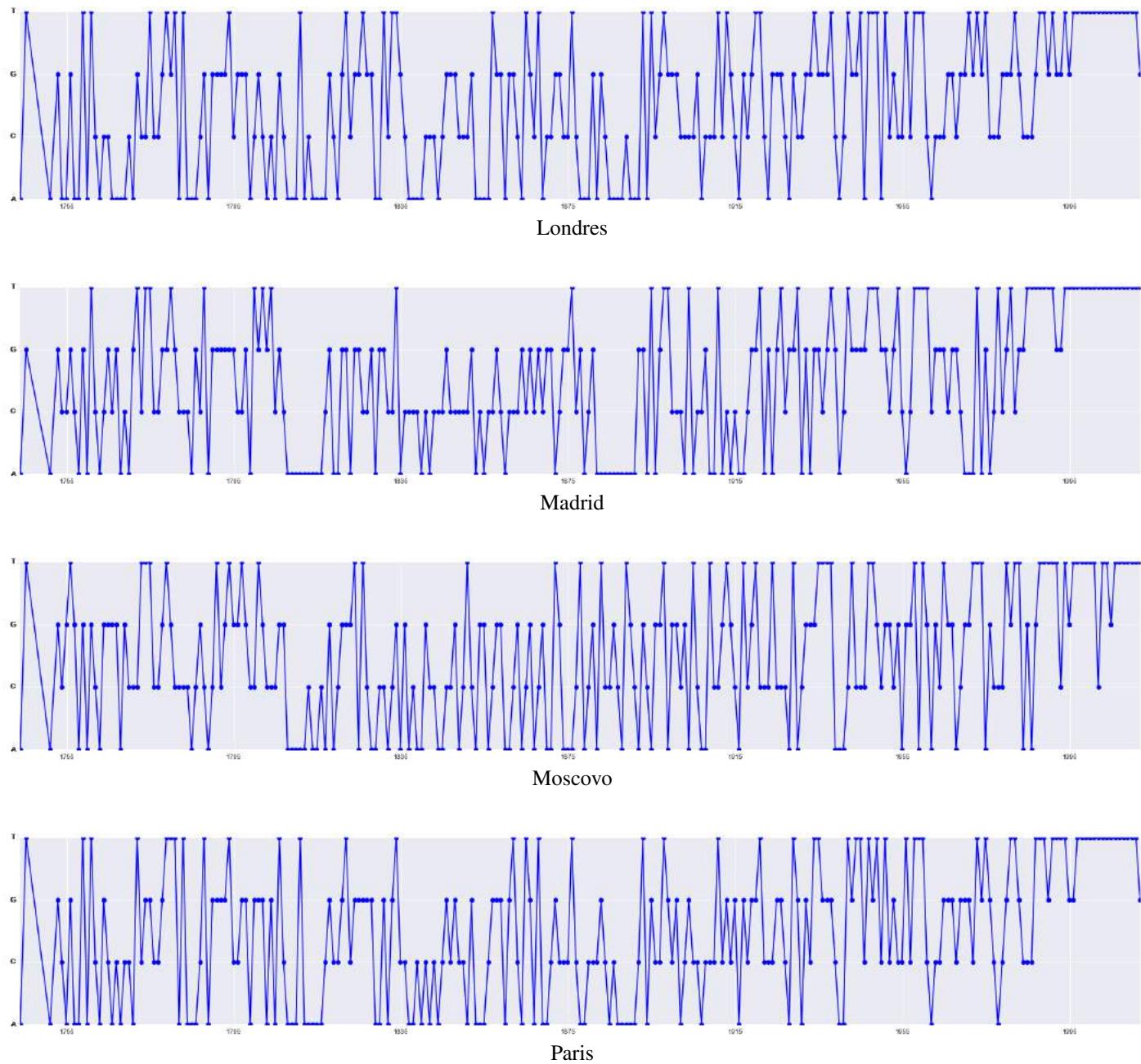
Luanda

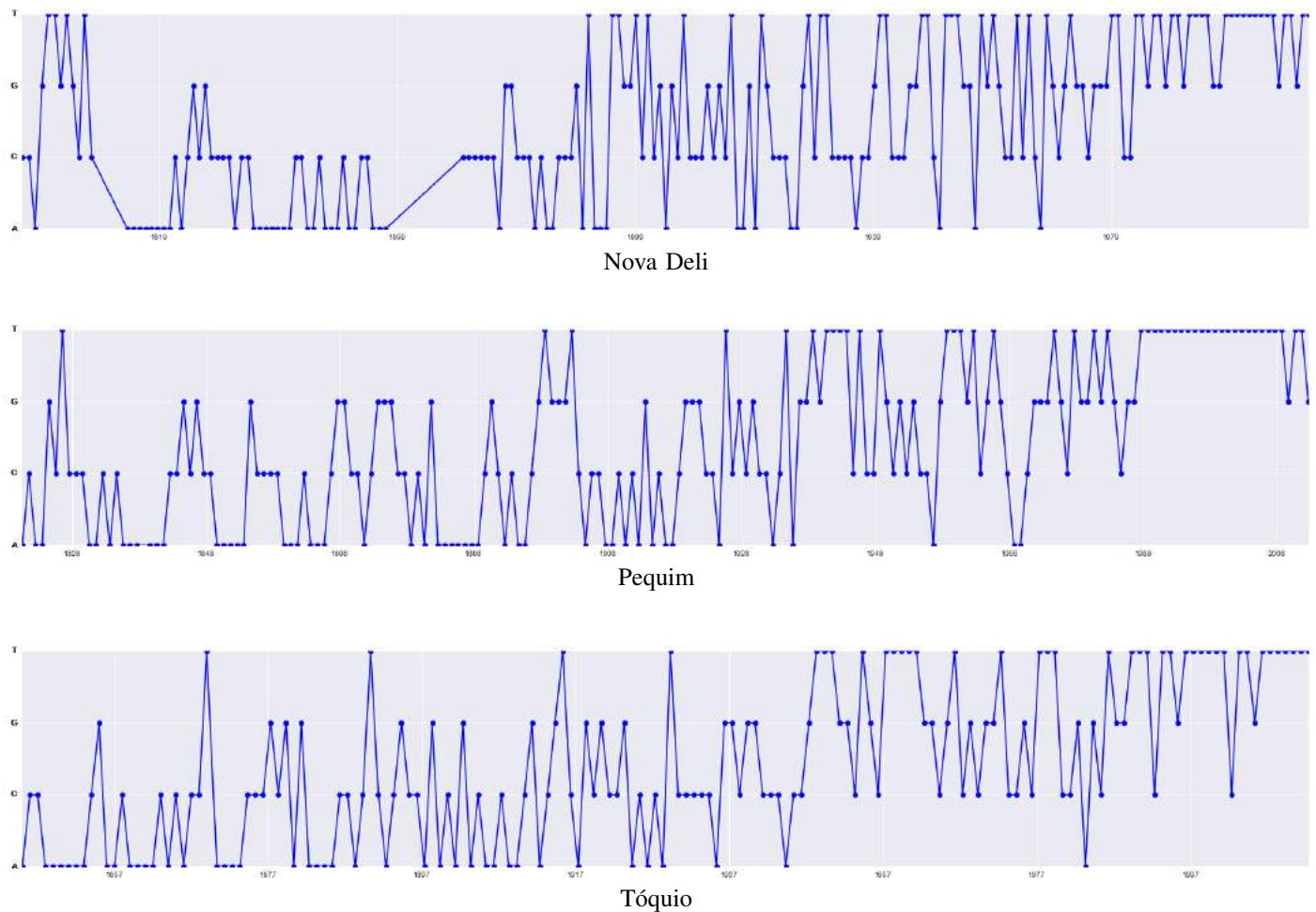
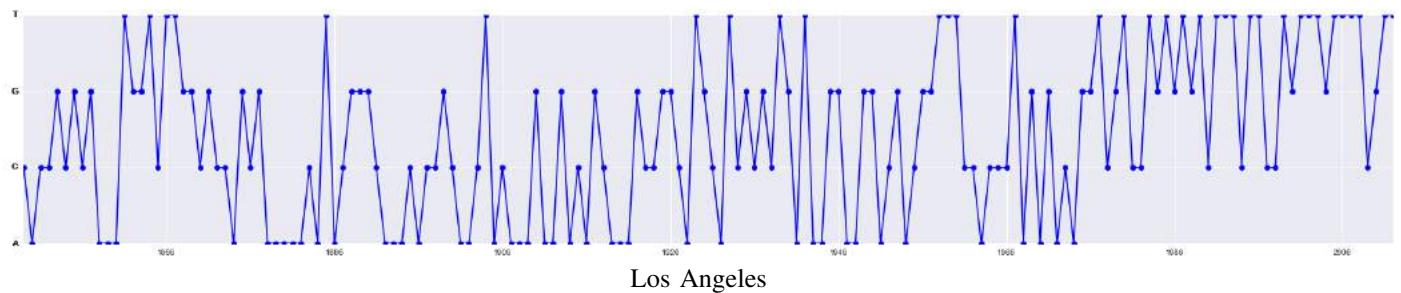
F. Oceânia

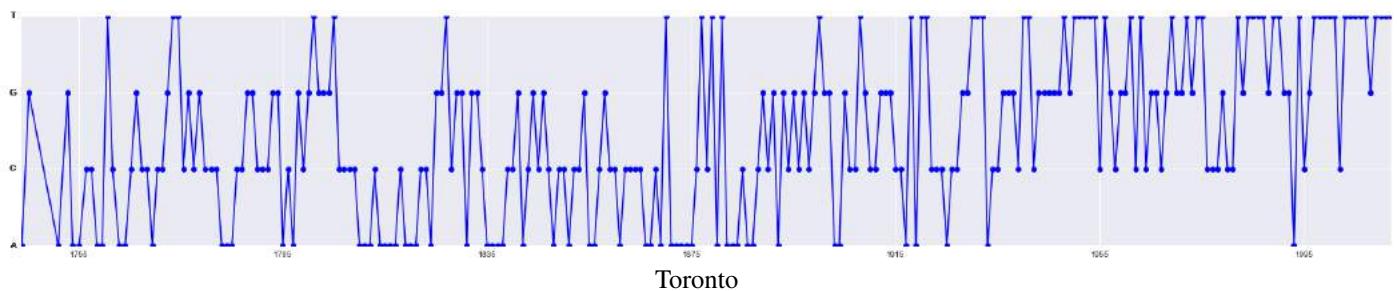
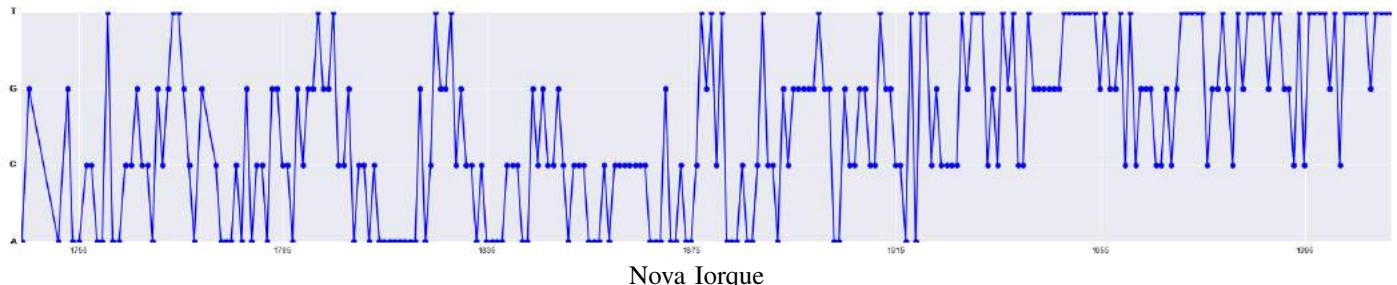
Sidney

APÊNDICE B
SÉRIES DE TEMPERATURAS SELECIONADAS (SAX COM ALFABETO DO ADN, 1 PONTO POR ANO)

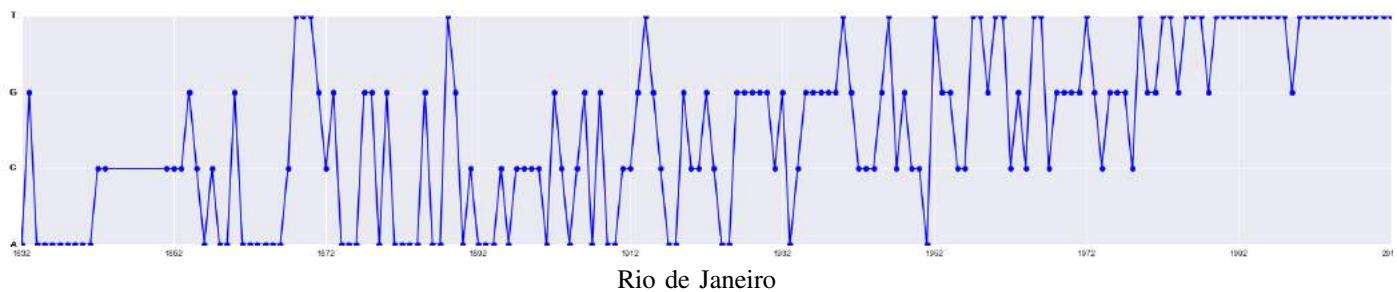
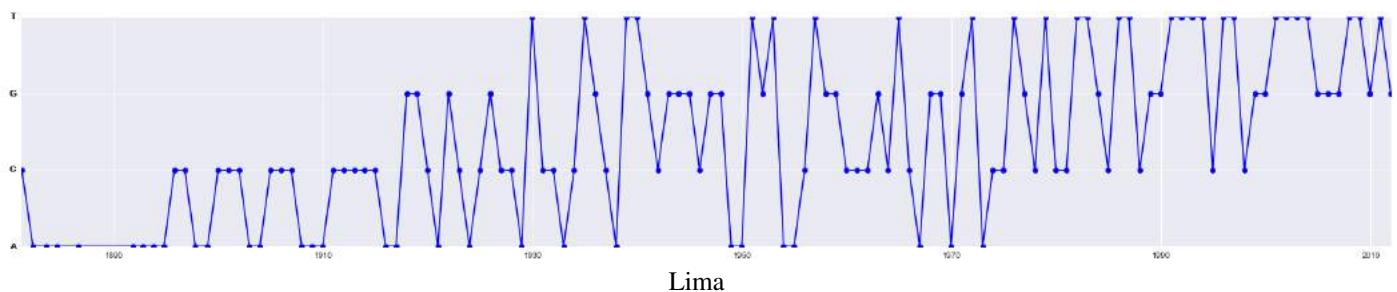
A. Europa

