UNIVERSIDADE ESTADUAL PAULISTA
"JÚLIO DE MESQUITA FILHO"
Câmpus de São José do Rio Preto

Bianca Barreiro

Clusterings Hierárquicos em *Networks* e Aplicações

São José do Rio Preto
2019

Bianca Barreiro

Clusterings Hierárquicos em *Networks* e Aplicações

Dissertação apresentada como parte dos requisitos para obtenção do título de Mestre em Matemática, junto ao Programa de Pós-Graduação em Matemática, do Instituto de Biociências, Letras e Ciências Exatas da Universidade Estadual Paulista "Júlio de Mesquita Filho", Câmpus de São José do Rio Preto.

Orientador: Prof. Dr. Thiago de Melo
Financiadora: CAPES

São José do Rio Preto
2019

Barreiro, Bianca
B271c Clusterings Hierárquicos em Networks e Aplicações / Bianca Barreiro. -- São José do Rio Preto, 2019
82 p. : il., tabs.

Dissertação (mestrado) - Universidade Estadual Paulista (Unesp), Instituto de Biociências Letras e Ciências Exatas, São José do Rio Preto
Orientador: Thiago de Melo

1. Análise Topológica de Dados. 2. Clustering Hierárquico. 3. Networks. I. Título.

Sistema de geração automática de fichas catalográficas da Unesp. Biblioteca do Instituto de Biociências Letras e Ciências Exatas, São José do Rio Preto. Dados fornecidos pelo autor(a).

Essa ficha não pode ser modificada.

Bianca Barreiro

Clusterings Hierárquicos em *Networks* e Aplicações

Dissertação apresentada como parte dos requisitos para obtenção do título de Mestre em Matemática, junto ao Programa de Pós-Graduação em Matemática, do Instituto de Biociências, Letras e Ciências Exatas da Universidade Estadual Paulista "Júlio de Mesquita Filho", Câmpus de São José do Rio Preto.
Financiadora: CAPES

Comissão Examinadora

Prof. Dr. Thiago de Melo
Orientador

Sergio Tsuyoshi Ura
UNESP - Câmpus de Rio Claro

Marcio Fuzeto Gameiro
USP - Câmpus de São Carlos

Rio Claro
18 de fevereiro de 2019

AGRADECIMENTOS

Quero dedicar esse espaço para agradecer a todas as pessoas que foram importantes para mim desde o início da minha jornada até o presente momento.

Quero agradecer aos meus pais, Luiz Antonio Barreiro e Lucimara Cristina Fosaluza Barreiro, que sempre me apoiaram e acreditaram em minha capacidade e que fizeram inúmeros sacrifícios para que eu pudesse chegar onde estou.

Ao meu irmão Felipe, que mesmo ainda pequeno, me fez amadurecer e criar responsabilidades.

Ao meu namorado e melhor amigo Levi, que sempre me incentivou a continuar nessa jornada, nunca duvidou do meu potencial, até mesmo quando eu duvidei.

Ao meu orientador, Prof. Dr. Thiago de Melo, que teve paciência pra me ensinar sobre programação, acreditou em meu potencial e confiou em meu trabalho.

À Profa. Dra. Alice Kimie Miwa Libardi, que foi minha primeira orientadora, me fez apreciar a beleza de matemática e acreditou na minha capacidade.

A todos os outros professores que passaram pela minha graduação e pós-graduação e que de alguma forma fizeram com que eu me apaixonasse cada vez mais pela matemática e pelo compromisso de sempre querer aprender coisas novas.

Dedico um agradecimento especial a todos os meus amigos de graduação e pós-graduação, que me ajudaram e me acompanharam em toda esta jornada.

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Código de Financiamento 001.

RESUMO

Neste trabalho estudamos *networks* e métodos de *Clustering* Hierárquico de forma axiomática. Apresentamos alguns programas na linguagem Python aplicados à análise de dados de migração populacional, com o intuito de ilustrar os métodos estudados.

Palavras-chave: Análise Topológica de Dados, *Clustering* Hierárquico, *Networks*.

ABSTRACT

In this work we study networks and an axiomatic construction of Hierarchical Clustering. We present some Python programs used to analyze human migration data and illustrate the studied methods.

Keywords: Topological Data Analysis, Hierarchical Clustering, Networks.

Lista de Figuras

Lista de Tabelas

Sumário

Introdução

Agrupar, armazenar e analisar dados é de fundamental importância em todas as áreas da Ciência. Quase todo tipo de informação pode ser transformada em um conjunto de dados.

Por exemplo, a ideia de usar um conjunto de pontos (ou nuvem de pontos) para visualizar um objeto foi introduzida recentemente, mas não tornou-se popular até o surgimento de uma grande quantia de dados disponíveis para estudo. Portanto, o desenvolvimento de métodos para entendermos dados tão complexos se faz necessário, e um grande esforço para adaptar métodos topológicos a várias aplicações tem sido feito recentemente. Esta nova linha de pesquisa é chamada de Análise Topológica de Dados–TDA (do inglês *Topological Data Analysis*).

Um conjunto de dados admite diferentes interpretações que dependem se esse conjunto de dados é métrico; simétrico, mas não necessariamente métrico, ou até mesmo assimétrico.

Esses conjuntos de dados são denominados *networks*. Teoricamente, uma *network* é um conjunto finito de nós juntamente com uma função dissimilaridade, em que cada nó está ligado a outro e existe um peso entre eles. Um nó está sempre ligado a ele mesmo, porém esse peso é nulo. Além disso, quando falamos de dois nós distintos x e x', seus pesos não são necessariamente os mesmos, ou seja, o peso de x estar ligado a x' pode ser diferente do peso de x' a x.

Quando temos um conjunto de dados, precisamos saber como interpretá-lo. Um dos meios para fazê-lo é construir métodos de *Clustering* Hierárquicos. Esses métodos servem para nos dizer quando os nós da *network* se agrupam mais rapidamente que outros, ou seja, quando temos um conjunto de dados de um problema real, dois dados se agruparem mais rapidamente significa que eles têm mais afinidade. Para interpretar de forma mais clara o que esses métodos nos dizem, usamos dendrogramas.

Os dendrogramas podem ser interpretados como uma estrutura que produz diferentes *clusterings* em diferentes parâmetros. No Capítulo 5, usamos dados reais de migração dos EUA e migração mundial para produzir os dendrogramas descritos no Apêndice A.

Neste trabalho, estudamos *networks* e métodos de *Clustering* Hierárquico de forma axiomática. Apresentamos alguns programas na linguagem Python aplicados à análise de dados de migração populacional, com o intuito de ilustrar os métodos estudados. A principal referência para esse projeto é [1].

1 Preliminares

Neste capítulo, apresentamos os principais conceitos sobre *networks*, necessários para as aplicações.

Seja X um conjunto finito não-vazio. Uma função $A_X : X \times X \to \mathbb{R}_+$ é uma *função dissimilaridade* se

$$A_X(x, x') = 0 \Leftrightarrow x = x',\ \forall x, x' \in X.$$

Observação 1.1. Uma função dissimilaridade A_X não é, necessariamente, uma métrica no conjunto finito X, pois não se exige a validade da desigualdade triangular. Mais ainda, a função pode ser assimétrica, ou seja, $A_X(x, x') \neq A_X(x', x)$, para algum $x, x' \in X$.

Definição 1.2. *Uma* network N_X *é um par* (X, A_X) *onde* X *é um conjunto finito não-vazio e* $A_X : X \times X \to \mathbb{R}_+$ *é uma função dissimilaridade. O conjunto de todas as* networks *será denotado por* $\mathcal{N}$.

Em alguns casos é conveniente escrever $X = \{x_1, x_2, \ldots, x_n\}$ e chamar cada elemento de *nó*, além de interpretar a função dissimilaridade A_X como uma matriz $A_X \in M(n \times n, \mathbb{R}_+)$, com $(A_X)_{i,j} = A_X(x_i, x_j)$, para $1 \leq i, j \leq n$. Note que os elementos da diagonal $(A_X)_{i,i} = A_X(x_i, x_i)$ são zero.

Networks em $\mathcal{N}$ podem ter diferentes conjuntos de nós X, assim como diferentes funções dissimilaridades A_X.

Se $X = \{x_1\}$, necessariamente A_X é nula e, neste caso, a *network* (X, A_X) é chamada *trivial*. As menores *networks* não triviais contêm dois nós p e q e duas dissimilaridades α e β, como na Figura 1.1.