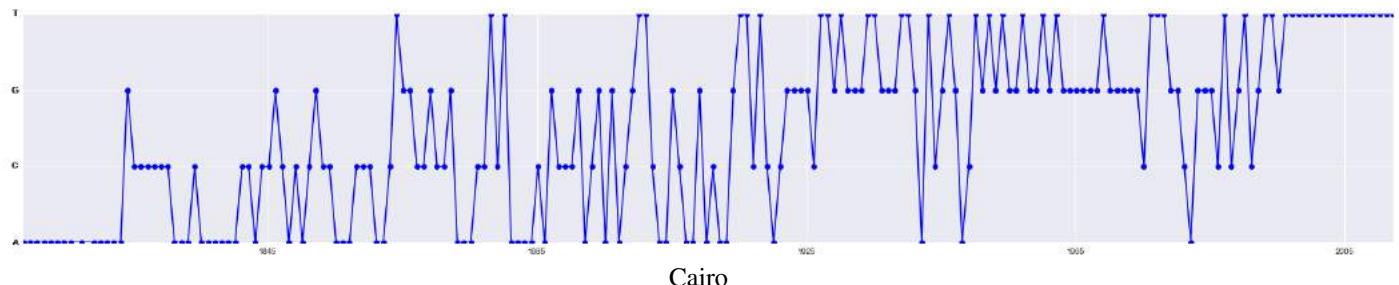


B. Ásia*C. América do Norte*

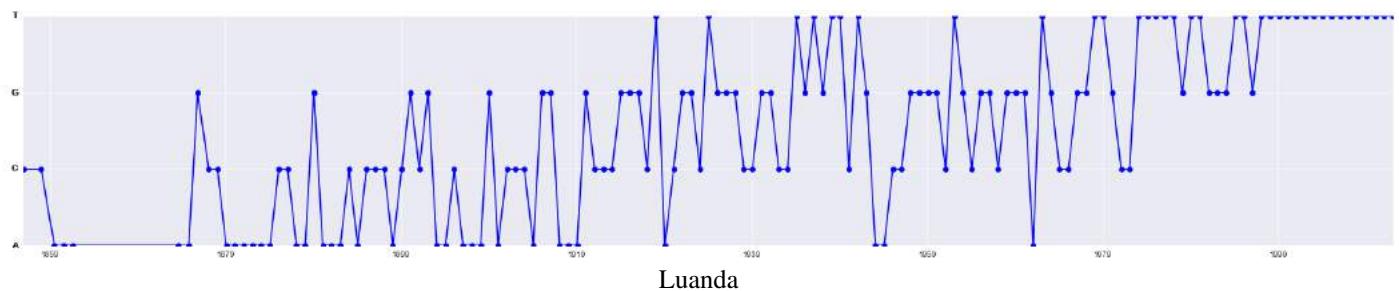


D. América do Sul

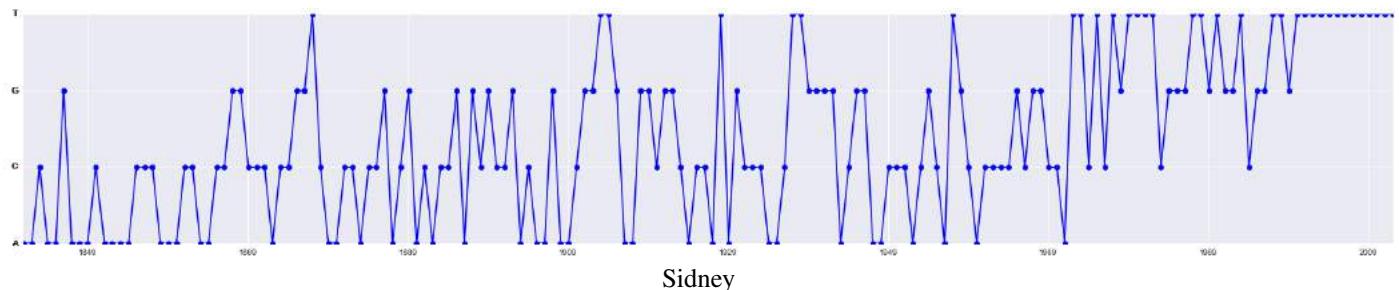


E. África

Cairo



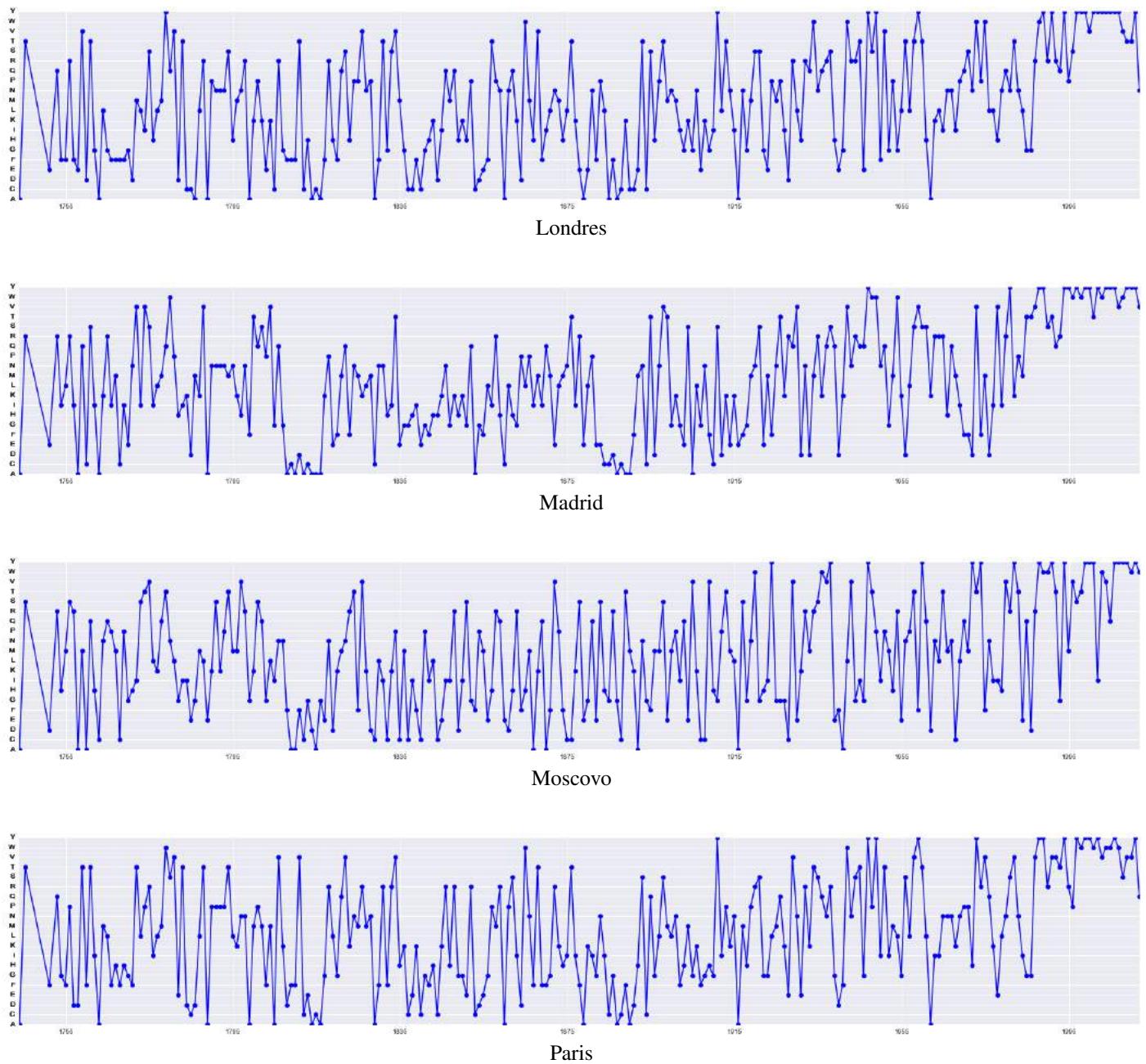
Luanda

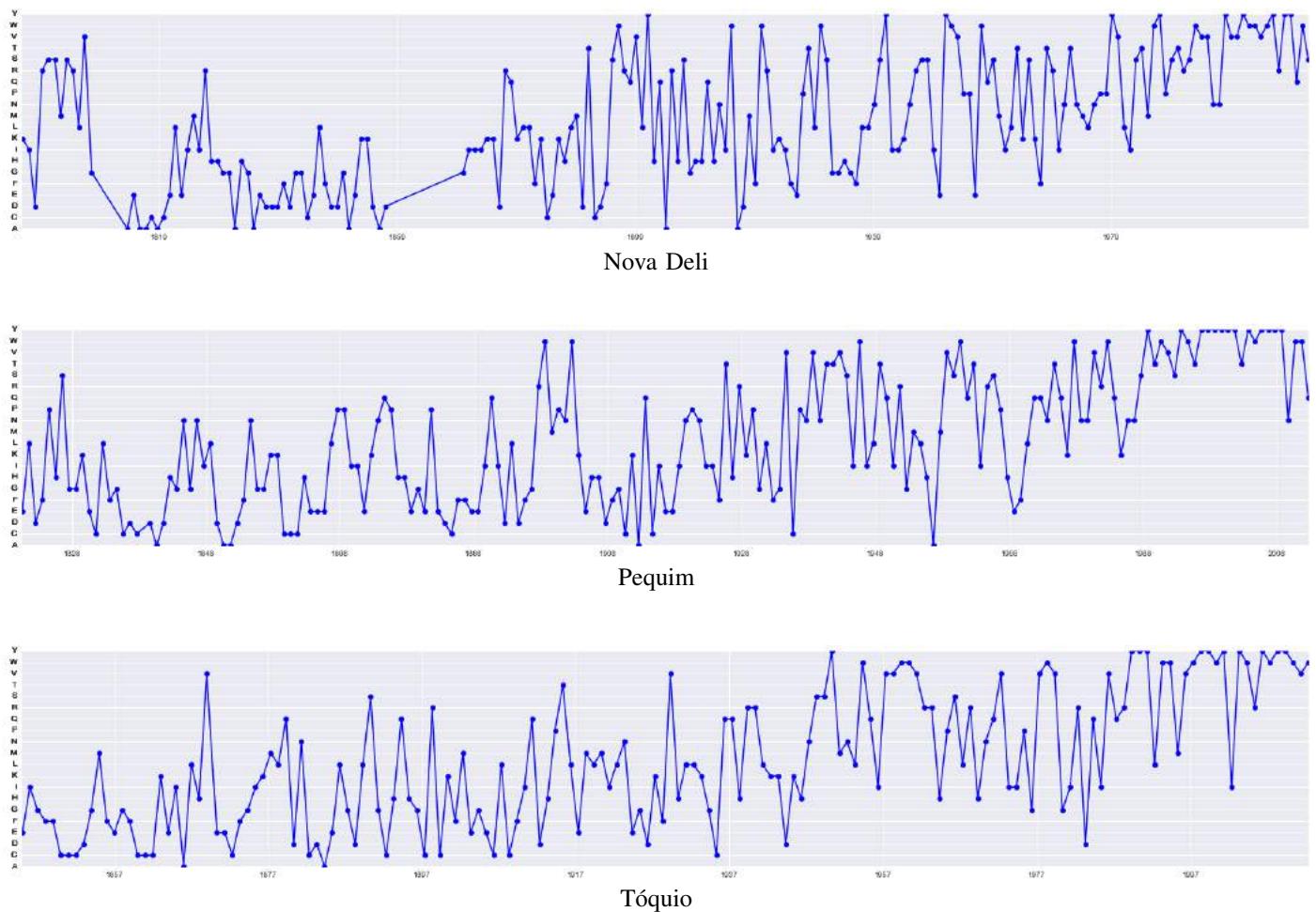
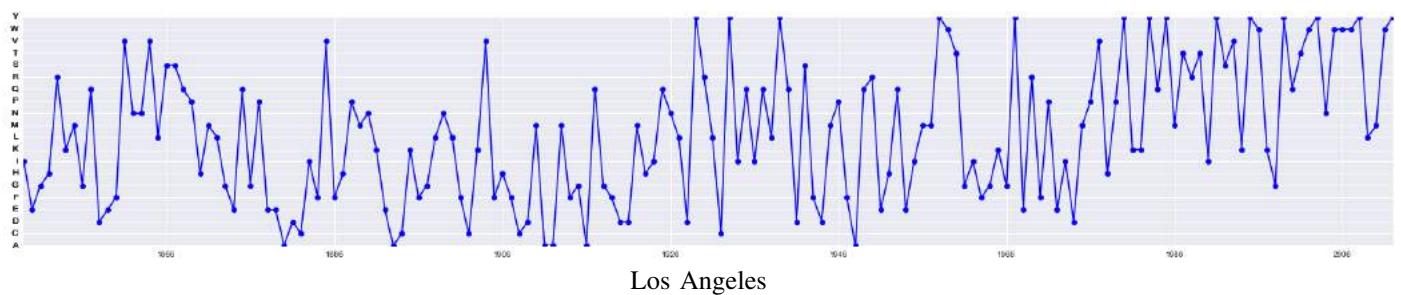
F. Oceânia

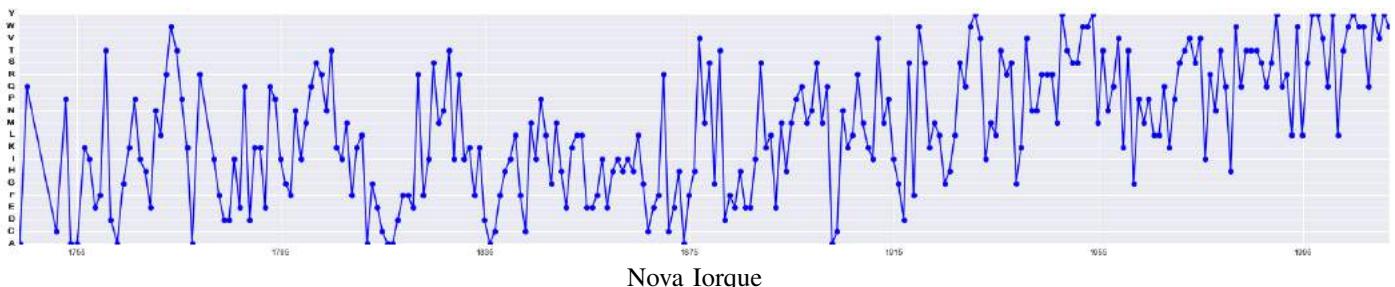
Sidney

APÊNDICE C
SÉRIES DE TEMPERATURAS SELECIONADAS (SAX COM ALFABETO DAS PROTEÍNAS, 1 PONTO POR ANO)

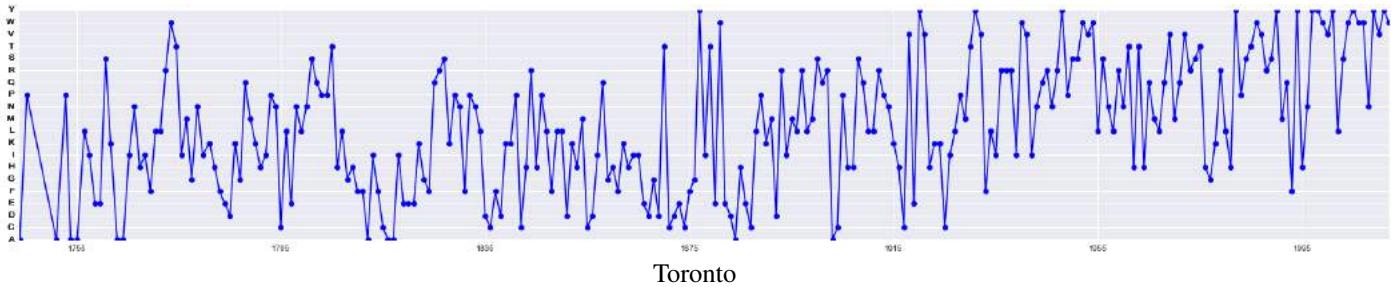
A. Europa



B. Ásia*C. América do Norte*

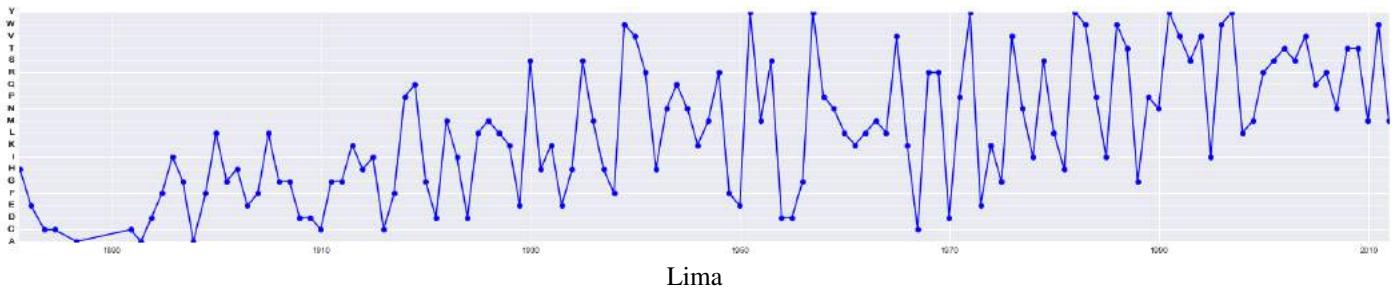


Nova Iorque

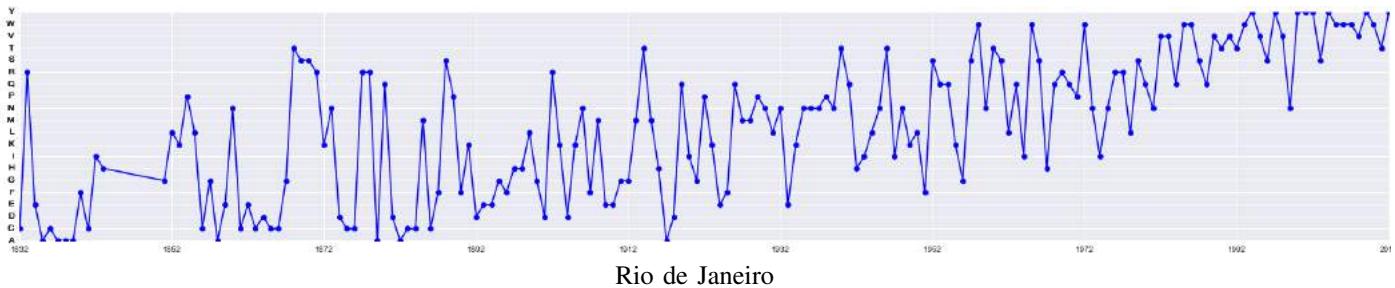


Toronto

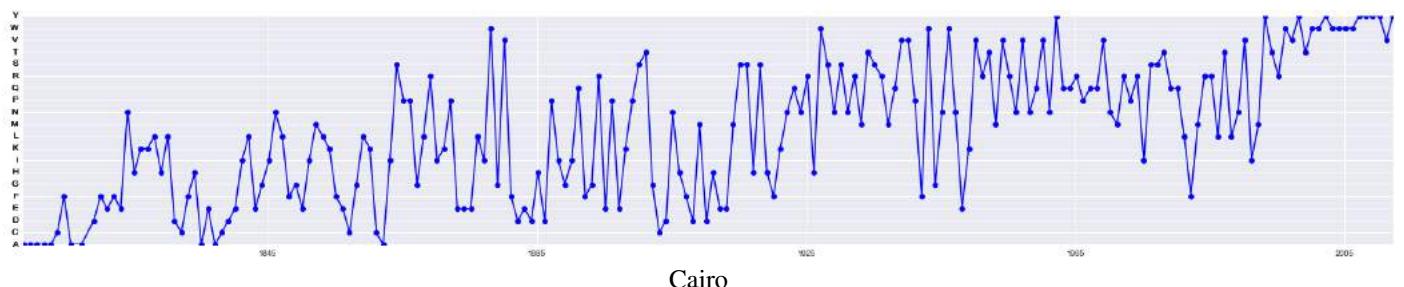
D. América do Sul



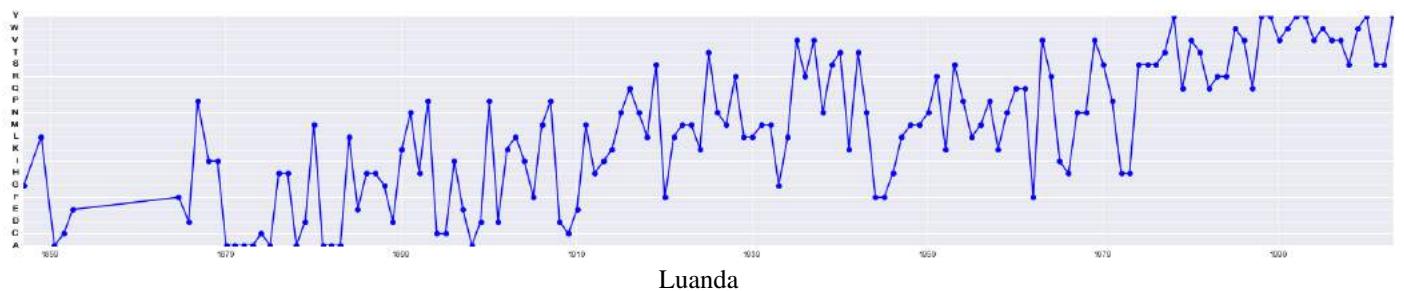
Lima



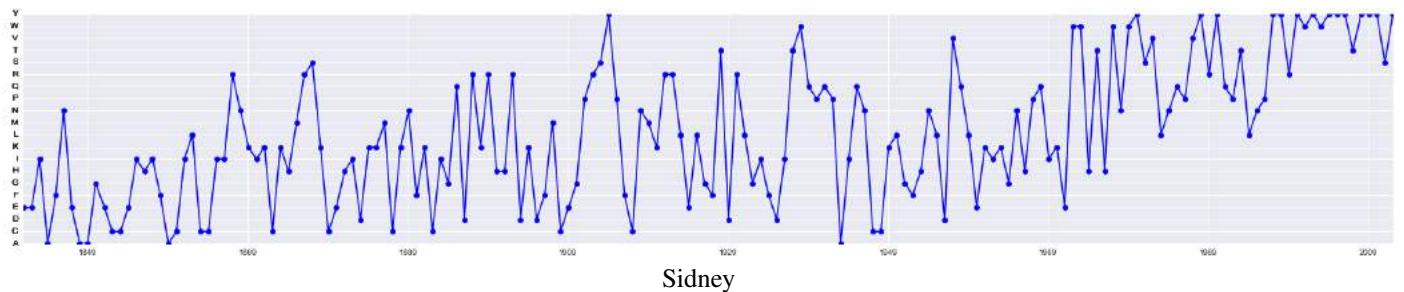
Rio de Janeiro

E. África

Cairo

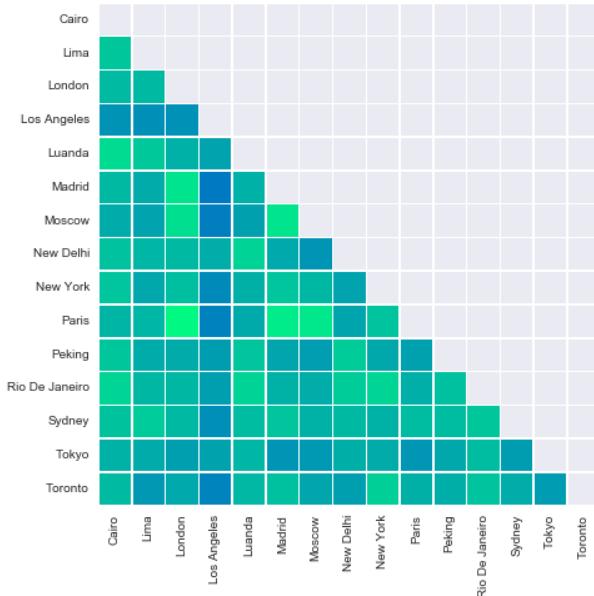


Luanda

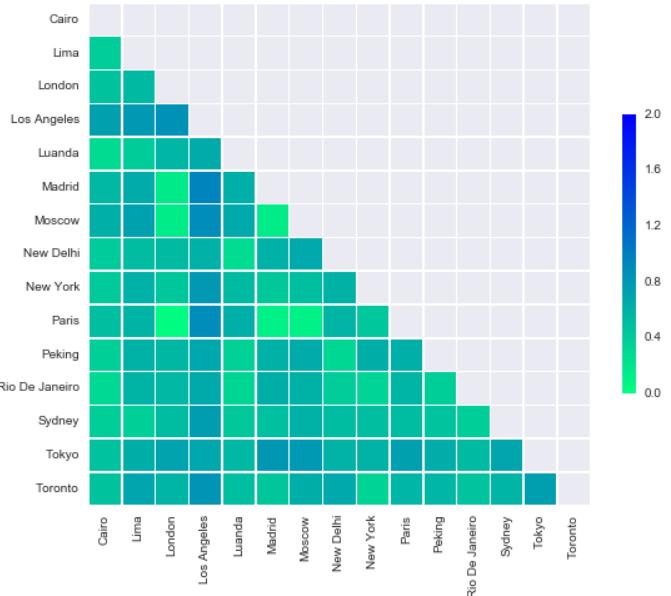
F. Oceânia

Sidney

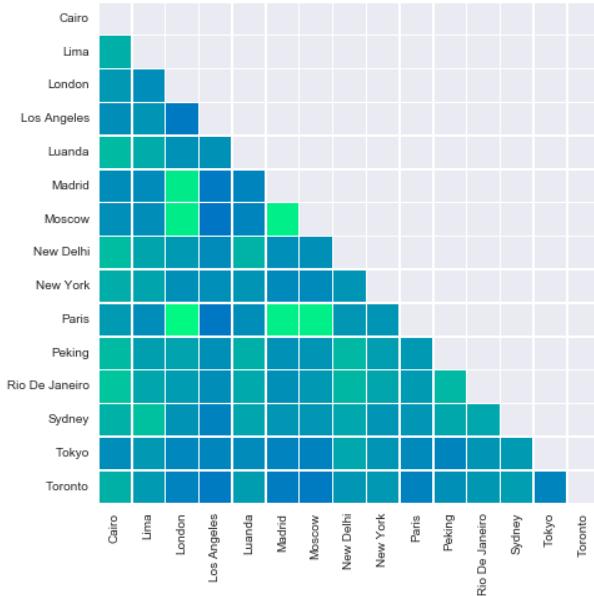
APÊNDICE D
MATRIZES DE DISTÂNCIAS DE ACORDO COM AS DIMENSÕES DO ALFABETO E SÉRIE DE TEMPERATURAS