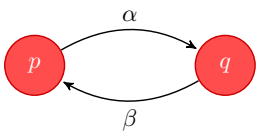


Figura 1.1: A menor *network* não trivial

Definição 1.3. *Considere a função dissimilaridade* $A_{p,q}$ *dada por* $A_{p,q}(p, q) = \alpha$ *e* $A_{p,q}(q, p) = \beta$, *para algum* $\alpha, \beta > 0$ *e defina a* network *de dois nós* $\overrightarrow{\Delta}_2(\alpha, \beta)$ *com parâmetros* α *e* β *por*

$$\overrightarrow{\Delta}_2(\alpha, \beta) \coloneqq (\{p, q\}, A_{p,q}). \tag{1.1}$$

Exemplo 1.4. Considere a *network* da Figura 1.2, cujo conjunto de nós é $X = \{a, b, c\}$. Assim, podemos interpretar a função dissimilaridade como uma matriz (não-simétrica)

$$A_X = \begin{bmatrix} 0 & 2 & 5 \\ 4 & 0 & 1 \\ 3 & 7 & 0 \end{bmatrix}.$$

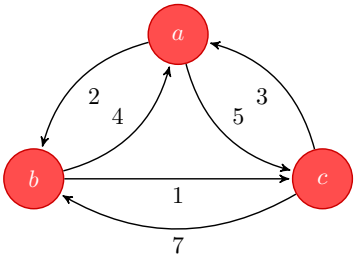


Figura 1.2: *Network* assimétrica de três nós

Definição 1.5. *Um* clustering *do conjunto X é uma partição P_X de X, isto é, uma coleção $P_X = \{B_1, \ldots, B_J\}$ de subconjuntos de X que são dois a dois disjuntos, isto é, $B_i \cap B_j = \emptyset$ para $i \neq j$, e cobrem X, ou seja, $\cup_{i=1}^{J} B_i = X$. Os conjuntos $B_1, \ldots, B_J$ são chamados de* clusters *de P_X.*

Uma partição $P_X = \{B_1, \ldots, B_J\}$ de X sempre induz, e é induzida por, uma relação de equivalência $\sim_{P_X}$ em X, onde para todo $x, x' \in X$ temos que $x \sim_{P_X} x'$ se, e somente se, x e x' pertencem a um mesmo *cluster* B_i, para algum i.

Neste trabalho, vamos focar em métodos de *clustering* hierárquicos. O resultado desse tipo de método não é uma única partição P_X, mas sim uma coleção D_X de partições $D_X(\delta)$ de X indexadas por um parâmetro $\delta \geq 0$, para a qual escrevemos $x \sim_{D_X(\delta)} x'$ se, e somente se, x e x' estão em um mesmo *cluster* de $D_X(\delta)$.

Uma coleção D_X é chamada de dendrograma se possui as seguintes propriedades:

(D1) *Condições de fronteira:* Para $\delta = 0$, a partição $D_X(0)$ possui apenas nós unitários de X, ou seja,

$$D_X(0) = \{\{x\},\ x \in X\}$$

e para algum δ_0 suficientemente grande, $D_X(\delta_0)$ possui todos os elementos de X em um único conjunto, isto é,

$$D_X(\delta_0) = \{X\}.$$

(D2) *Hierarquia:* À medida que δ aumenta, os *clusters* podem ser combinados, mas nunca separados. Mais precisamente, para quaisquer $\delta_1 < \delta_2$, se $x \sim_{D_X(\delta_1)} x'$ então $x \sim_{D_X(\delta_2)} x'$, para quaisquer pontos $x, x' \in X$.

(D3) *Continuidade à direita:* Para todo $\delta \geq 0$, existe $\epsilon > 0$ tal que

$$D_X(\delta) = D_X(\delta'),\ \forall \delta' \in [\delta, \delta + \epsilon].$$

Observação 1.6. Note que se o processo de *clustering* for via bola fechada, ou seja, $x \sim_{d_X(\delta)} x' \Leftrightarrow d_X(x,x') \leq \delta$, em que d_X é uma métrica, a condição (D1) é trivial. A segunda parte da condição (D1) juntamente com (D2) implica que devemos ter $D_X(\delta) = \{X\}$ para todo $\delta \geq \delta_0$. Denotamos por $[x]_\delta$ a classe de equivalência de $x \in X$ no parâmetro δ, ou seja, $[x]_\delta \coloneqq \{x' \in X \mid x \sim_{D_X(\delta)} x'\}$. Por (D1), temos que $[x]_0 = \{x\}$ e $[x]_{\delta_0} = X$, para cada $x \in X$.

Denotando por $\mathcal{D}$ o conjunto de todos os dendrogramas, um método de *Clustering* Hierárquico é uma função

$$\mathcal{H} : \mathcal{N} \to \mathcal{D},$$

tal que o conjunto X é preservado, ou seja, a imagem de uma *network* sobre X é um dendrograma sobre X.

Dada uma *network* $N_X = (X, A_X)$, denotamos por $D_X = \mathcal{H}(X, A_X)$ sua imagem pelo método $\mathcal{H}$.

Definição 1.7. *Sejam (X, A_X) uma* network *e $x, x' \in X$. Uma cadeia de x a x' é uma sequência ordenada de nós em X, escrita como*

$$C(x,x') \coloneqq [x = x_0, x_1, \ldots, x_l = x'],$$

a qual dizemos que conecta x a x'. Os links *da cadeia são as arestas que conectam nós consecutivos na direção imposta pela cadeia.*

Definição 1.8. *Dadas duas cadeias $C(x,x') = [x = x_0, x_1, \ldots, x_l = x']$ e $C(x',x'') = [x' = x'_0, x'_1, \ldots, x'_{l'} = x'']$, definimos a cadeia concatenada,*

$$C(x,x') \uplus C(x',x'') \coloneqq [x = x_0, \ldots, x_l = x' = x'_0, \ldots, x'_{l'} = x''].$$

A operação $\uplus$ de concatenação de cadeias é associativa, isto é,

$$[C(x,x') \uplus C(x',x'')] \uplus C(x'',x''') = C(x,x') \uplus [C(x',x'') \uplus C(x'',x''')].$$

Além disso, observe que as cadeias $C(x,x') = [x = x_0, x_1, \ldots, x_l = x']$ e sua inversa $C(x',x) \coloneqq [x' = x_l, x_{l-1}, \ldots, x_0 = x]$ são diferentes.

Definição 1.9. *Dada uma cadeia $C(x,x') = [x = x_0, x_1, \ldots, x_l = x']$, definimos o custo da cadeia como sendo*

$$\max_{i | x_i \in C(x,x')} A_X(x_i, x_{i+1}).$$

O custo mínimo de cadeias entre x e x' é definido por

$$\widetilde{u}^*_X(x,x') \coloneqq \min_{C(x,x')} \max_{i | x_i \in C(x,x')} A_X(x_i, x_{i+1}), \tag{1.2}$$

ou seja, é o custo mínimo entre todas as cadeias que conectam x a x'.

O uso da notação $\widetilde{u}^*_X$ se deve ao fato de que no Capítulo 2 veremos outro conceito que utiliza u_X como notação.

Em *networks* assimétricas, os custos mínimos de cadeia $\widetilde{u}^*_X(x,x')$ e $\widetilde{u}^*_X(x',x)$ em geral são diferentes, mas são iguais em *networks* simétricas. Nesse caso, os custos

$\widetilde{u}^*_X(x,x') = \widetilde{u}^*_X(x',x)$ fazem parte da definição do dendrograma *single-linkage*, que veremos mais adiante.

Definimos um *ciclo* como sendo uma cadeia da forma $C(x,x)$, para algum $x \in X$, tal que $C(x,x)$ contenha pelo menos um nó além de x. Como um ciclo é uma cadeia particular, o custo do ciclo é dado na Definição 1.9. Além disso, definimos o *custo mínimo de ciclos* da *network* (X, A_X) por

$$\operatorname{mlc}(X, A_X) := \min_x \min_{C(x,x)} \max_{i|x_i \in C(x,x)} A_X(x_i, x_{i+1}). \tag{1.3}$$

Definição 1.10. *A separação de uma* network (X, A_X) *é definida por*

$$\operatorname{sep}(X, A_X) := \min_{x \neq x'} A_X(x, x'), \tag{1.4}$$

ou seja, é a menor dissimilaridade positiva.

Note que, por (1.2) e (1.4), temos $\widetilde{u}^*_X(x,x') \geq \operatorname{sep}(X, A_X)$. Assim, por (1.3),

$$\operatorname{sep}(X, A_X) \leq \operatorname{mlc}(X, A_X). \tag{1.5}$$

Além disso, no caso de *networks* simétricas, temos a igualdade em (1.5), pois

$$\begin{aligned}
\operatorname{mlc}(X, A_X) &= \min_x \min_{C(x,x)} \max_{i|x_i \in C(x,x)} A_X(x_i, x_{i+1}) \\
&= \min_x \min_{C(x,x)} \max_{i|x_i \in C(x,x)} A_X(x_{i+1}, x_i) \\
&\leq \min_{\substack{C(x,x') \uplus C(x',x) \\ x \neq x'}} \max A_X(x_i, x_{i+1}) \\
&\leq \min_{x \neq x'} A_X(x, x') \\
&= \operatorname{sep}(X, A_X).
\end{aligned}$$

Exemplo 1.11. Considere a *network* $N_X = (X, A_X)$, com conjunto de nós $X = \{a, b, c, d\}$, representada pela Figura 1.3.