A. Alfabeto Do ADN (4 elementos)

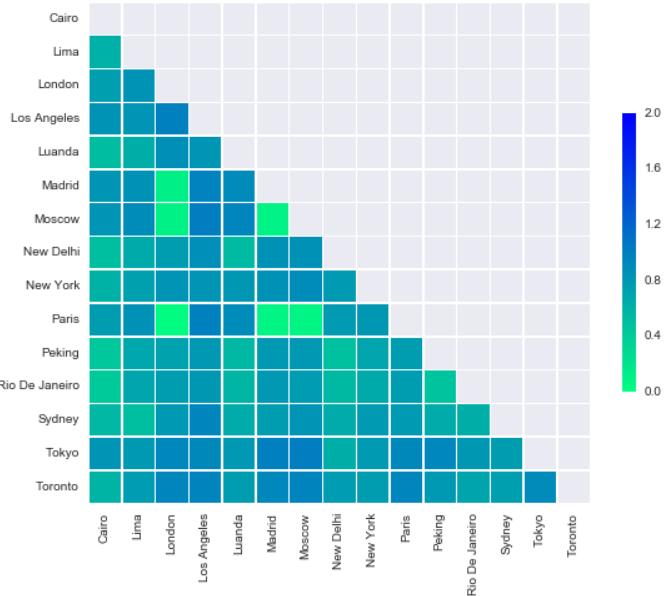
ADN, PAA tamanho 50

B. Alfabeto das proteínas (20 elementos)

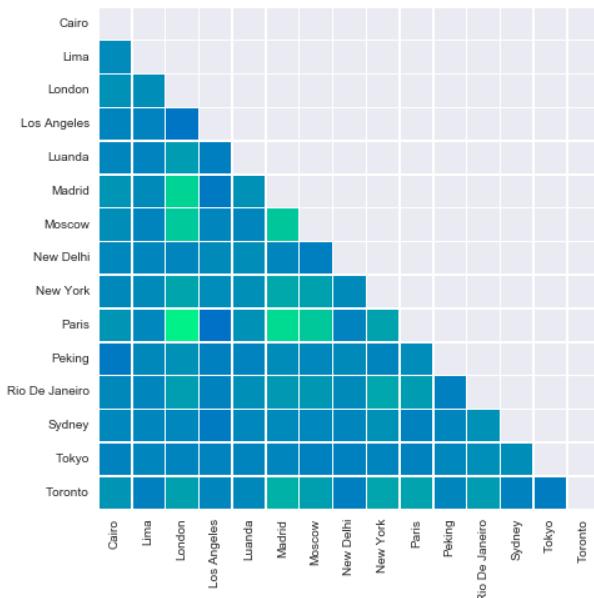
Proteína, PAA tamanho 50



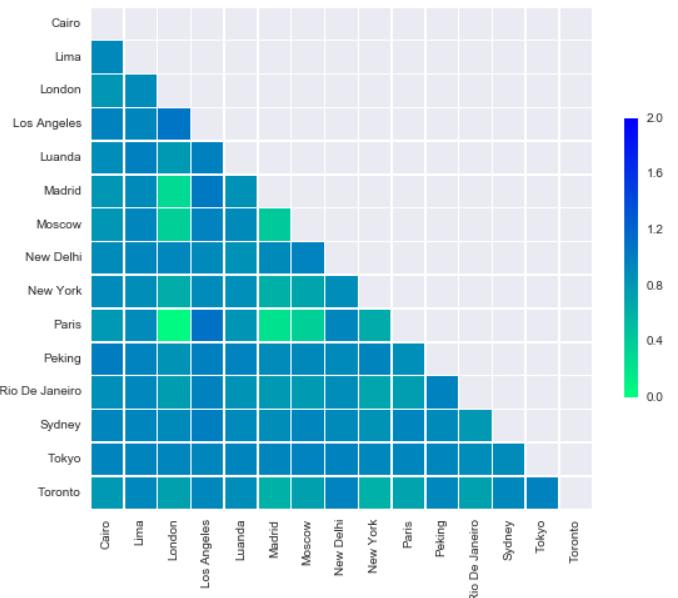
ADN, PAA tamanho 100



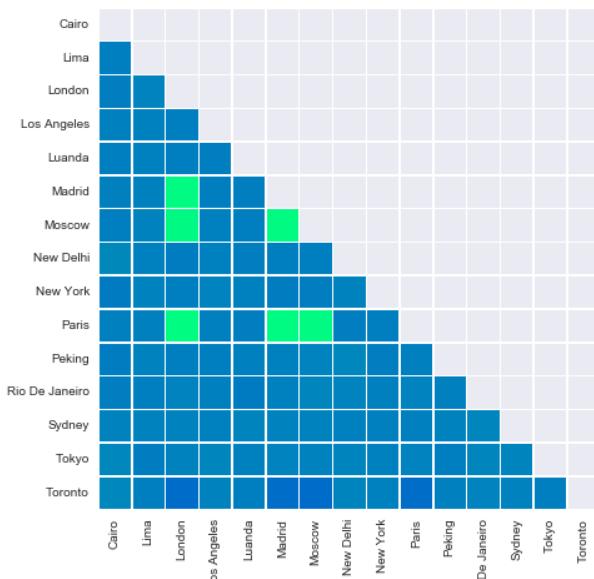
Proteína, PAA tamanho 100



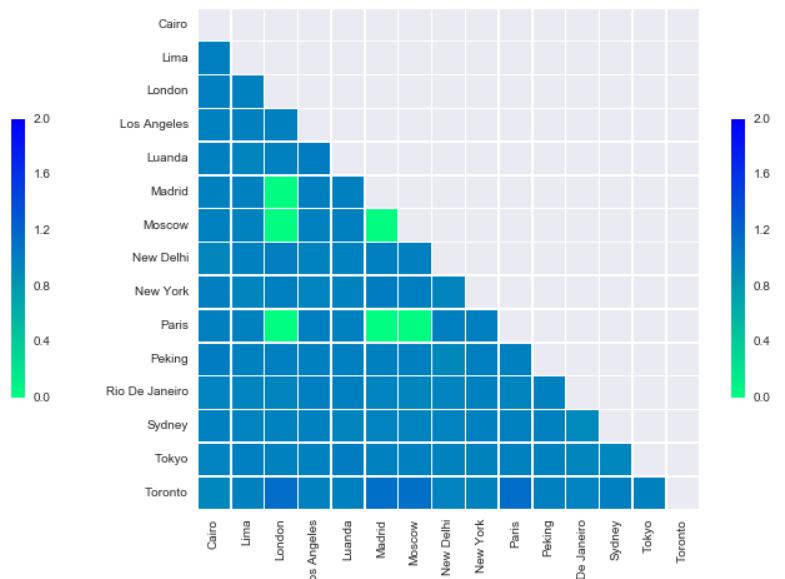
ADN, PAA tamanho 250



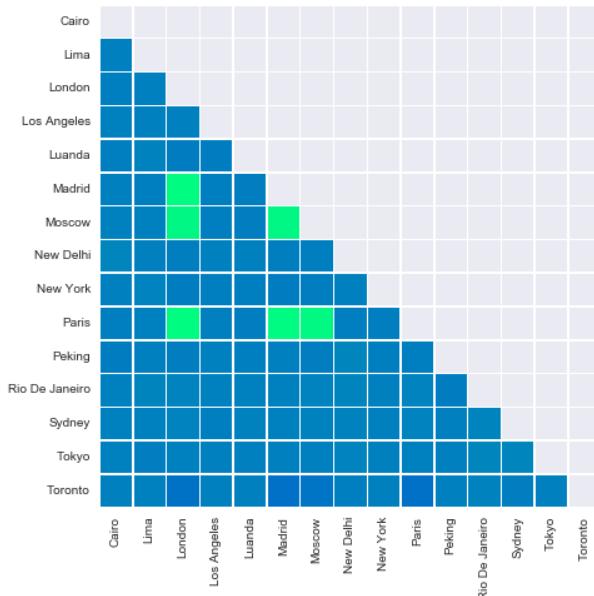
Proteína, PAA tamanho 250



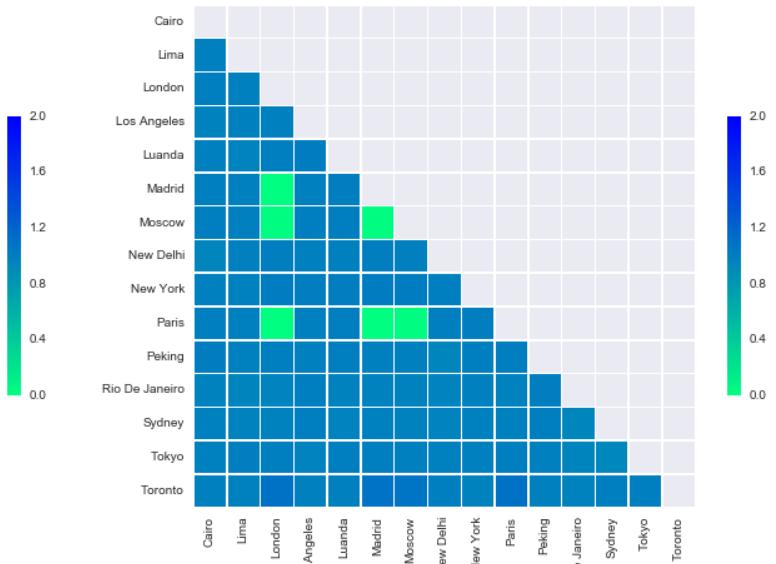
ADN, PAA tamanho 500



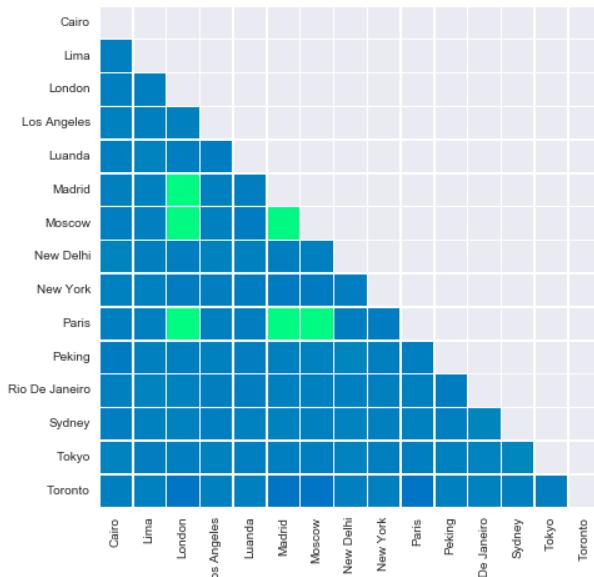
Proteína, PAA tamanho 500



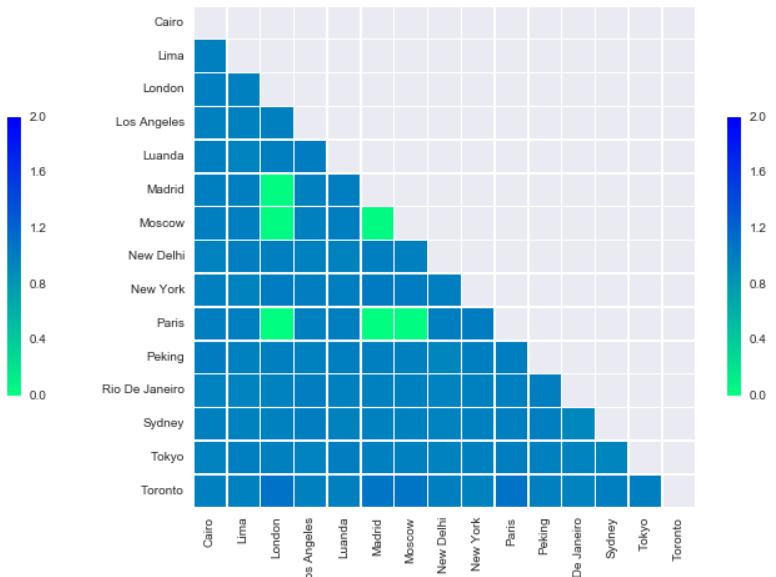
ADN, PAA tamanho 1000



Proteína, PAA tamanho 1000



ADN, PAA tamanho 1500

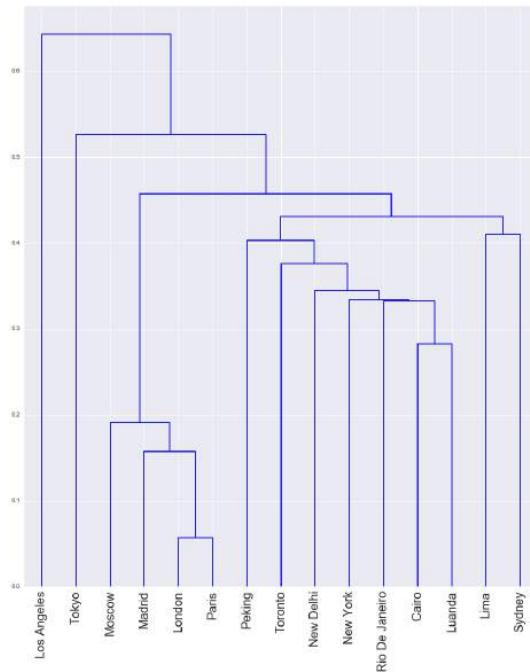


Proteína, PAA tamanho 1500

APÊNDICE E

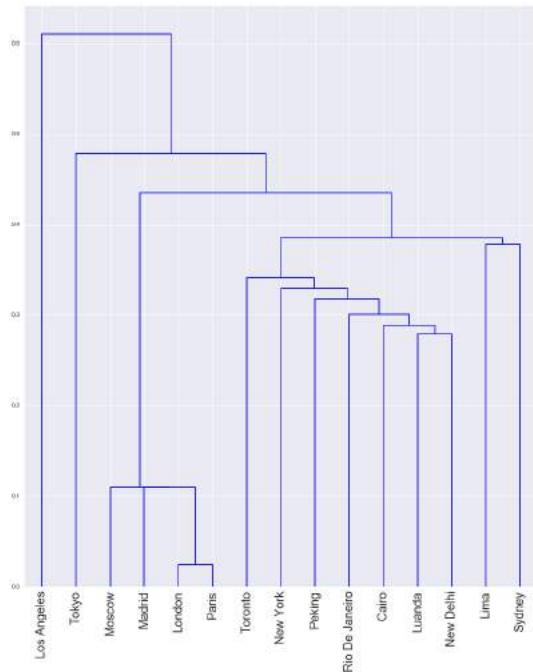
AGRUPAMENTOS HIERÁRQUICOS DE ACORDO COM AS DIMENSÕES DO ALFABETO E SÉRIE DE TEMPERATURAS

A. Alfabeto Do ADN (4 elementos)

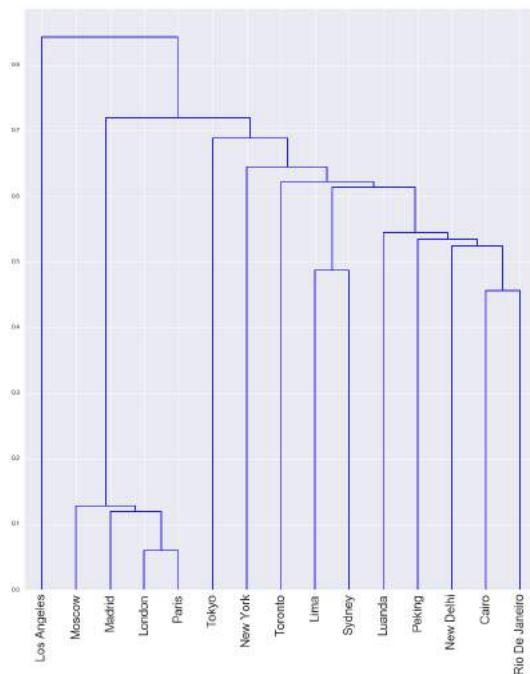


ADN, PAA tamanho 50

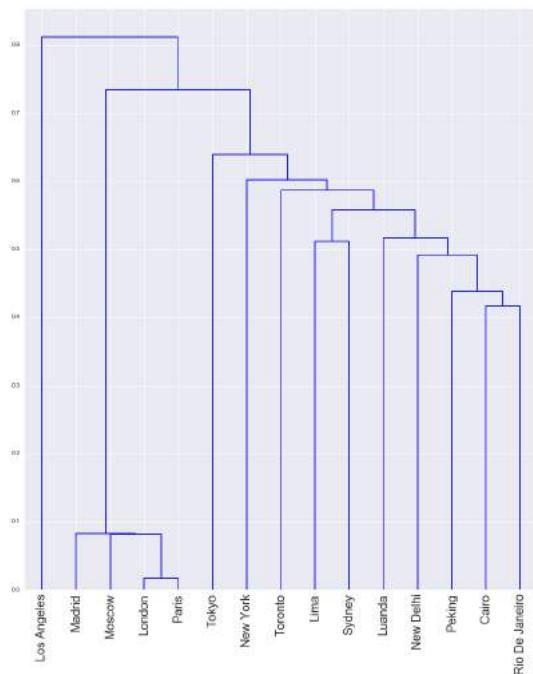
B. Alfabeto das proteínas (20 elementos)