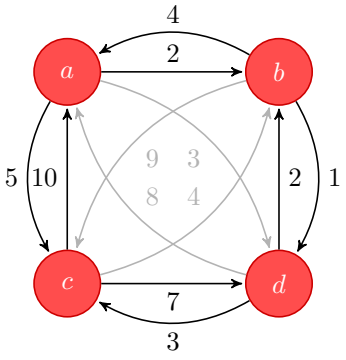


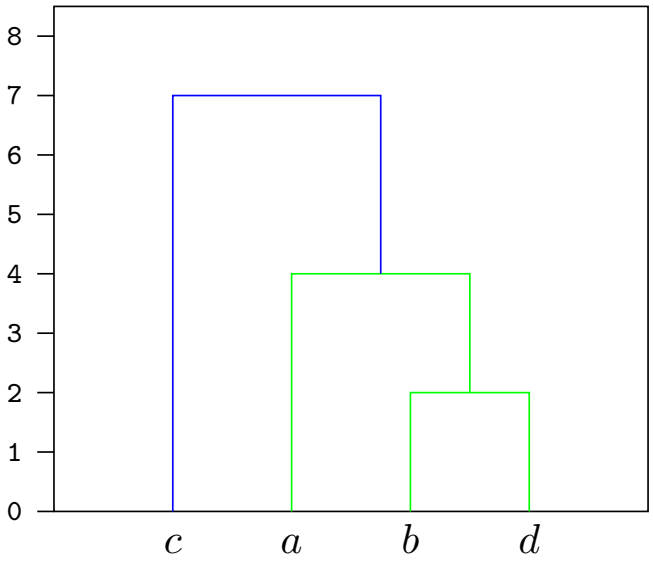
Figura 1.3: *Network* assimétrica de quatro nós

A função dissimilaridade é representada pela matriz

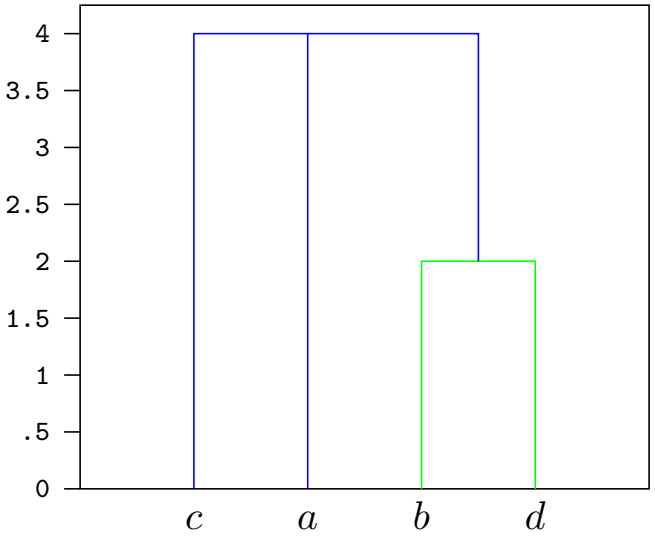
$$A_X = \begin{bmatrix} 0 & 2 & 5 & 3 \\ 4 & 0 & 9 & 1 \\ 10 & 4 & 0 & 7 \\ 8 & 2 & 3 & 0 \end{bmatrix},$$

da qual vemos que $\text{sep}(X, A_X) = 1$.

As Figuras 1.4a e 1.4b representam os dendrogramas recíproco e não-recíproco (que veremos mais à frente) que podem ser derivados da *network* acima.



(a) Dendrograma recíproco



(b) Dendrograma não-recíproco

Figura 1.4: Exemplo de dendrogramas

2 Axiomas do Valor e da Transformação

Neste capítulo, estudaremos os Axiomas do Valor e da Transformação e ultramétricas, que se fazem necessários para podermos estudar métodos de *Clustering* Hierárquico.

Lembremos da definição de $\overrightarrow{\Delta}_2(\alpha, \beta)$ vista em (1.1),

$$\overrightarrow{\Delta}_2(\alpha, \beta) = (\{p, q\}, A_{p,q}), \text{ em que } A_{p,q}(p, q) = \alpha \text{ e } A_{p,q}(q, p) = \beta.$$

Definição 2.1. *Dadas duas* networks $N_X = (X, A_X)$ *e* $N_Y = (Y, A_Y)$*, uma função* $\phi : X \to Y$ *é chamada redução de dissimilaridade se*

$$A_X(x, x') \geq A_Y(\phi(x), \phi(x')), \ \forall x, x' \in X.$$

(A1) *Axioma do Valor.* Dizemos que $\mathcal{H}$ satisfaz o Axioma do Valor se o dendrograma $D_{p,q} = \mathcal{H}(\overrightarrow{\Delta}_2(\alpha, \beta))$ produzido por $\mathcal{H}$ aplicado à *network* $\overrightarrow{\Delta}_2(\alpha, \beta)$ satisfaz

$$D_{p,q}(\delta) = \begin{cases} \{\{p\}, \{q\}\}, & \text{para } 0 \leq \delta < \max(\alpha, \beta), \\ \{\{p, q\}\}, & \text{para } \delta \geq \max(\alpha, \beta). \end{cases}$$

(A2) *Axioma da Transformação.* Considere duas *networks* $N_X = (X, A_X)$ e $N_Y = (Y, A_Y)$ e uma função redução de dissimilaridade $\phi : X \to Y$. Dizemos que o método $\mathcal{H}$ satisfaz o Axioma da Transformação se $D_X = \mathcal{H}((X, A_X))$ e $D_Y = \mathcal{H}((Y, A_Y))$ são dendrogramas tais que

$$\text{se } x \sim_{D_X(\delta)} x' \text{ então } \phi(x) \sim_{D_Y(\delta)} \phi(x'), \text{ com } \delta \geq 0.$$

Dizemos que um método de *clustering* hierárquico $\mathcal{H}$ é *admissível* se valem os axiomas (A1) e (A2) para $\mathcal{H}$.

Definição 2.2. *Seja* X *um conjunto de nós. Uma ultramétrica em* X *é uma função não-negativa* $u_X : X \times X \to \mathbb{R}_+$ *satisfazendo:*

(i) *Identidade.* $u_X(x, x') = 0$ *se, e somente se,* $x = x'$*, para todo* $x, x' \in X$*.*

(ii) *Simetria.* $u_X(x, x') = u_X(x', x)$*, para todo* $x, x' \in X$*.*

(iii) *Desigualdade triangular forte. Para quaisquer* $x, x', x'' \in X$*,*

$$u_X(x, x'') \leq \max(u_X(x, x'), u_X(x', x'')). \tag{2.1}$$

Como (2.1) implica que $u_X(x,x'') \leq u_X(x,x')+u_X(x',x'')$, para quaisquer $x,x',x'' \in X$, segue que conjuntos ultramétricos são casos particulares de conjuntos métricos.

O interesse principal em estudar ultramétricas é estabelecer uma estrutura que preserva bijeção entre o conjunto dos dendrogramas e o conjunto das ultramétricas.

A seguir, definiremos uma função $\Psi : \mathcal{D} \to \mathcal{U}$ do conjunto $\mathcal{D}$ dos dendrogramas no conjunto $\mathcal{U}$ das *networks* dotadas com ultramétricas. Mais precisamente, dado um dendrograma D_X, seja $\Psi(D_X) = (X, u_X)$, onde a ultramétrica u_X é dada por

$$u_X(x,x') \coloneqq \min\{\delta \geq 0 \mid x \sim_{D_X(\delta)} x'\}. \tag{2.2}$$

Por outro lado, definiremos também uma função $\Upsilon : \mathcal{U} \to \mathcal{D}$. Dada uma ultramétrica u_X no conjunto finito X e dado $\delta \geq 0$, seja a relação $\sim_{u_X(\delta)}$ em X dada por

$$x \sim_{u_X(\delta)} x' \Leftrightarrow u_X(x,x') \leq \delta. \tag{2.3}$$

Daí, tomamos $D_X(\delta) \coloneqq X \bmod \sim_{u_X(\delta)}$ e definimos $\Upsilon(X,u_X) \coloneqq D_X$.

Isto posto, tem-se:

Teorema 2.3. *As funções $\Psi : \mathcal{D} \to \mathcal{U}$ e $\Upsilon : \mathcal{U} \to \mathcal{D}$ estão bem definidas. Além disso, $\Psi \circ \Upsilon$ é a identidade em $\mathcal{U}$ e $\Upsilon \circ \Psi$ é a identidade em $\mathcal{D}$.*

Demonstração. Será feita em duas partes.

Parte 1. Primeiramente, mostremos que Ψ está bem definida. Para isso, basta mostrar que u_X é uma ultramétrica.

Notemos que o mínimo em (2.2) existe. De fato, seja $\gamma = \inf\{\delta \geq 0 \mid x \sim_{D_X(\delta)} x'\}$ e suponhamos que $\gamma \notin \{\delta \geq 0 \mid x \sim_{D_X(\delta)} x'\}$. Por (D3) aplicado a γ temos que existe $\epsilon > 0$ tal que $D_X(\delta) = D_X(\gamma)$, para todo $\delta \in [\gamma, \gamma+\epsilon]$. Em particular, $D_X(\gamma) = D_X(\gamma+\epsilon)$, implicando que $\gamma+\epsilon$ é cota inferior do conjunto acima, o que contradiz o fato de γ ser o ínfimo de tal conjunto. Portanto, γ é elemento mínimo.

Agora, provemos as condições (i)-(iii) da Definição 2.2.

(i) Temos que $u_X(x,x') = 0 \Leftrightarrow x \sim_{D_X(0)} x' \Leftrightarrow x = x'$, pois $\sim_{D_X(\delta)}$ é uma relação de equivalência.

(ii) Como $\sim_{D_X(\delta)}$ é uma relação de equivalência, temos que $x \sim_{D_X(\delta)} x' \Leftrightarrow x' \sim_{D_X(\delta)} x$. Logo, $u_X(x,x') = u_X(x',x)$.

(iii) Dados $x,x',x'' \in X$, consideremos $u_X(x,x') = \delta_1$ e $u_X(x',x'') = \delta_2$. Tomando $\delta = \max\{\delta_1,\delta_2\}$, temos que $x \sim_{D_X(\delta_1)} x'$ implica $x \sim_{D_X(\delta)} x'$ e que $x' \sim_{D_X(\delta_2)} x''$ implica $x' \sim_{D_X(\delta)} x''$. Pela transitividade da relação $\sim_{D_X(\delta)}$, segue que $x \sim_{D_X(\delta)} x''$, ou seja, $u_X(x,x'') \leq \delta$.

Logo, u_X é uma ultramétrica em X e Ψ é bem definida.

A seguir, para provarmos que Υ está bem definida, primeiro mostremos que $\sim_{u_X(\delta)}$ dada em (2.3) é uma relação de equivalência, e portanto induz uma partição do conjunto X, para cada $\delta \geq 0$.

- $\sim_{u_X(\delta)}$ é reflexiva, ou seja, $x \sim_{u_X(\delta)} x$, pois $0 = u_X(x,x) \leq \delta$, $\forall \delta \geq 0$.
- $\sim_{u_X(\delta)}$ é simétrica, pois dados $x,x' \in X$, se $x \sim_{u_X(\delta)} x'$ então $u_X(x,x') \leq \delta$, e como u_X é ultramétrica, $u_X(x,x') = u_X(x',x)$. Portanto $x' \sim_{u_X(\delta)} x$.

- $\sim_{u_X(\delta)}$ é transitiva. De fato, sejam $x, x', x'' \in X$ tais que $x \sim_{u_X(\delta)} x'$ e $x' \sim_{u_X(\delta)} x''$. Então $u_X(x, x'), u_X(x', x'') \leq \delta$. Pela desigualdade triangular forte, temos que $u_X(x, x'') \leq \max\{u_X(x, x'), u_X(x', x'')\} \leq \delta$, de onde segue que $x \sim_{u_X(\delta)} x''$.

Agora, mostremos que $D_X = \{D_X(\delta) \mid \delta \geq 0\}$ é um dendrograma.

(D1) Para $\delta = 0$, temos que $u_X(x, x') = 0$ se, e somente se, $x = x'$, $\forall x, x' \in X$. Logo, $D_X(0) = \{\{x\},\ x \in X\}$. Tomando $\delta_0 = \max_{x \neq x'}\{u_X(x, x')\}$, temos que $u_X(x, x') \leq \delta_0$, $\forall x, x' \in X$. Logo, $x \sim_{u_X(\delta_0)} x'$, $\forall x, x' \in X$, isto é, $D_X(\delta_0) = \{X\}$.

(D2) Para $\delta_1 \leq \delta_2$ e $x, x' \in X$ tais que $x \sim_{u_X(\delta_1)} x'$, temos que $u_X(x, x') \leq \delta_1 \leq \delta_2$. Logo $x \sim_{u_X(\delta_2)} x'$.

(D3) Para cada $\delta \geq 0$ tal que $D_X(\delta) \neq \{X\}$, tomamos $\epsilon(\delta)$ um número positivo qualquer tal que $0 < \epsilon(\delta) < m - \delta$, onde

$$m = \min_{x,x' \in X}\{u_X(x, x') \mid u_X(x, x') > \delta\}.$$

Note que a finitude de X garante a existência de $\epsilon(\delta)$. Daí, para qualquer $\delta' \in [\delta, \delta + \epsilon(\delta)]$, se $x \sim_{u_X(\delta)} x'$ então $x \sim_{u_X(\delta')} x'$, ou seja, $u_X(x, x') \leq \delta' \leq \delta + \epsilon(\delta) < m$. De fato, suponha, por absurdo, que $u_X(x, x') > \delta$, então $u_X(x, x') > m$, o que é uma contradição.

Para $D_X(\delta) = \{X\}$, a propriedade (D3) é satisfeita trivialmente, pois o dendrograma se mantém inalterado para qualquer parâmetro suficientemente grande.

Em resumo, como $\sim_{u_X(\delta)}$ é relação de equivalência e D_X é um dendrograma, segue que Υ está bem definida.

Parte 2. Mostremos que $\Psi \circ \Upsilon = \mathrm{id}$. Considere um conjunto ultramétrico (X, u_X) e escreva $\delta_0 = u_X(x, x')$, para $x, x' \in X$ dados. Sendo $(X, u_X^*) = \Psi \circ \Upsilon(X, u_X)$, basta mostrar que $u_X(x, x') = u_X^*(x, x')$.

Para isso, se $\delta < \delta_0$, x e x' não se agrupam e para $\delta = \delta_0$, se agrupam em um único *cluster*, pois $u_X(x, x') = \delta_0$. Assim, obtemos que $\delta_0 = \min\{\delta \geq 0 \mid x \sim_{u_X^*(\delta)} x'\}$, ou seja, $u_X^*(x, x') = \delta_0 = u_X(x, x')$.

Finalmente, mostremos que $\Upsilon \circ \Psi = \mathrm{id}$. Para isso, considere D_X um dendrograma com a relação de equivalência $\sim_{D_X(\delta)}$ e $(X, u_X) = \Psi(D_X)$. Seja $D_X^* = \Upsilon \circ \Psi(D_X)$, cuja relação de equivalência $\sim_{D_X^*}$ proveniente da ultramétrica u_X, para cada parâmetro $\delta \geq 0$, é dada por

$$x \sim_{D_X^*(\delta)} x' \Leftrightarrow u_X(x, x') \leq \delta.$$

Assim, claramente as relações $\sim_{D_X(\delta)}$ e $\sim_{D_X^*(\delta)}$ são equivalentes, já que ambas são definidas a partir da mesma ultramétrica u_X. Logo, os dendrogramas correspondentes são iguais, isto é, $D_X = D_X^*$. □

Dada a equivalência entre o conjunto dos dendrogramas e o das ultramétricas estabelecida no Teorema 2.3, agora podemos considerar os métodos de *clustering* hierárquicos $\mathcal{H}$ com as ultramétricas induzidas no conjunto X baseado nas funções dissimilaridades A_X.

Contudo, ultramétricas são casos particulares de funções dissimilaridades e assim podemos reinterpretar o método $\mathcal{H}$ como a função

$$\mathcal{H} : \mathcal{N} \to \mathcal{U}$$

onde $u(x, x')$ induzida por $\mathcal{H}$ é o parâmetro mínimo para o qual x e x' serão agrupados por $\mathcal{H}$, para $x, x' \in X$.

Dois métodos $\mathcal{H}_1$ e $\mathcal{H}_2$ são equivalentes se, e somente se, $\mathcal{H}_1(N) = \mathcal{H}_2(N)$, para qualquer $N \in \mathcal{N}$.