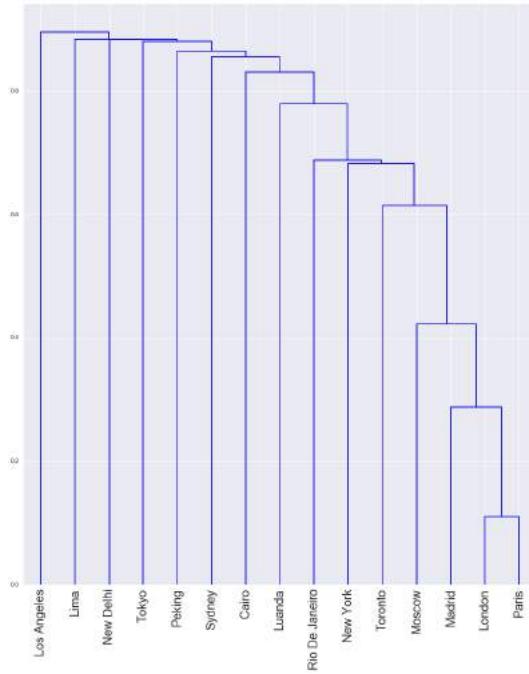
Proteína, PAA tamanho 50



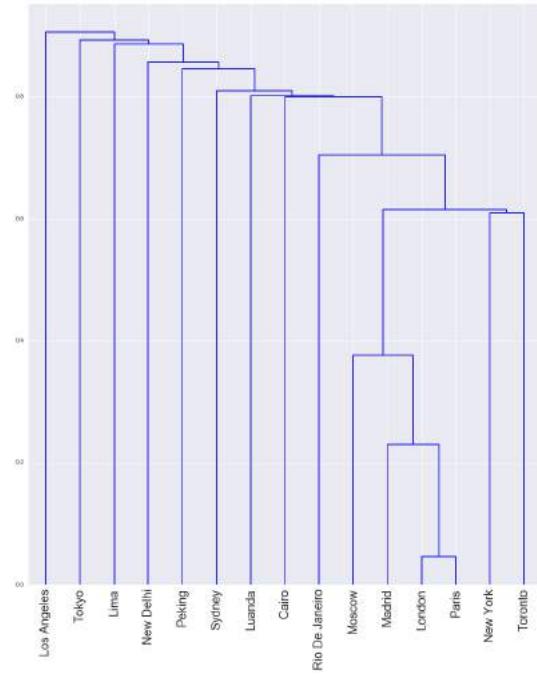
ADN, PAA tamanho 100



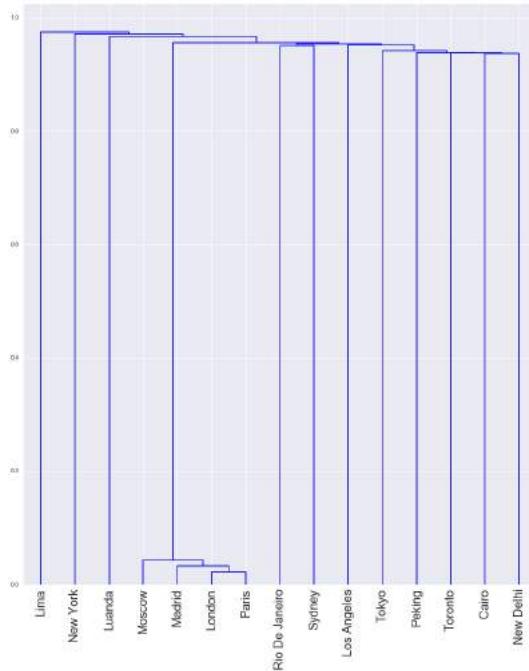
Proteína, PAA tamanho 100



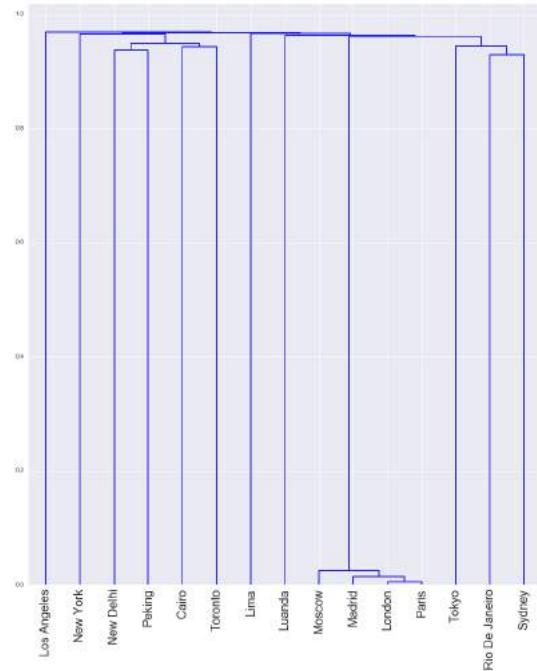
ADN, PAA tamanho 250



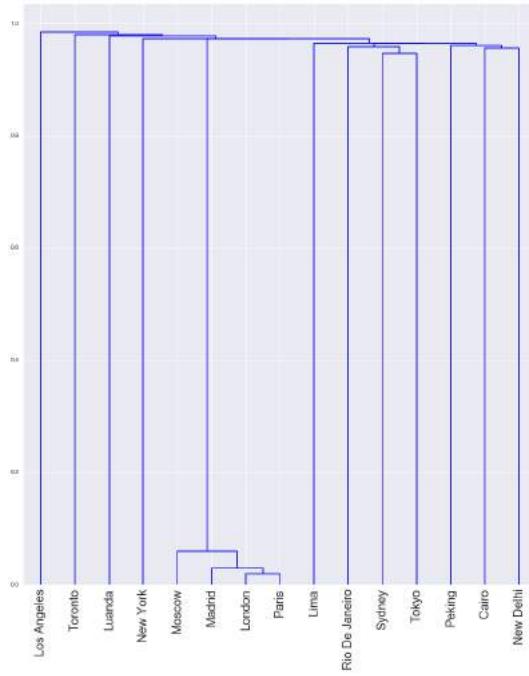
Proteína, PAA tamanho 250



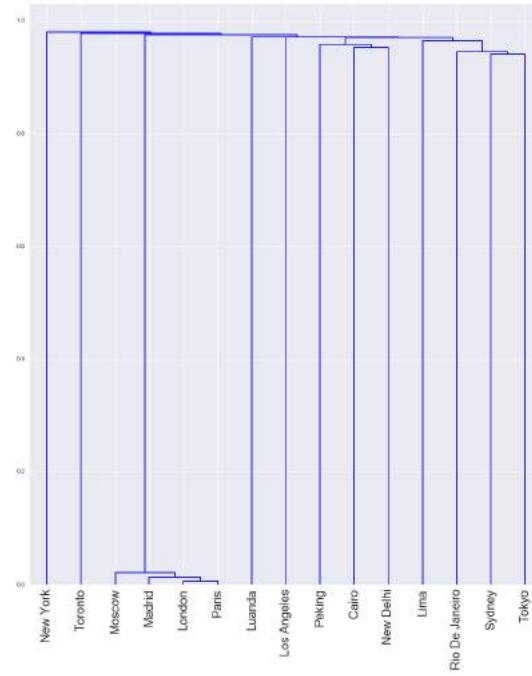
ADN, PAA tamanho 500



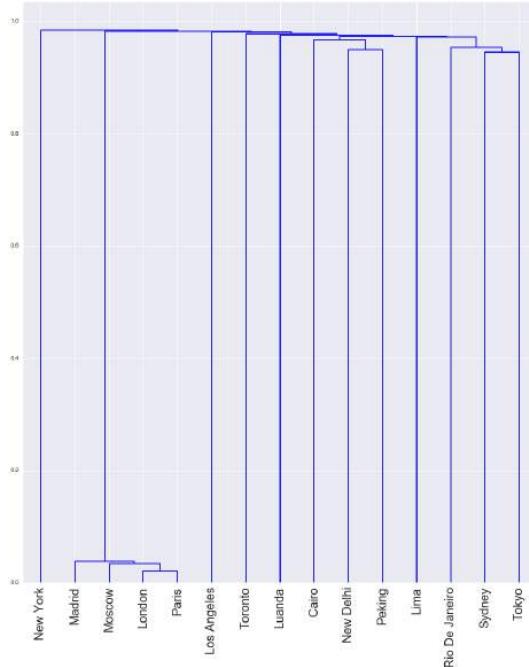
Proteína, PAA tamanho 500



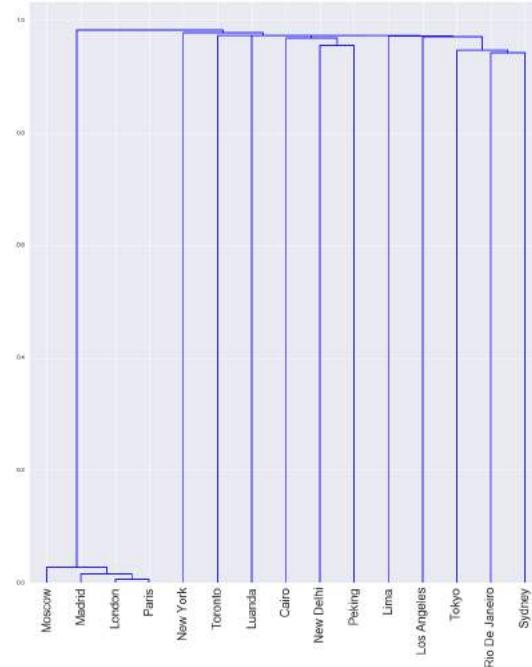
ADN, PAA tamanho 1000



Proteína, PAA tamanho 1000

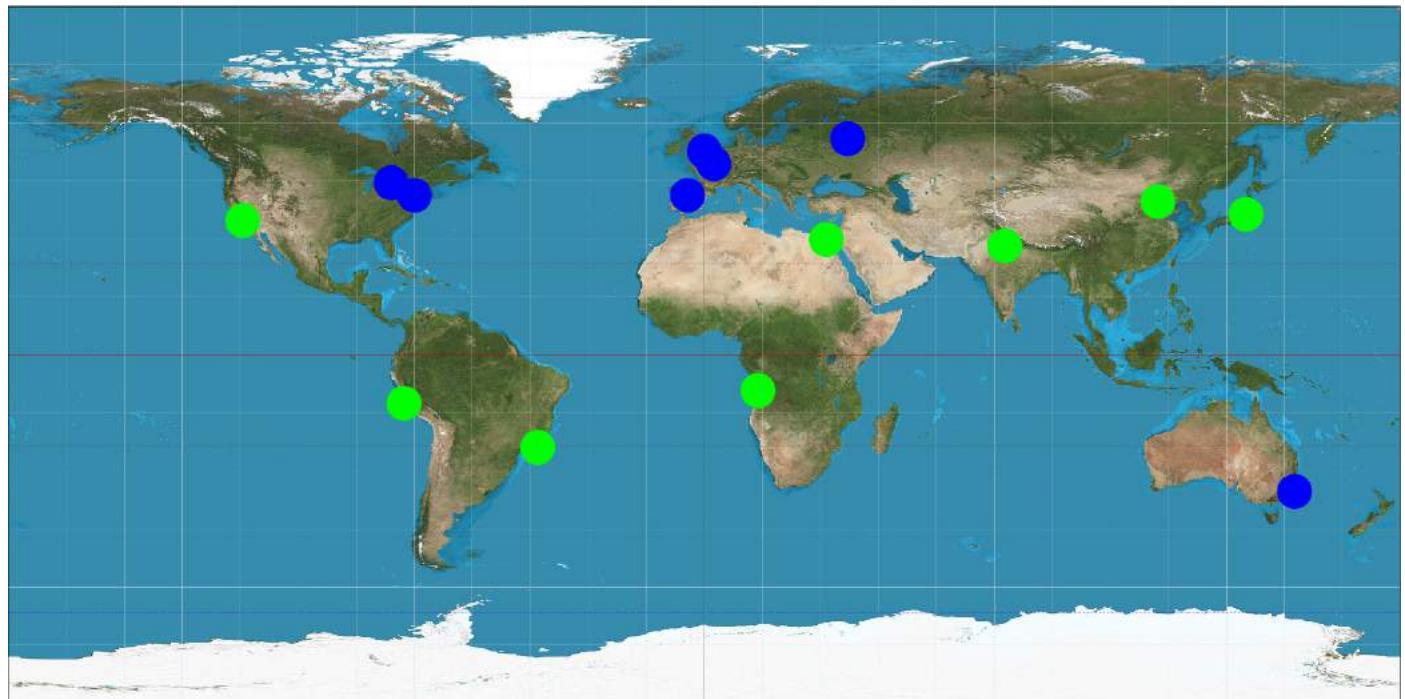


ADN, PAA tamanho 1500

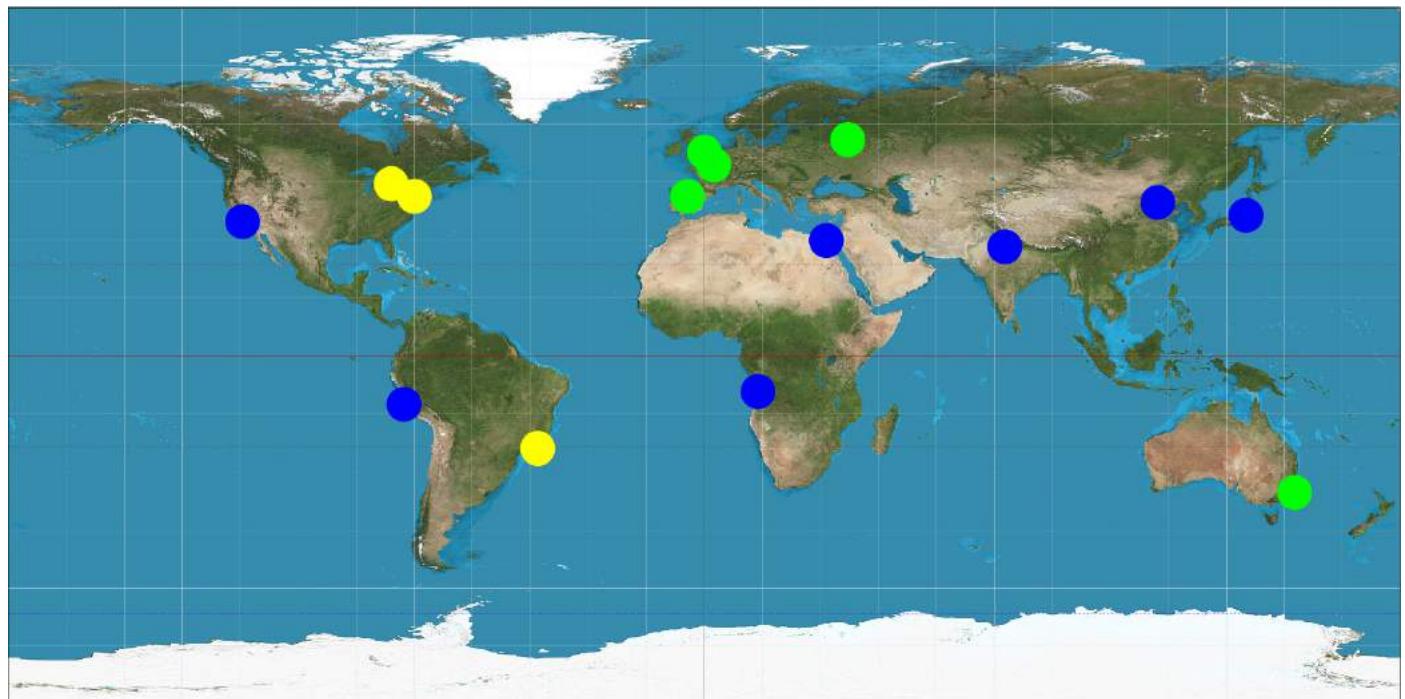


Proteína, PAA tamanho 1500

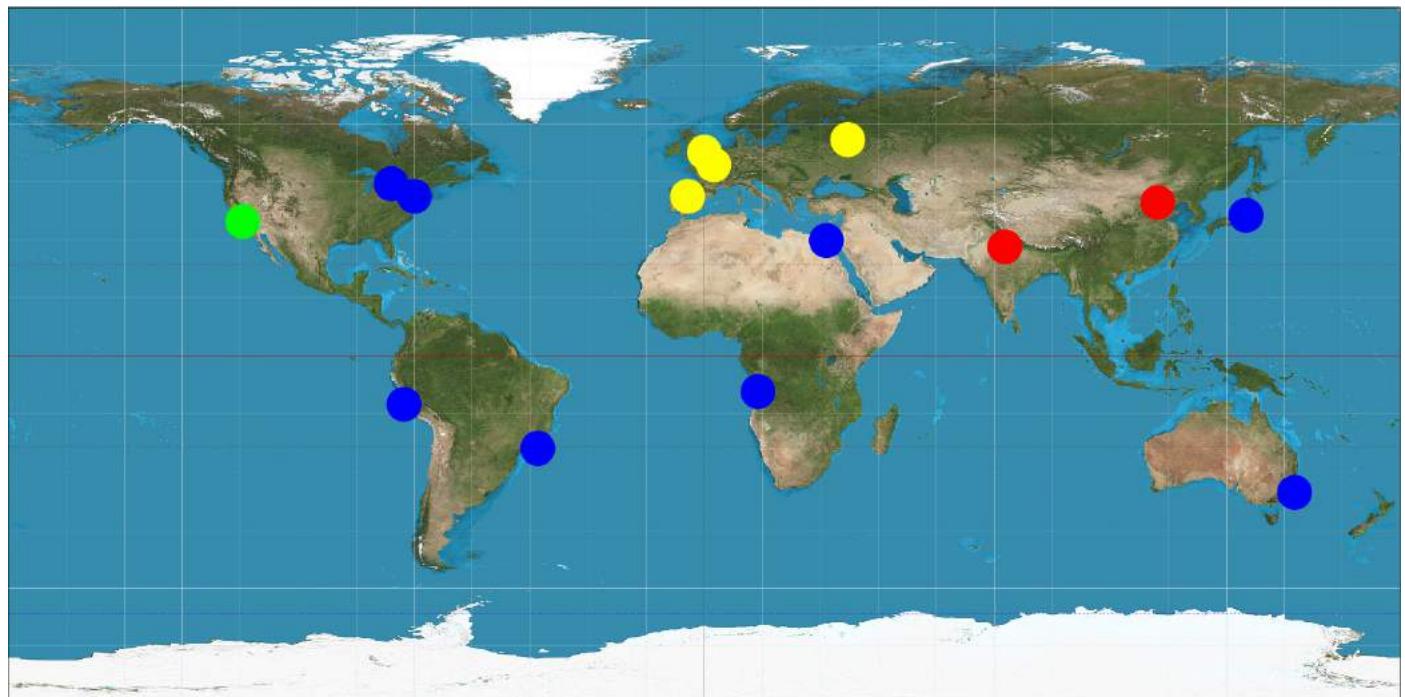
APÊNDICE F
AGRUPAMENTOS K-MEDOIDS DE ACORDO COM AS DIMENSÕES DA SÉRIE E NÚMERO DE GRUPOS