(A1) *Axioma do Valor.* A imagem de $\mathcal{H}$ aplicada à *network* de dois nós satisfaz

$$u_{p,q}(p, q) = \max(\alpha, \beta).$$

(A2) *Axioma da Transformação.* Considere duas *networks* $N_X = (X, A_X)$ e $N_Y = (Y, A_Y)$ e uma função redução de dissimilaridade $\phi : X \to Y$. Então para quaisquer $x, x' \in X$, as imagens $(X, u_X) = \mathcal{H}((X, A_X))$ e $(Y, u_Y) = \mathcal{H}((Y, A_Y))$ satisfazem

$$u_X(x, x') \geq u_Y(\phi(x), \phi(x')).$$

Para o caso particular de *networks* simétricas, definimos o dendrograma *single-linkage* SL_X através da relação de equivalência

$$x \sim_{SL_X(\delta)} x' \Leftrightarrow \widetilde{u}^*_X(x, x') = \widetilde{u}^*_X(x', x) \leq \delta. \tag{2.4}$$

Em outras palavras, para $\delta \geq 0$, o método *single-linkage* faz com que x e x' façam parte do mesmo *cluster* se, e somente se, o custo mínimo de cadeia entre x e x' não exceda δ.

Pelo Teorema 2.3, existe uma equivalência entre o dendrograma proveniente da relação (2.4) e o conjunto ultramétrico, que denotamos por (X, u^{SL}_X), em que

$$u^{SL}_X(x, x') = \widetilde{u}^*_X(x, x') = \widetilde{u}^*_X(x', x). \tag{2.5}$$

Para parâmetros $0 \leq \delta < \text{mlc}(X, A_X)$ é impossível encontrar cadeias de influência mútua com custo máximo menor que δ entre quaisquer dois nós de X. De fato, suponhamos, por absurdo, que possamos ligar x a x' em uma cadeia com custo máximo menor ou igual que δ. Da mesma forma, podemos ligar x' a x. Assim, concatenando essas cadeias podemos formar um ciclo com custo menor ou igual a δ.

Assim, temos a seguinte propriedade:

(P1) *Propriedade de Influência.* Para qualquer *network* $N_X = (X, A_X)$, a imagem $(X, u_X) = \mathcal{H}((X, A_X))$ correspondente à função do método de *clustering* hierárquico é tal que a ultramétrica $u_X(x, x')$ entre quaisquer dois nós x e x' não pode ser menor que o custo mínimo de ciclo $\text{mlc}(X, A_X)$ da *network*, ou seja,

$$u_X(x, x') \geq \text{mlc}(X, A_X), \ \forall x \neq x'.$$

Uma segunda afirmação intuitiva sobre influência em *networks* de tamanho arbitrário é formalizada na forma do Axioma do Valor Estendido.

Para isso, vamos introduzir o seguinte conceito.

Definição 2.4. *Defina uma família canônica de* networks *assimétricas, representada na Figura 2.1, por*

$$\overrightarrow{\Delta}_n(\alpha, \beta) \coloneqq (\{1, \ldots, n\}, A_{n,\alpha,\beta}), \tag{2.6}$$

com $n \in \mathbb{N}$ e $\alpha, \beta > 0$, e para $i < j$, o valor da dissimilaridade é $A_{n,\alpha,\beta}(i, j) = \alpha$ e para $i > j$, a dissimilaridade é $A_{n,\alpha,\beta}(i, j) = \beta$.

Na forma matricial, temos

$$A_{n,\alpha,\beta} = \begin{bmatrix} 0 & \alpha & \alpha & \alpha & \cdots & \alpha \\ \beta & 0 & \alpha & \alpha & \cdots & \alpha \\ \beta & \beta & 0 & \alpha & \cdots & \alpha \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ \beta & \beta & \beta & \cdots & 0 & \alpha \\ \beta & \beta & \beta & \cdots & \beta & 0 \end{bmatrix}$$

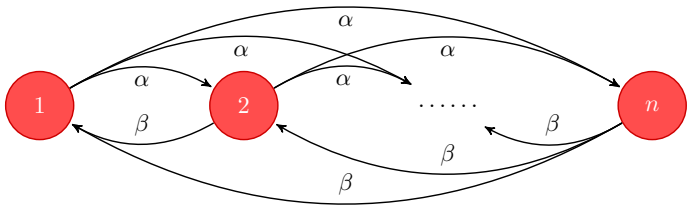


Figura 2.1: *Network* canônica

Agora, considere uma permutação $\Pi = \{\pi_1, \pi_2, \ldots, \pi_n\}$ de $\{1, \ldots, n\}$ e a ação $\Pi(A)$ de Π em uma matriz $A_{n\times n}$, que definimos por $(\Pi(A))_{i,j} = A_{\pi_i,\pi_j}$, $\forall i, j$. Definindo a *network* $\overrightarrow{\Delta}_n(\alpha, \beta, \Pi) := (\{1, \ldots, n\}, \Pi(A_{n,\alpha,\beta}))$, podemos introduzir o conceito do Axioma do Valor Estendido.

(A1′) *Axioma do Valor Estendido.* Seja $\overrightarrow{\Delta}_n(\alpha, \beta, \Pi) := (\{1, \ldots n\}, \Pi(A_{n,\alpha,\beta}))$. Então, para todos os índices $n \in \mathbb{N}$, constantes $\alpha, \beta > 0$, e permutações Π de $\{1, \ldots, n\}$, a saída $(\{1, \ldots, n\}, u_{n,\alpha,\beta}) = \mathcal{H}(\overrightarrow{\Delta}_n(\alpha, \beta, \Pi))$ do método de *clustering* hierárquico $\mathcal{H}$ satisfaz

$$u_{n,\alpha,\beta}(i, j) = \max(\alpha, \beta), \ \forall i \neq j.$$

Observe que o Axioma do Valor (A1) é um caso particular do Axioma do Valor Estendido (A1′) para $n = 2$. Além disso, note que $\text{mlc}(\overrightarrow{\Delta}_n(\alpha, \beta)) = \max(\alpha, \beta)$ pois, para formar um ciclo nessa *network*, é necessário atravessar pelo menos um *link* de custo α e pelo menos um *link* de custo β, como mostra a Figura 2.1. Como a permutação não altera o custo de um ciclo, então

$$\text{mlc}(\overrightarrow{\Delta}_n(\alpha, \beta, \Pi)) = \text{mlc}(\overrightarrow{\Delta}_n(\alpha, \beta)) = \max(\alpha, \beta). \tag{2.7}$$

2.1 Algumas equivalências

Nos próximos teoremas, veremos que os Axiomas do Valor e do Valor Estendido são equivalentes quando o Axioma da Transformação for válido. Além disso, veremos que se o método $\mathcal{H}$ for admissível, então a Propriedade da Influência se verifica. Para isso, começaremos demonstrando o seguinte lema.

Lema 2.5. *Seja $N = (X, A_X)$ uma* network *com n nós e $\delta > 0$ uma constante. Suponha que $x, x' \in X$ são tais que o custo mínimo de cadeia associado satisfaz*

$$\widetilde{u}^*_X(x, x') \geq \delta. \tag{2.8}$$

Então existe uma partição $P_\delta(x,x') = \{B_\delta(x), B_\delta(x')\}$ *do conjunto* X *em* clusters $B_\delta(x) \ni x$ *e* $B_\delta(x') \ni x'$ *tais que, para todos os pontos* $b \in B_\delta(x)$ *e* $b' \in B_\delta(x')$,

$$A_X(b,b') \geq \delta. \tag{2.9}$$

Demonstração. Suponhamos, por absurdo, que não exista uma partição $P_\delta(x,x') = \{B_\delta(x), B_\delta(x')\}$ com $x \in B_\delta(x)$ e $x' \in B_\delta(x')$ satisfazendo (2.9), para quaisquer $x, x' \in X$ satisfazendo (2.8). Então existe pelo menos um par de nós $x, x' \in X$ satisfazendo (2.8) tal que para toda partição de X em dois *clusters* $P = \{B, B'\}$ com $x \in B$ e $x' \in B'$, podemos encontrar pelo menos um par de elementos $b_P \in B$ e $b'_P \in B'$ para os quais

$$A_X(b_P, b'_P) < \delta. \tag{2.10}$$

Consideremos, primeiramente, a partição $P_1 = \{B_1, B'_1\}$, onde $B_1 = \{x\}$ e $B'_1 = X - B_1$. Como (2.10) é verdadeiro para todas as partições e x é o único elemento de B_1, então existe $b'_{P_1} \in B'_1$ tal que

$$A_X(x, b'_{P_1}) < \delta. \tag{2.11}$$

Assim, a cadeia $C(x, b'_{P_1}) = [x, b'_{P_1}]$ composta por esses dois nós tem custo menor que δ. Além disso, como $\widetilde{u}^*_X(x, b'_{P_1})$ representa o custo mínimo sobre todas as cadeias que ligam x a b'_{P_1}, podemos concluir que

$$\widetilde{u}^*_X(x, b'_{P_1}) \leq A_X(x, b'_{P_1}) < \delta.$$

A seguir, consideremos a partição $P_2 = \{B_2, B'_2\}$, onde $B_2 = \{x, b'_{P_1}\}$ e $B'_2 = X - B_2$. Por (2.10), existe um nó $b'_{P_2} \in B'_2$ que satisfaz pelo menos uma das seguintes condições:

$$A_X(x, b'_{P_2}) < \delta, \tag{2.12}$$

$$A_X(b'_{P_1}, b'_{P_2}) < \delta. \tag{2.13}$$

Se (2.12) for verdadeira, a cadeia $C(x, b'_{P_2}) = [x, b'_{P_2}]$ tem custo menor que δ. Se (2.13) for verdadeira, combinando com a afirmação (2.11), temos que a cadeia $C(x, b'_{P_2}) = [x, b'_{P_1}, b'_{P_2}]$ tem custo menor que δ. Em ambos os casos, concluímos que existe uma cadeia $C(x, b'_{P_2})$ ligando x a b'_{P_2} cujo custo é menor que δ. Consequentemente, o custo mínimo de cadeia deve satisfazer

$$\widetilde{u}^*_X(x, b'_{P_2}) < \delta.$$

Consideremos, agora, a partição $P_3 = \{B_3, B'_3\}$ com $B_3 = \{x, b'_{P_1}, b'_{P_2}\}$ e $B'_3 = X - B_3$. Segue de (2.10) que existe um ponto b'_{P_3} tal que pelo menos uma das dissimilaridades $A_X(x, b'_{P_3})$, $A_X(b'_{P_1}, b'_{P_3})$ ou $A_X(b'_{P_2}, b'_{P_3})$ é menor que δ, e assim como foi argumentado em (2.12) e (2.13), existe uma cadeia $C(x, b'_{P_3})$ ligando x a b'_{P_3} cujo custo é menor que δ e daí segue que

$$\widetilde{u}^*_X(x, b'_{P_3}) < \delta.$$

Essa construção pode ser repetida $n-1$ vezes, obtendo-se partições $P_1, P_2, \ldots, P_{n-1}$ com os nós correspondentes $b'_{P_1}, b'_{P_2}, \ldots, b'_{P_{n-1}}$ tais que o custo mínimo de cadeia satisfaz

$$\widetilde{u}^*_X(x, b'_{P_i}) < \delta, \quad 1 \leq i \leq n-1. \tag{2.14}$$

Observe que, por construção, os nós b'_{P_i} são distintos entre si e também distintos de x. Como a *network* tem n nós, temos que $x' = b'_{P_k}$ para algum $k \in \{1, \ldots, n-1\}$. Segue de (2.14) que

$$\widetilde{u}^*_X(x, x') < \delta,$$

o que é uma contradição, pois x e x' satisfazem (2.8). □

Teorema 2.6. *Seja $\mathcal{H}$ um método de* clustering *hierárquico que satisfaz o Axioma da Transformação* (A2). *Então $\mathcal{H}$ satisfaz o Axioma do Valor* (A1) *se, e somente se, $\mathcal{H}$ satisfaz o Axioma do Valor Estendido* (A1′).

Demonstração. Suponhamos que $\mathcal{H}$ satisfaz o Axioma do Valor Estendido (A1′). Portanto $\mathcal{H}$ satisfaz o Axioma do Valor (A1), pois o Axioma (A1) é um caso particular do Axioma (A1′).

Reciprocamente, considere $\mathcal{H}$ um método satisfazendo (A1) e (A2), $\overrightarrow{\Delta}_n(\alpha, \beta, \Pi)$ a *network* dada em (A1′) e $(\{1, \ldots, n\}, u) = \mathcal{H}(\overrightarrow{\Delta}_n(\alpha, \beta, \Pi))$ sua imagem por $\mathcal{H}$.

Temos que mostrar que, para todo índice $n \in \mathbb{N}$, constantes $\alpha, \beta > 0$, permutações Π de $\{1, \ldots, n\}$ e pontos $i \neq j$, $u_{n,\alpha,\beta}(i, j) = \max(\alpha, \beta)$. Assim, mostraremos

$$u_{n,\alpha,\beta}(i, j) \leq \max(\alpha, \beta), \tag{2.15}$$

$$u_{n,\alpha,\beta}(i, j) \geq \max(\alpha, \beta). \tag{2.16}$$

Para (2.15), defina uma *network* de dois nós simétrica $\overrightarrow{\Delta}_2(\max(\alpha, \beta), \max(\alpha, \beta)) = (\{p, q\}, A_{p,q})$, onde $A_{p,q}(p, q) = A_{p,q}(q, p) = \max(\alpha, \beta)$ e denote por $(\{p, q\}, u_{p,q}) = \mathcal{H}(\overrightarrow{\Delta}_2(\max(\alpha, \beta), \max(\alpha, \beta)))$. Como $\mathcal{H}$ satisfaz (A1),

$$u_{p,q}(p, q) = \max(\max(\alpha, \beta), \max(\alpha, \beta)) = \max(\alpha, \beta). \tag{2.17}$$

Considere, agora, a função $\phi_{i,j} : \{p, q\} \to \{1, \ldots, n\}$, onde $\phi_{i,j}(p) = i$ e $\phi_{i,j}(q) = j$. Como as dissimilaridades em $\overrightarrow{\Delta}_n(\alpha, \beta, \Pi)$ são α ou β e as dissimilaridades em $\overrightarrow{\Delta}_2(\max(\alpha, \beta), \max(\alpha, \beta))$ são $\max(\alpha, \beta)$, segue que a função $\phi_{i,j}$ é uma redução de dissimilaridade, independentemente dos valores i, j. Como o método $\mathcal{H}$ satisfaz (A2), temos que

$$u_{p,q}(p, q) \geq u_{n,\alpha,\beta}(\phi_{i,j}(p), \phi_{i,j}(q)) = u_{n,\alpha,\beta}(i, j). \tag{2.18}$$

Substituindo (2.17) em (2.18), segue a desigualdade (2.15).

Para mostrar a desigualdade (2.16), considere dois nós arbitrários distintos $i, j \in \{1, \ldots, n\}$ da *network* $\overrightarrow{\Delta}_n(\alpha, \beta, \Pi)$. Denote por $C(i, j)$ e $C(j, i)$ duas cadeias cujo custo é mínimo, ou seja, são as cadeias que realizam o custo mínimo de cadeias $\widetilde{u}^*_{n,\alpha,\beta}(i, j)$ e $\widetilde{u}^*_{n,\alpha,\beta}(j, i)$, respectivamente.

Observe que deve ser verdadeira pelo menos uma das desigualdades:

$$\widetilde{u}^*_{n,\alpha,\beta}(i, j) \geq \max(\alpha, \beta), \tag{2.19}$$

$$\widetilde{u}^*_{n,\alpha,\beta}(j, i) \geq \max(\alpha, \beta). \tag{2.20}$$

De fato, se ambas forem falsas, a concatenação de $C(i, j)$ e $C(j, i)$ formaria um ciclo $C(i, i) = C(i, j) \uplus C(j, i)$ de custo estritamente menor que $\max(\alpha, \beta)$, o que não pode ocorrer, pois $\mathrm{mlc}(\overrightarrow{\Delta}_n(\alpha, \beta, \Pi)) = \max(\alpha, \beta)$ por (2.7).

Sem perda de generalidade, assuma que vale (2.19) e considere $\delta = \max(\alpha, \beta)$. Pelo Lema 2.5, podemos garantir que existe uma partição do conjunto de nós $\{1, \ldots, n\}$ em dois blocos $B_\delta(i)$ e $B_\delta(j)$, com $i \in B_\delta(i)$ e $j \in B_\delta(j)$ tais que para todo $b \in B_\delta(i)$ e $b' \in B_\delta(j)$, temos

$$\Pi(A_{n,\alpha,\beta})(b, b') \geq \delta = \max(\alpha, \beta). \tag{2.21}$$

Defina uma *network* de dois nós $\overrightarrow{\Delta}_2(\max(\alpha, \beta), \min(\alpha, \beta)) = (\{r, s\}, A_{r,s})$, em que $A_{r,s}(r, s) = \max(\alpha, \beta)$ e $A_{r,s}(s, r) = \min(\alpha, \beta)$ e escreva

$$(\{r, s\}, u_{r,s}) = \mathcal{H}(\overrightarrow{\Delta}_2(\max(\alpha, \beta), \min(\alpha, \beta))).$$

Como $\mathcal{H}$ satisfaz (A1), temos

$$u_{r,s}(r,s) = \max(\max(\alpha,\beta), \min(\alpha,\beta)) = \max(\alpha,\beta). \tag{2.22}$$

Considere a função $\phi'_{i,j} : \{1,\ldots,n\} \to \{r,s\}$ tal que $\phi'_{i,j}(b) = r$ e $\phi'_{i,j}(b') = s$, para todo $b \in B_\delta(i)$ e $b' \in B_\delta(j)$. A função $\phi'_{i,j}$ é redução de dissimilaridade, pois

$$\Pi(A_{n,\alpha,\beta})(k,l) \geq A_{r,s}(\phi'_{i,j}(k), \phi'_{i,j}(l)),\ \forall k,l \in \{1,\ldots,n\}. \tag{2.23}$$

Com efeito, considere três possíveis casos.