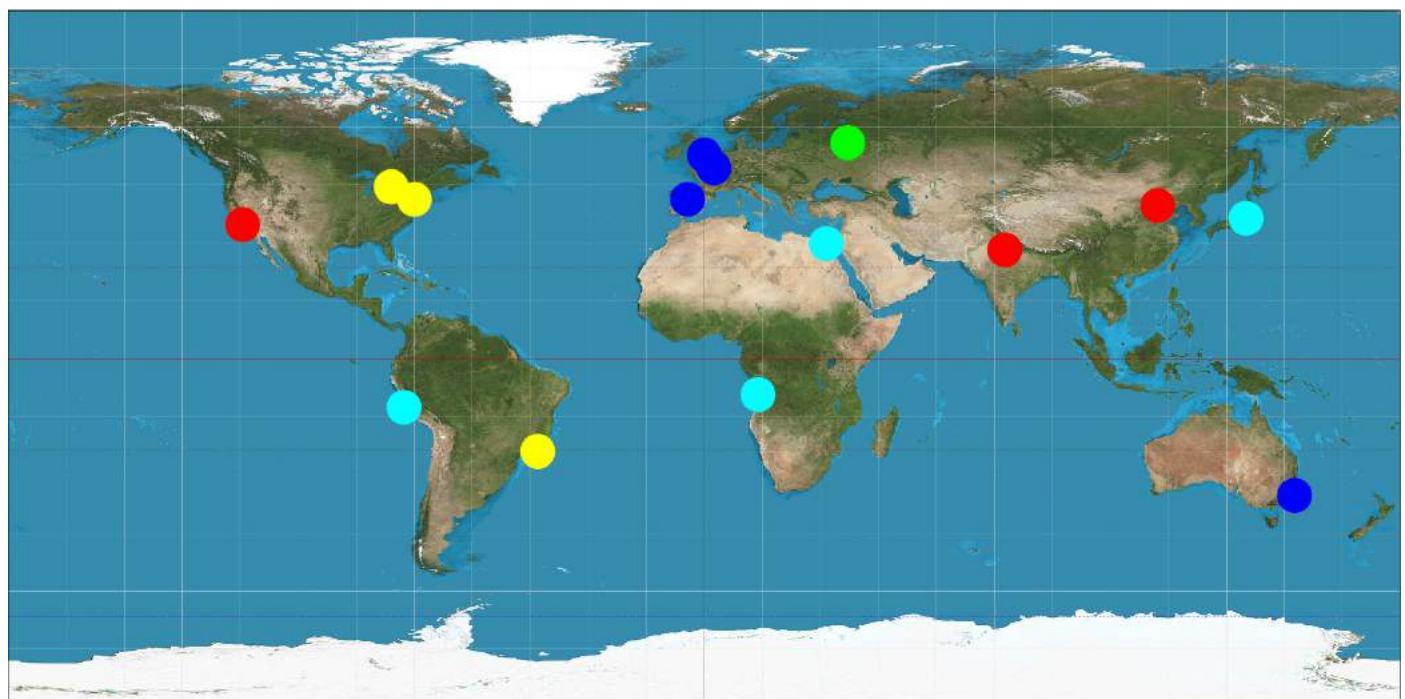
SAX(20,50), 2 clusters



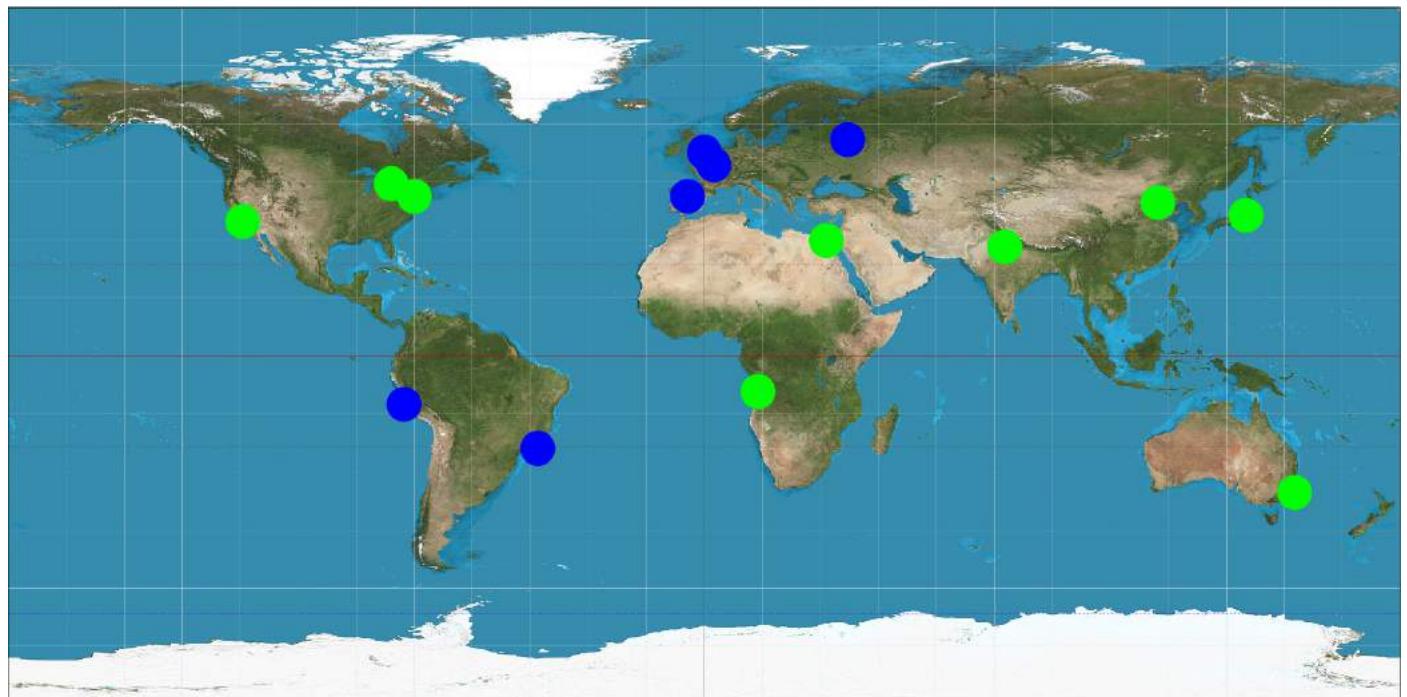
SAX(20,50), 3 clusters



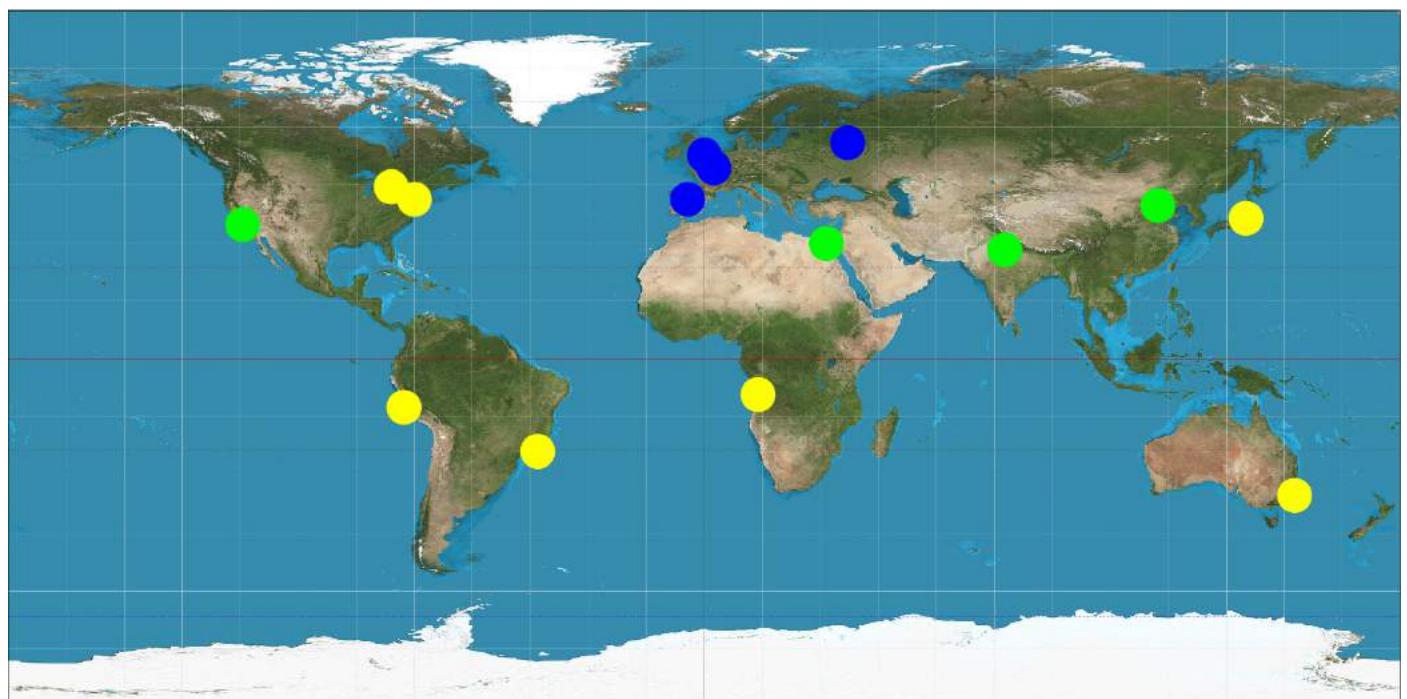
SAX(20,50), 4 clusters



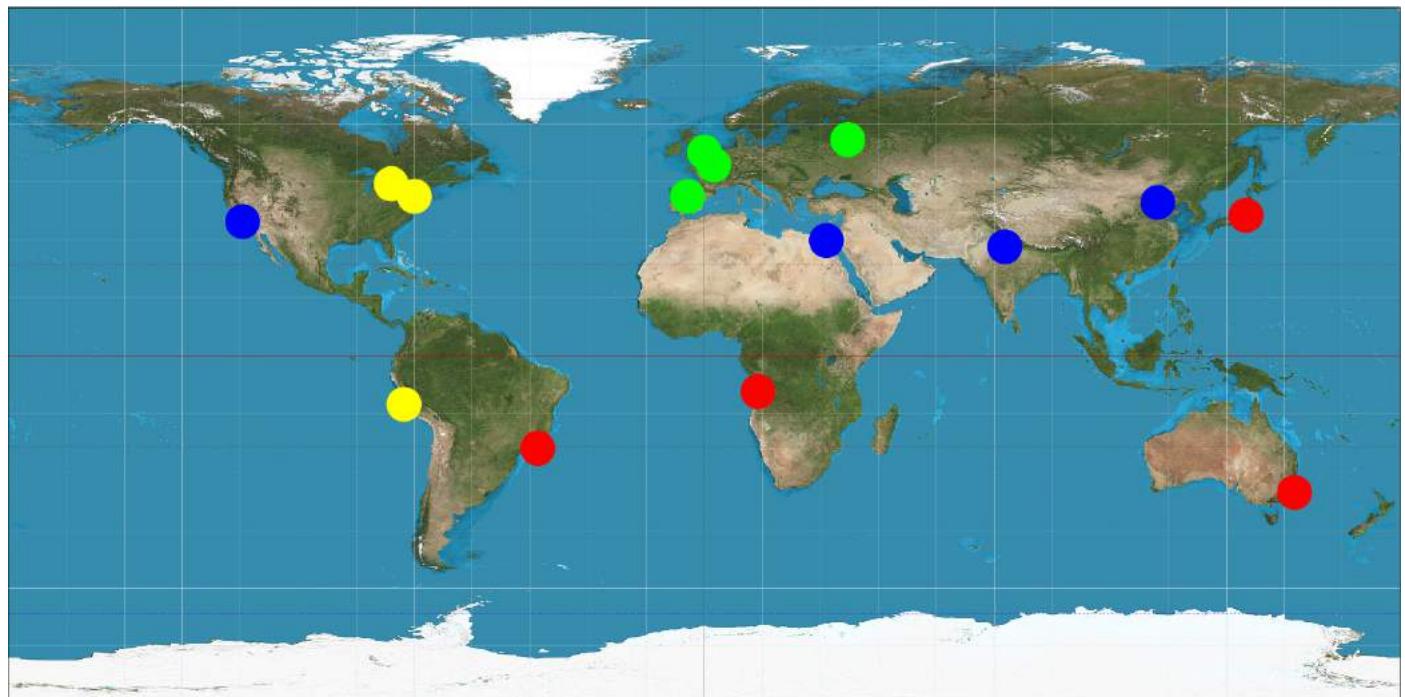
SAX(20,50), 5 clusters



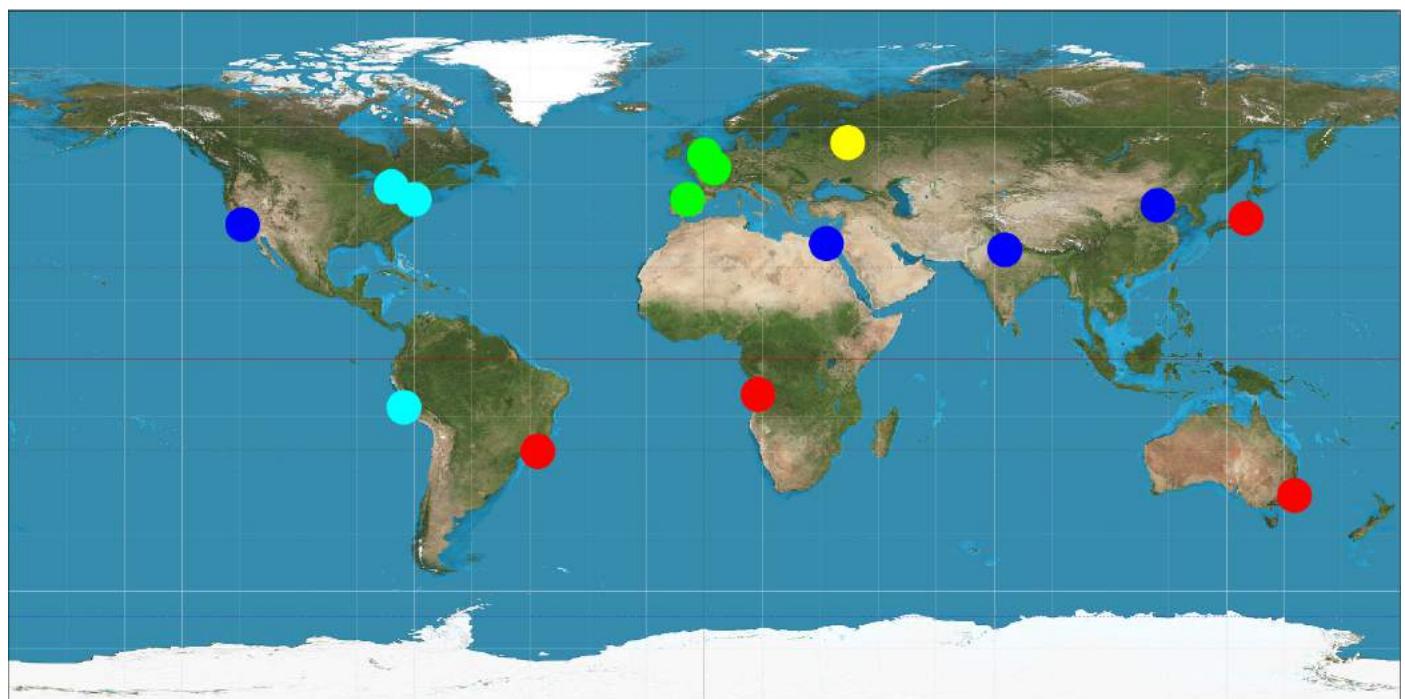
SAX(20,1500), 2 clusters



SAX(20,1500), 3 clusters

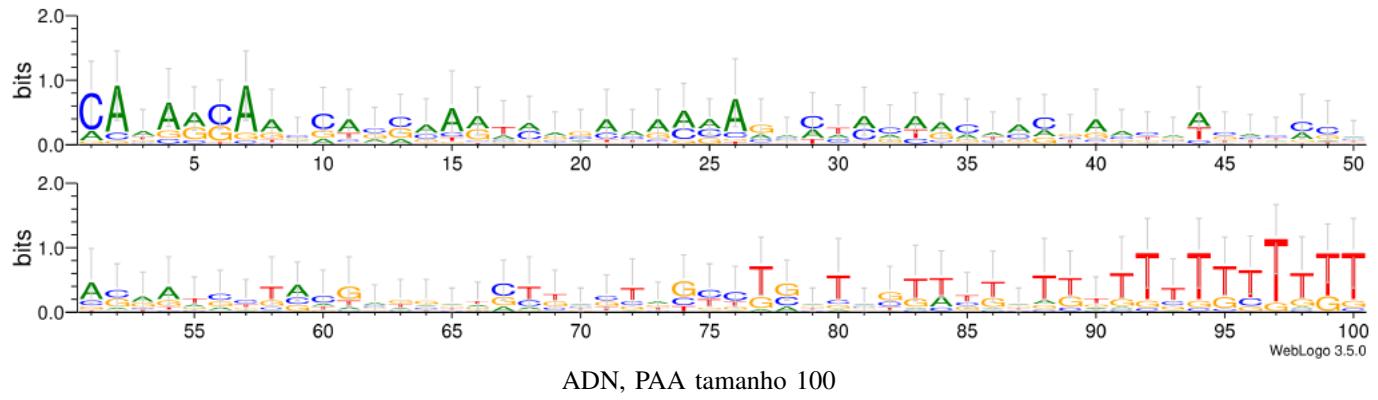
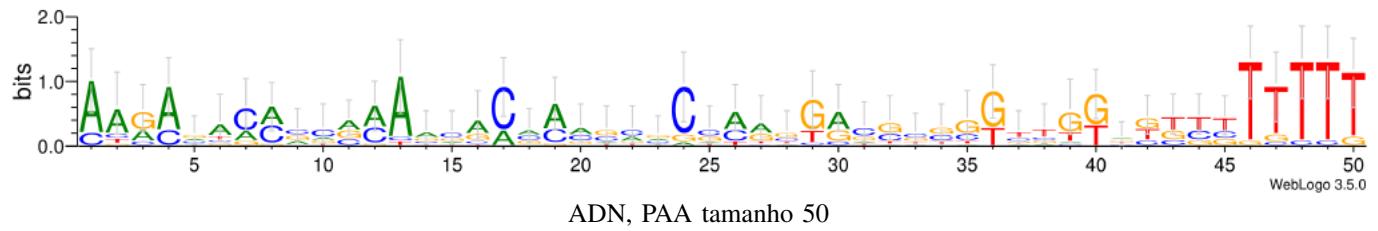