- Se k e l pertencem a um mesmo bloco, isto é, $k,l \in B_\delta(i)$ ou $k,l \in B_\delta(j)$, então $\phi'_{i,j}(k) = \phi'_{i,j}(l)$ e $A_{r,s}(\phi'_{i,j}(k), \phi'_{i,j}(l)) = 0$, e portanto não pode ser maior que $\Pi(A_{n,\alpha,\beta})(k,l)$.
- Se $k \in B_\delta(j)$ e $l \in B_\delta(i)$, segue que $A_{r,s}(\phi'_{i,j}(k), \phi'_{i,j}(l)) = A_{r,s}(s,r) = \min(\alpha,\beta)$. Logo, não pode exceder $\Pi(A_{n,\alpha,\beta})(k,l)$, que é igual a α ou β.
- Se $k \in B_\delta(i)$ e $l \in B_\delta(j)$, temos $A_{r,s}(\phi'_{i,j}(k), \phi'_{i,j}(l)) = A_{r,s}(r,s) = \max(\alpha,\beta)$, mas por (2.21), temos que $\Pi(A_{n,\alpha,\beta})(k,l) \geq \delta = \max(\alpha,\beta)$.

Como $\mathcal{H}$ satisfaz (A2), segue que

$$u_{n,\alpha,\beta}(i,j) \geq u_{r,s}(\phi'_{i,j}(i), \phi'_{i,j}(j)) = u_{r,s}(r,s). \tag{2.24}$$

Substituindo (2.22) em (2.24), obtemos a desigualdade (2.16), o que completa a prova. □

Lema 2.7. *Sejam $N = (X, A_X)$ uma* network *arbitrária com n nós e $\overrightarrow{\Delta}_n(\alpha,\beta) = (\{1,\ldots,n\}, A_{n,\alpha,\beta})$ a* network *canônica com $0 < \alpha \leq \mathrm{sep}(X, A_X)$ e $\beta = \mathrm{mlc}(X, A_X)$. Então existe uma bijeção $\phi : X \to \{1,\ldots,n\}$ tal que*

$$A_X(x,x') \geq A_{n,\alpha,\beta}(\phi(x), \phi(x')),\ \forall x,x' \in X. \tag{2.25}$$

Demonstração. Para construir a bijeção ϕ, consideremos a função $P : X \to \mathcal{P}(X)$ tal que

$$P(x) = \{x' \in X \mid x' \neq x, A_X(x',x) < \beta\}.$$

É importante observar que existe $x \in X$ tal que $P(x) = \emptyset$. De fato, suponhamos, por absurdo, que tal fato não ocorra. Daí, considere $x_n \in X$ um nó qualquer e construa a cadeia $[x_0, x_1, \ldots, x_n]$ onde o i-ésimo elemento da cadeia, x_{i-1}, está na P-imagem de x_i. Da definição da função P, segue que todas as dissimilaridades dessa cadeia satisfazem $A_X(x_{i-1}, x_i) < \beta = \mathrm{mlc}(X, A_X)$. Mas como a cadeia $[x_0, x_1, \ldots, x_n]$ contém $n+1$ elementos, pelo menos um nó deve ser repetido, e assim encontramos um ciclo para o qual todas as dissimilaridades estão limitadas por $\beta = \mathrm{mlc}(X, A_X)$, o que é impossível, pois contradiz a definição de mlc.

Portanto, existe um nó x_{i_1} para o qual $P(x_{i_1}) = \emptyset$. Definimos $\phi(x_{i_1}) = 1$.

Tomemos agora $x_{i_2} \neq x_{i_1}$ cuja P-imagem é $\{x_{i_1}\}$ ou $\emptyset$, ou seja, $P(x_{i_2}) \subseteq \{x_{i_1}\}$. Tal nó existe, pois caso contrário podemos tomar $x_{n-1} \in X - \{x_{i_1}\}$ e construir a cadeia $[x_0, x_1, \ldots, x_{n-1}]$, em que $x_{i-1} \in P(x_i) - \{x_{i_1}\}$, isto é, x_{i-1} está na P-imagem de x_i e $x_{i-1} \neq x_{i_1}$. Como a cadeia $[x_0, x_1, \ldots, x_{n-1}]$ contém n elementos do conjunto $X - \{x_{i_1}\}$ de cardinalidade $n-1$, pelo menos um dos nós se repete. Assim, encontramos um ciclo

onde todas as dissimilaridades entre nós consecutivos satisfazem $A_X(x_{i-1}, x_i) < \beta = \text{mlc}(X, A_X)$, contradizendo a definição de mlc. Definimos $\phi(x_{i_2}) = 2$.

Repetindo esse processo, no k-ésimo passo temos $\phi(x_{i_k}) = k$ para algum nó $x_{i_k} \notin \{x_{i_1}, x_{i_2}, \ldots, x_{i_{k-1}}\}$, cuja P-imagem é um subconjunto dos nós já escolhidos, ou seja, $P(x_{i_k}) \subseteq \{x_{i_1}, x_{i_2}, \ldots, x_{i_{k-1}}\}$. Como todos os nós x_{i_k} são distintos, para todo k, a função ϕ dada por $\phi(x_{i_k}) = k$ é bijetiva.

Por construção, para todo $l > k$, tem-se $x_{i_l} \notin P(x_{i_k})$, e da definição de P, vale

$$A_X(x_{i_l}, x_{i_k}) \geq \beta, \ \forall l > k.$$

Além disso, da definição da dissimilaridade $A_{n,\alpha,\beta}$, segue que

$$A_{n,\alpha,\beta}(\phi(x_{i_l}), \phi(x_{i_k})) = A_{n,\alpha,\beta}(l, k) = \beta, \ \forall l > k.$$

Assim, concluímos que (2.25) é verdadeiro para todos os nós em que $\phi(x) > \phi(x')$. Quando $\phi(x) < \phi(x')$, temos que $A_{n,\alpha,\beta}(\phi(x), \phi(x')) = \alpha \leq \text{sep}(X, A_X) \leq \text{mlc}(X, A_X) = \beta$.

Portanto

$$A_X(x, x') \geq A_{n,\alpha,\beta}(\phi(x), \phi(x')), \ \forall x, x' \in X. \qquad \square$$

Teorema 2.8. *Considere $\mathcal{H}$ um método de* clustering *hierárquico que satisfaz* (A1′) *e* (A2). *Então $\mathcal{H}$ satisfaz* (P1).

Demonstração. Seja $N = (X, A_X)$ uma *network* arbitrária tal que $X = \{x_1, \ldots, x_n\}$ e denote por $(X, u_X) = \mathcal{H}(X, A_X)$. Assim, basta provar que $u_X(x, x') \geq \text{mlc}(X, A_X)$, para todo $x \neq x'$.

Considere a *network* canônica $\overrightarrow{\Delta}_n(\alpha, \beta) = (\{1, \ldots, n\}, A_{n,\alpha,\beta})$ com $\beta = \text{mlc}(X, A_X)$ e $0 < \alpha \leq \text{sep}(X, A_X)$. Assim, temos que $0 < \alpha \leq \text{sep}(X, A_X) \leq \text{mlc}(X, A_X) = \beta$.

Denote por $(\{1, \ldots, n\}, u_{\alpha,\beta}) = \mathcal{H}(\overrightarrow{\Delta}_n(\alpha, \beta))$. Como $\mathcal{H}$ satisfaz (A1′), então para todos os índices $i, j \in \{1, \ldots, n\}$ tal que $i \neq j$, temos

$$u_{\alpha,\beta}(i, j) = \max(\alpha, \beta) = \beta = \text{mlc}(X, A_X). \tag{2.26}$$

Além disso, considere a função redução de dissimilaridade bijetiva do Lema 2.7 e note que, como o método $\mathcal{H}$ satisfaz (A2), para todo $x, x' \in X$ vale

$$u_X(x, x') \geq u_{\alpha,\beta}(\phi(x), \phi(x')).$$

Como a igualdade em (2.26) é verdadeira para todo $i \neq j$ e como ϕ é bijetiva, temos que

$$u_X(x, x') \geq \beta = \text{mlc}(X, A_X),$$

para $x \neq x'$ em X, e portanto $\mathcal{H}$ satisfaz (P1). $\square$

Corolário 2.9. *Considere $\mathcal{H}$ um método de* clustering *hierárquico que satisfaz* (A1) *e* (A2), *então $\mathcal{H}$ satisfaz* (P1).