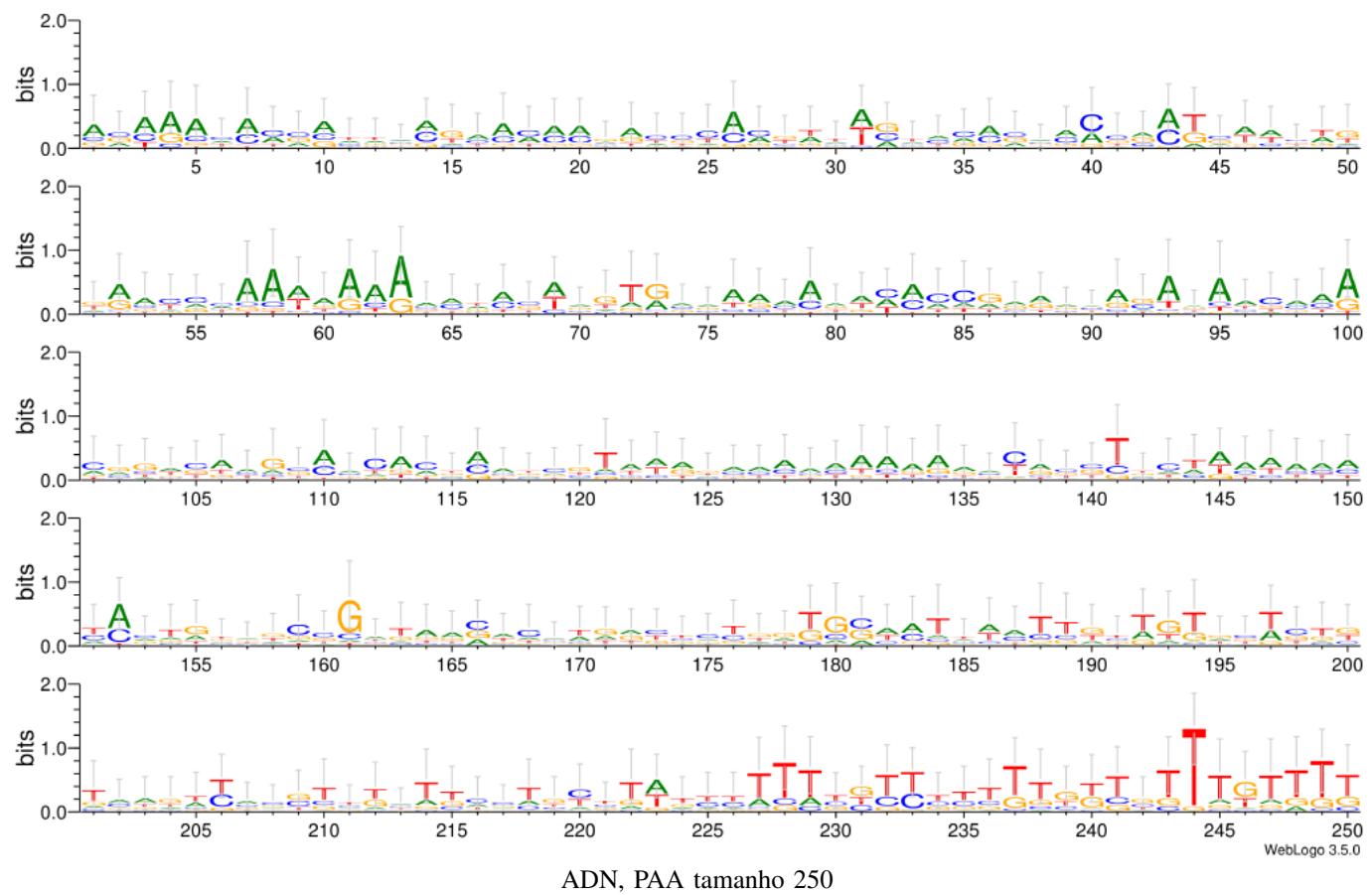
SAX(20,1500), 4 clusters

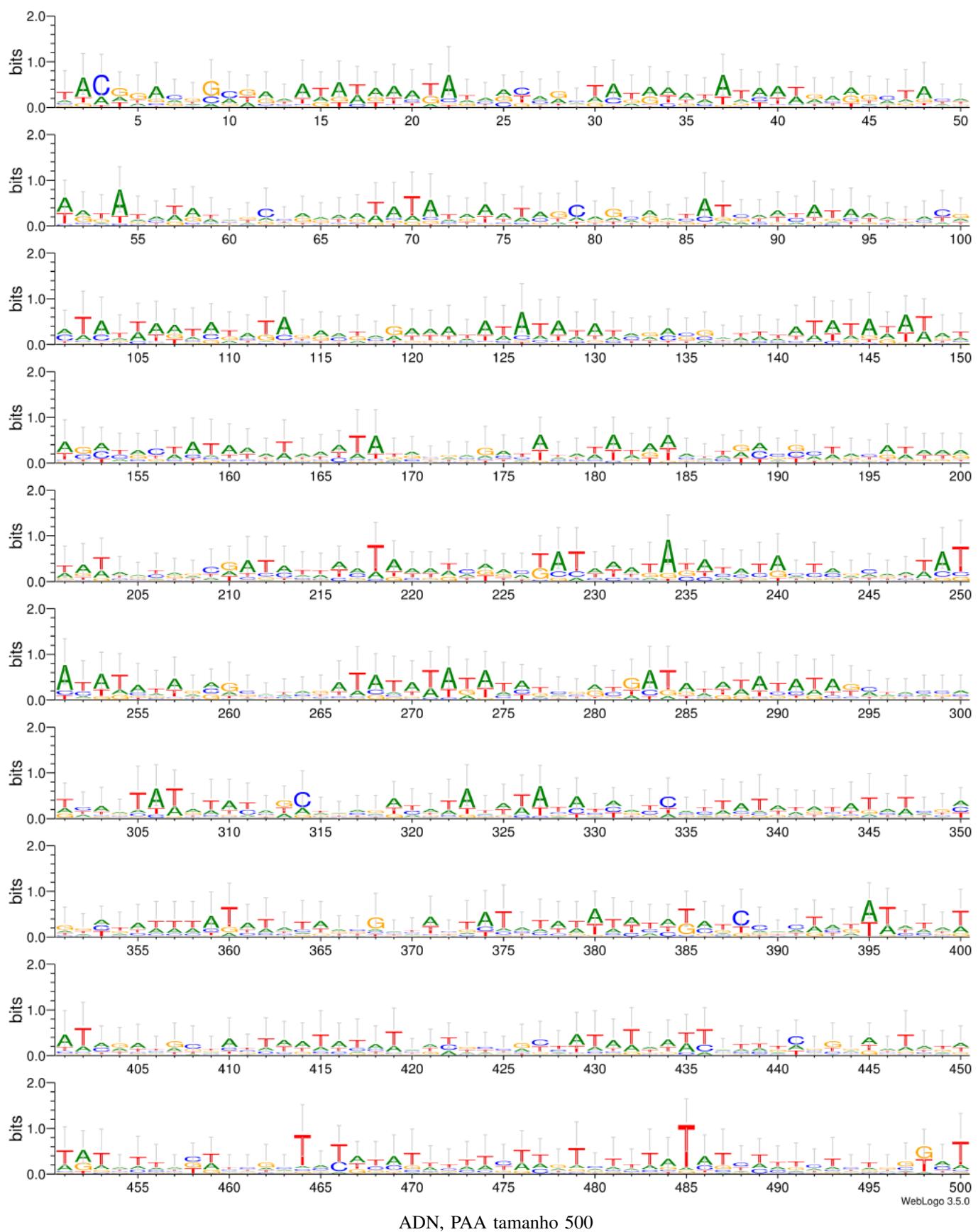


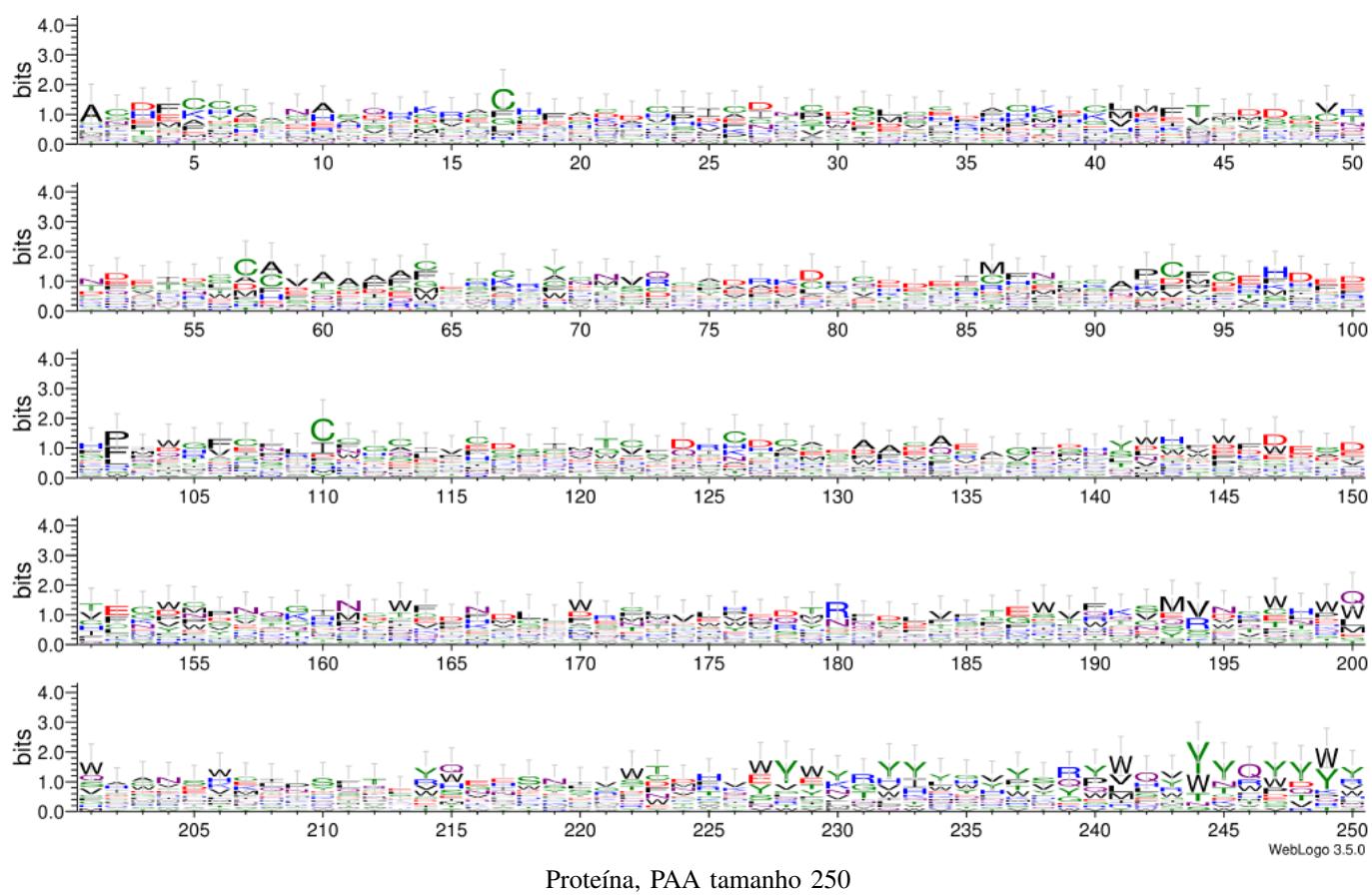
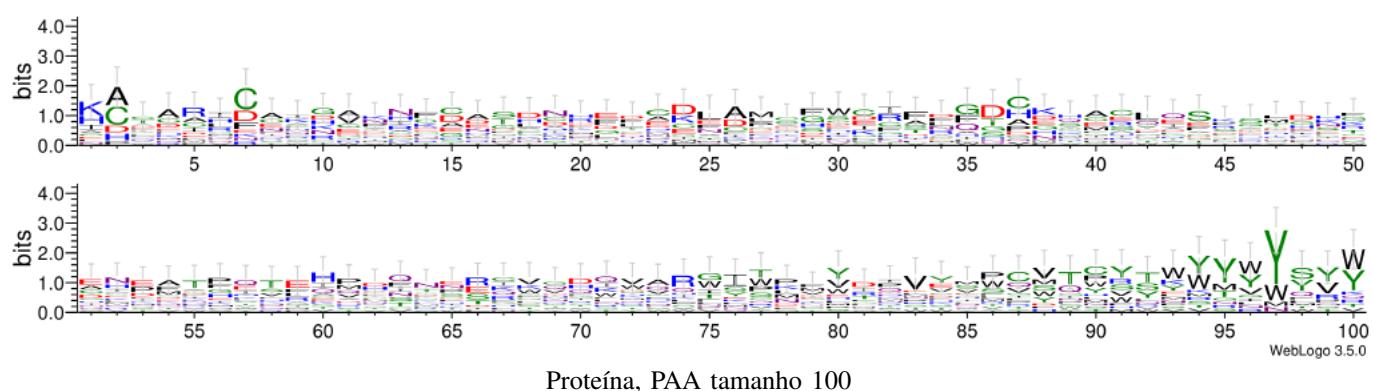
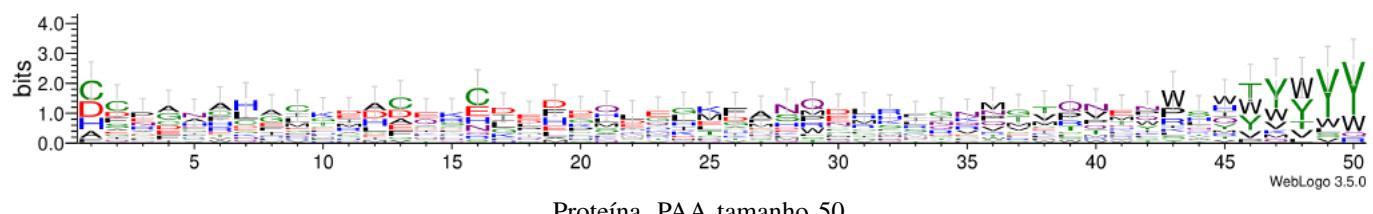
SAX(20,1500), 5 clusters

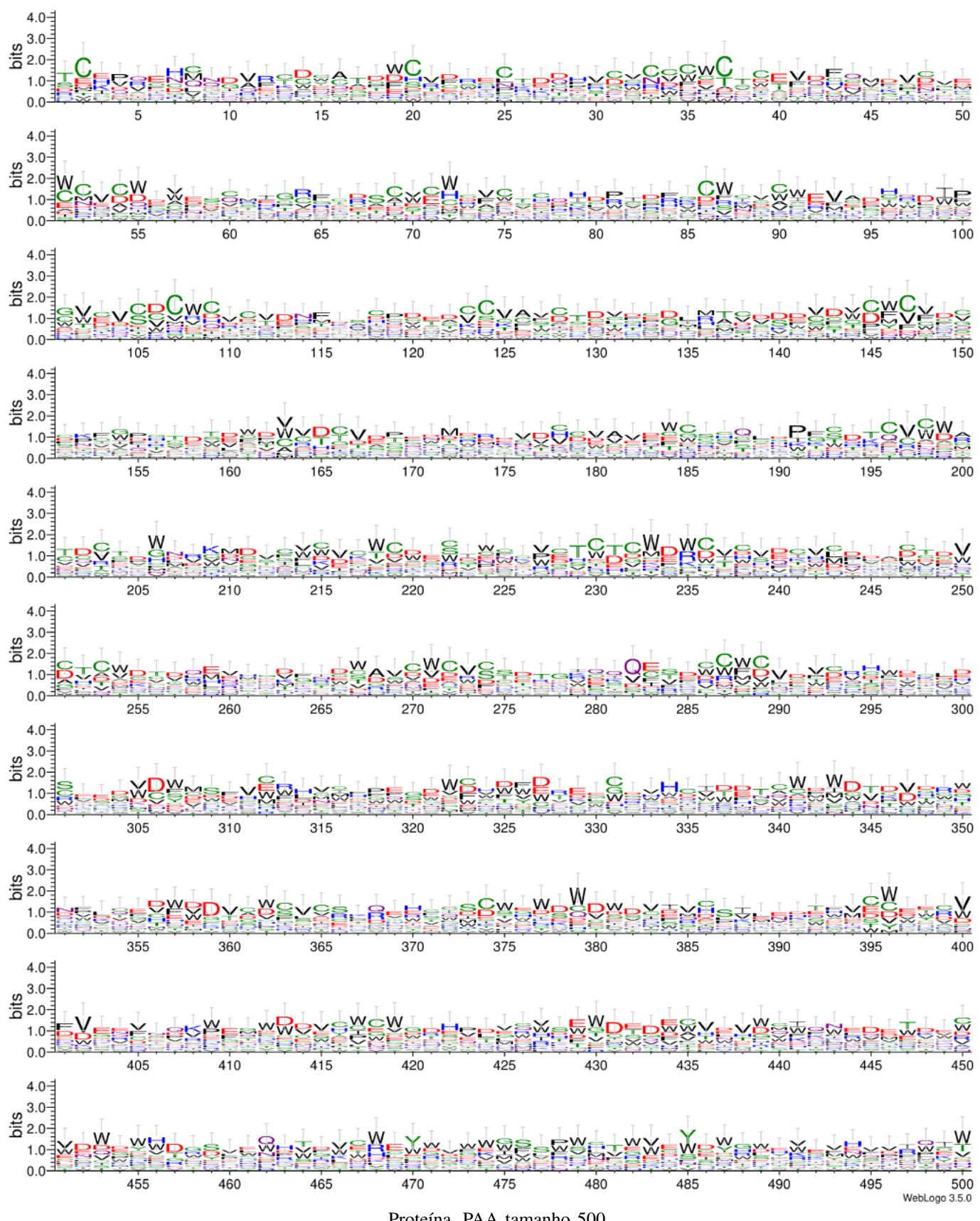
APÊNDICE G
SEQUENCE LOGOS DE ACORDO COM AS DIMENSÕES DO ALFABETO E SÉRIE DE TEMPERATURAS



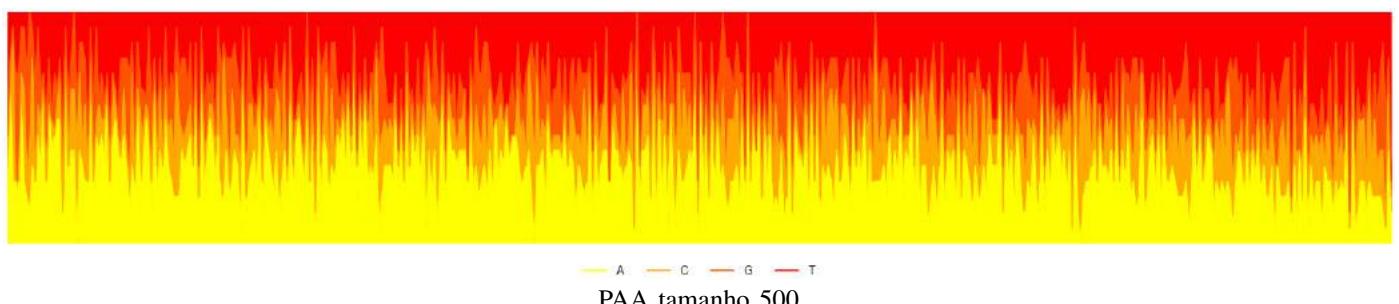
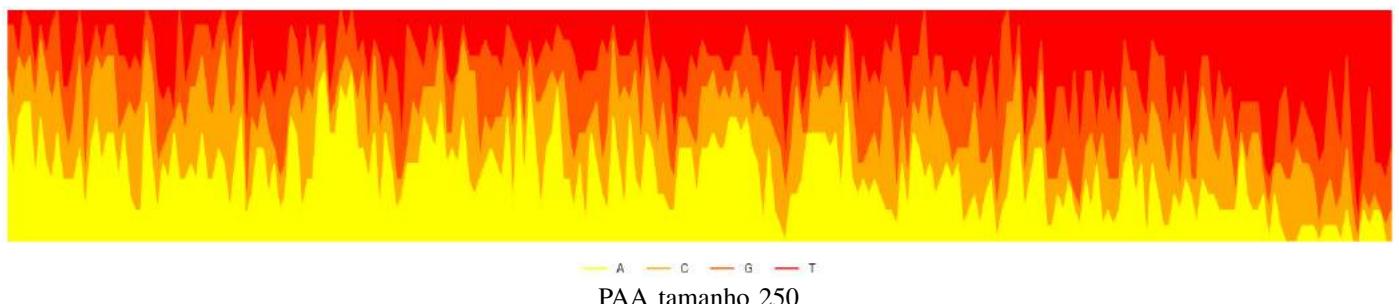
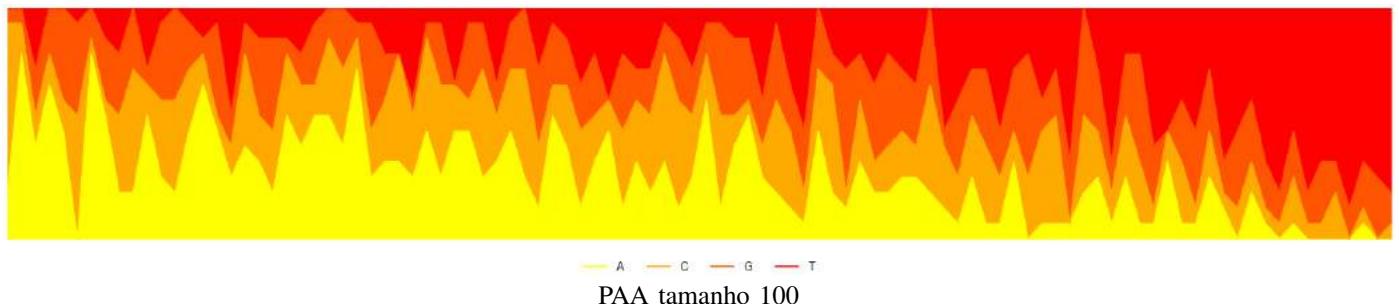
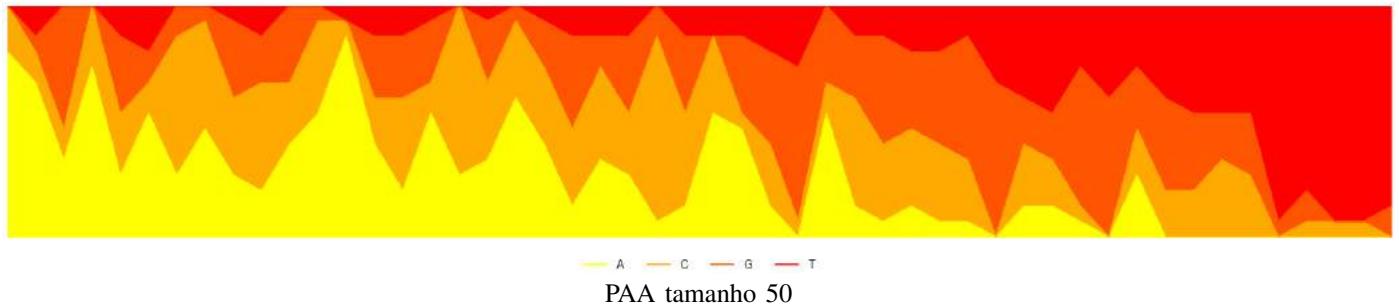


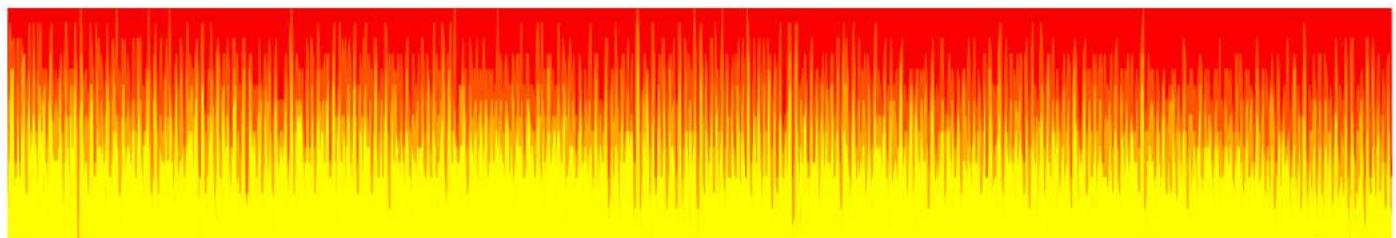




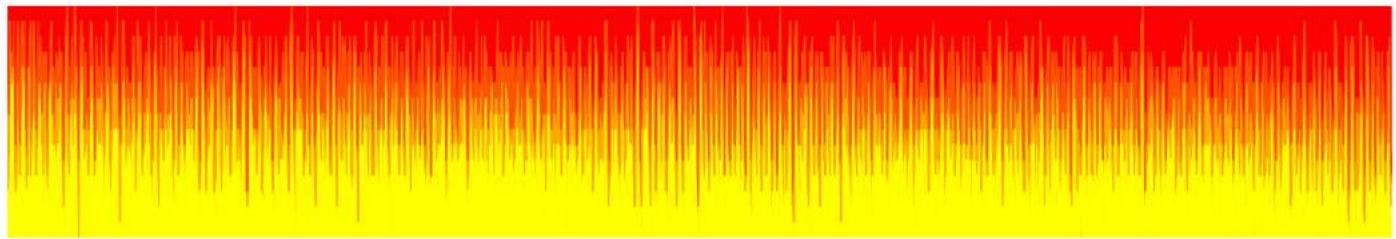


APÊNDICE H
SÉRIES DE TEMPERATURAS SELECIONADAS (PERCENTAGEM DE CADA ELEMENTO DO ALFABETO DO ADN)





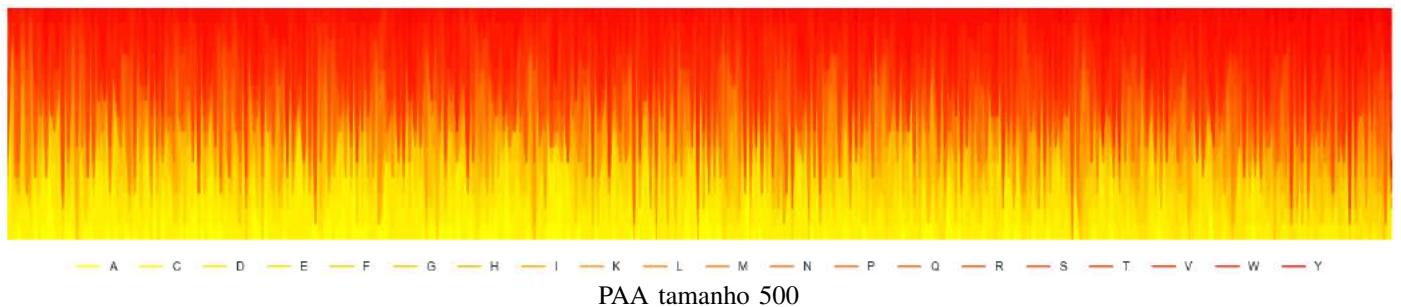
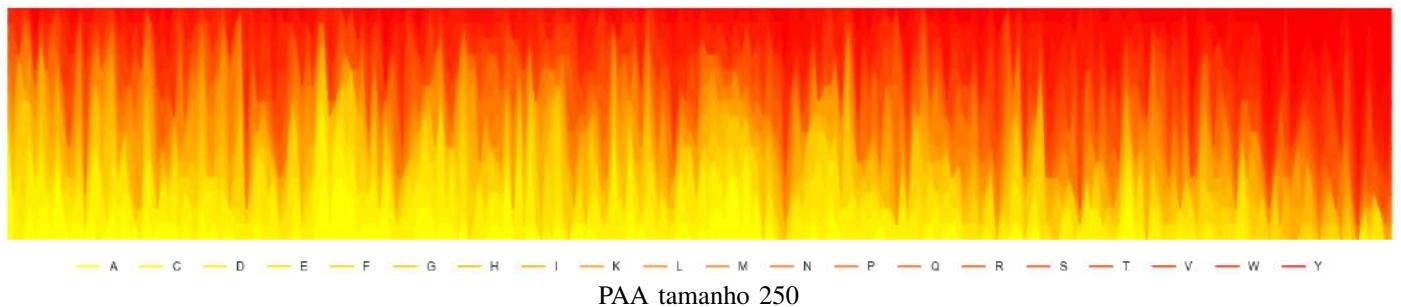
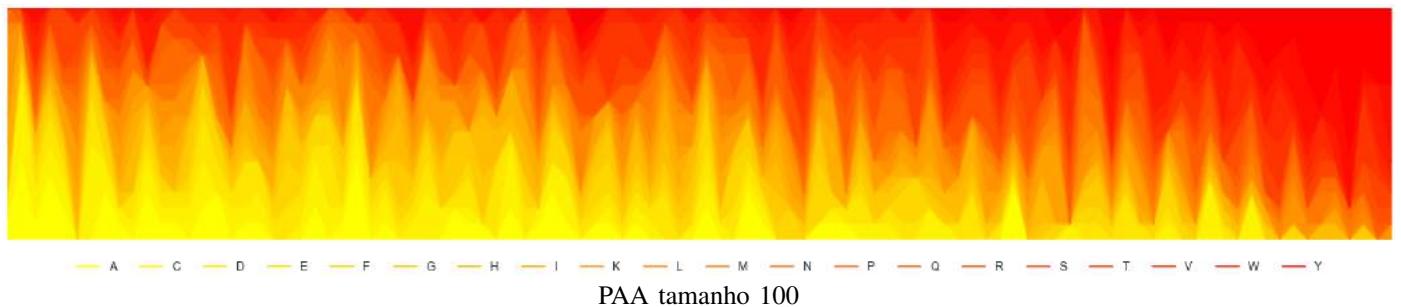
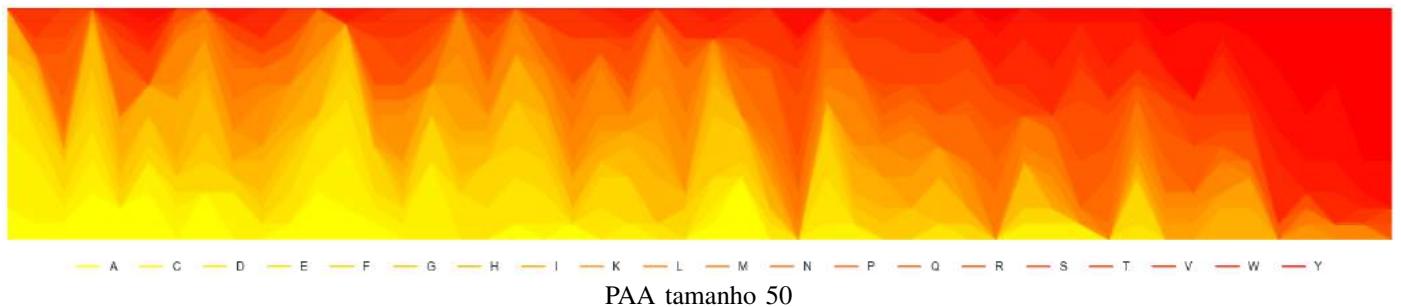
— A — C — G — T
PAA tamanho 1000

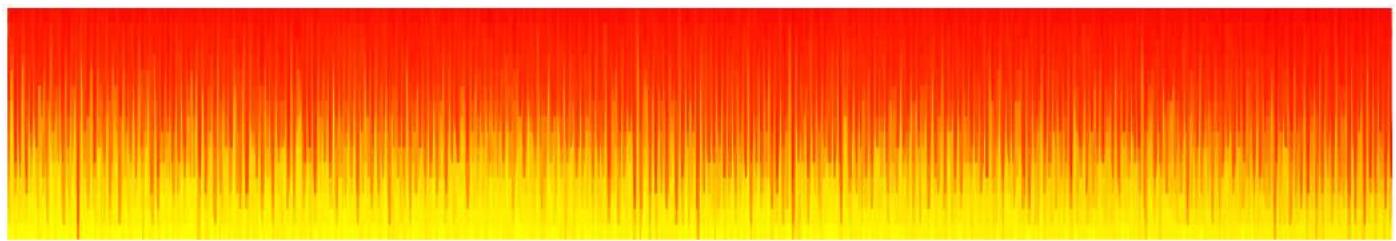


— A — C — G — T
PAA tamanho 1500

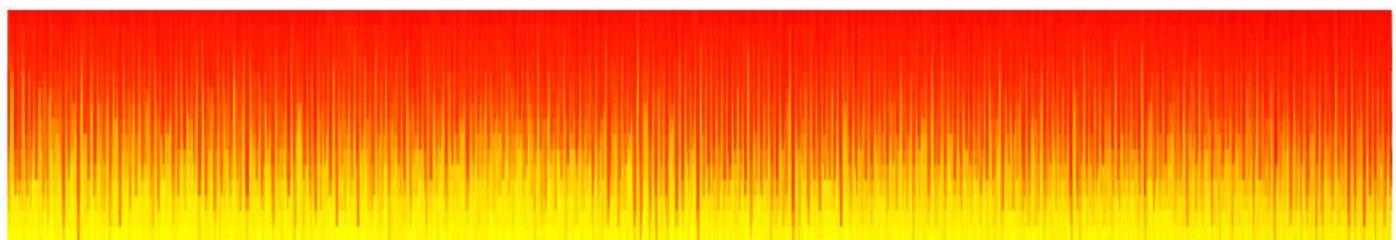
APÊNDICE I

SÉRIES DE TEMPERATURAS SELECIONADAS (PERCENTAGEM DE CADA ELEMENTO DO ALFABETO DAS PROTEÍNAS)





— A — C — D — E — F — G — H — I — K — L — M — N — P — Q — R — S — T — V — W — Y
PAA tamanho 1000



— A — C — D — E — F — G — H — I — K — L — M — N — P — Q — R — S — T — V — W — Y
PAA tamanho 1500