Demonstração. Pelo Teorema 2.6, temos que se $\mathcal{H}$ satisfaz (A2), então (A1) e (A1′) são equivalentes, e portanto, pelo Teorema 2.8, $\mathcal{H}$ satisfaz (P1). $\square$

3 *Clustering* Recíproco e Não-Recíproco

Neste capítulo, iremos construir dois métodos de *clustering* hierárquico que satisfazem (A1) e (A2), chamados de *clustering* recíproco e não-recíproco.

Seja $N_X = (X, A_X) \in \mathcal{N}$ uma *network* qualquer. Considere a dissimilaridade simétrica

$$\bar{A}_X(x, x') \coloneqq \max\{A_X(x, x'), A_X(x', x)\},\ \forall x, x' \in X. \tag{3.1}$$

Definição 3.1. *Definimos o método de* clustering *recíproco* $\mathcal{H}^R$ *por* $(X, u_X^R) = \mathcal{H}^R(X, A_X)$*, onde a ultramétrica* $u_X^R(x, x')$ *entre os nós* x *e* x' *é dada por*

$$u_X^R(x, x') \coloneqq \min_{C(x,x')} \max_{i|x_i \in C(x,x')} \bar{A}_X(x_i, x_{i+1}). \tag{3.2}$$

Lembremos que, pelo Teorema 2.3, existe uma equivalência entre os dendrogramas e as ultramétricas. Sendo R_X o dendrograma produzido pelo *clustering* recíproco, podemos escrever as classes de equivalência do *clustering* recíproco como

$$x \sim_{R_X(\delta)} x' \Leftrightarrow \min_{C(x,x')} \max_{i|x_i \in C(x,x')} \bar{A}_X(x_i, x_{i+1}) \leq \delta.$$

Proposição 3.2. *O método de* clustering *recíproco* $\mathcal{H}^R$ *é válido e admissível.*

Demonstração. Para provar que $\mathcal{H}^R$ é válido e admissível, temos que mostrar que u_X^R é de fato uma ultramétrica e que valem os axiomas (A1) e (A2).

Primeiro, mostremos que u_X^R é uma ultramétrica. As propriedades (i) e (ii) da Definição 2.2 seguem direto da definição de u_X^R. Para verificar a desigualdade triangular forte, considere $C^*(x, x')$ e $C^*(x', x'')$ duas cadeias que atingem o mínimo em (3.2) para $u_X^R(x, x')$ e $u_X^R(x', x'')$, respectivamente. Assim, o custo máximo na cadeia concatenada $C(x, x'') = C^*(x, x') \uplus C^*(x', x'')$ não excede o custo máximo em cada cadeia individualmente. Portanto, embora o custo máximo possa ser menor em cadeias diferentes, a cadeia $C(x, x'')$ é suficiente para que

$$u_X^R(x, x'') \leq \max(u_X^R(x, x'), u_X^R(x', x'')).$$

Agora, provemos que $\mathcal{H}^R$ satisfaz os Axiomas (A1) e (A2).

Dada uma *network* de dois nós $\overrightarrow{\Delta}_2(\alpha, \beta)$, denote $(\{p, q\}, u_{p,q}^R) = \mathcal{H}^R(\overrightarrow{\Delta}_2(\alpha, \beta))$. Como toda cadeia possível de p para q deve conter p e q como nós consecutivos, temos que

$$u_{p,q}^R(p, q) = \max(A_{p,q}(p, q), A_{p,q}(q, p)) = \max(\alpha, \beta),$$

provando, assim, o Axioma (A1).

A seguir, considere duas *networks* (X, A_X) e (Y, A_Y) quaisquer e uma função redução de dissimilaridade $\phi : X \to Y$. Sejam $(X, u_X^R) = \mathcal{H}^R(X, A_X)$ e $(Y, u_Y^R) = \mathcal{H}^R(Y, A_Y)$. Denote por $C_X^*(x, x') = [x = x_0, x_1, \ldots, x_l = x']$ a cadeia que atinge o mínimo em (3.2), com $x, x' \in X$ arbitrários e considere $C_Y(\phi(x), \phi(x')) = [\phi(x) = \phi(x_0), \phi(x_1), \ldots, \phi(x_l) = \phi(x')]$, uma cadeia em Y.

Como a função ϕ é uma redução de dissimilaridade, temos que $A_Y(\phi(x_i), \phi(x_{i+1})) \leq A_X(x_i, x_{i+1})$ e $A_Y(\phi(x_{i+1}), \phi(x_i)) \leq A_X(x_{i+1}, x_i)$, de onde obtemos

$$\max_{\phi(x_i) \in C_Y(\phi(x), \phi(x'))} \bar{A}_Y(\phi(x_i), \phi(x_{i+1})) \leq u_X^R(x, x').$$

Além disso, note que $C_Y(\phi(x), \phi(x'))$ é uma cadeia particular que liga $\phi(x)$ a $\phi(x')$ e que a ultramétrica recíproca faz uso da cadeia de custo mínimo entre todas as cadeias que ligam $\phi(x)$ a $\phi(x')$. Portanto,

$$u_Y^R(\phi(x), \phi(x')) \leq \max_{\phi(x_i) \in C_Y(\phi(x), \phi(x'))} \bar{A}_Y(\phi(x_i), \phi(x_{i+1})) \leq u_X^R(x, x'),$$

o que conclui a prova. □

Definição 3.3. *Seja $N_X = (X, A_X) \in \mathcal{N}$ uma network qualquer. Definimos o método de clustering não-recíproco $\mathcal{H}^{NR}$ por $(X, u_X^{NR}) = \mathcal{H}^{NR}(X, A_X)$, onde a ultramétrica $u_X^{NR}(x, x')$ entre os nós x e x' é dada por*

$$u_X^{NR}(x, x') \coloneqq \max(\widetilde{u}_X^*(x, x'), \widetilde{u}_X^*(x', x)), \tag{3.3}$$

lembrando que $\widetilde{u}_X^*(x, x') = \min_{C(x,x')} \max_{i | x_i \in C(x,x')} A_X(x_i, x_{i+1})$.

Agora, provaremos uma proposição análoga à Proposição 3.2 para *clustering* não-recíproco.

Proposição 3.4. *O método de* clustering *não-recíproco $\mathcal{H}^{NR}$ é válido e admissível.*

Demonstração. Como na proposição anterior, iniciamos mostrando a desigualdade triangular forte para u_X^{NR}. Para isso, considere as cadeias $C^*(x, x')$ e $C^*(x', x'')$ que atingem o mínimo em $\widetilde{u}_X^*(x, x')$ e $\widetilde{u}_X^*(x', x'')$, bem como as cadeias $C^*(x'', x')$ e $C^*(x', x)$ que atingem o mínimo em $\widetilde{u}_X^*(x'', x')$ e $\widetilde{u}_X^*(x', x)$, respectivamente.

As cadeias concatenadas $C(x, x'') = C^*(x, x') \uplus C^*(x', x'')$ e $C^*(x'', x) = C^*(x'', x') \uplus C^*(x', x)$ não podem exceder o custo máximo em cada cadeia individualmente, e portanto

$$\begin{aligned} u_X^{NR}(x, x'') &= \max(\widetilde{u}_X^*(x, x''), \widetilde{u}_X^*(x'', x)) \\ &\leq \max(\widetilde{u}_X^*(x, x'), \widetilde{u}_X^*(x', x''), \widetilde{u}_X^*(x'', x'), \widetilde{u}_X^*(x', x)) \\ &= \max(u_X^{NR}(x, x'), u_X^{NR}(x', x'')). \end{aligned}$$

Agora mostremos que valem os Axiomas (A1) e (A2).

Considere a *network* de dois nós $\overrightarrow{\Delta}_2(\alpha, \beta)$ e denote $(\{p, q\}, u_{p,q}^{NR}) = \mathcal{H}^{NR}(\overrightarrow{\Delta}_2(\alpha, \beta))$. Note que $\widetilde{u}_{p,q}^*(p, q) = \alpha$ e $\widetilde{u}_{p,q}^*(q, p) = \beta$. Assim, temos

$$u_{p,q}^{NR}(x, x') = \max(\widetilde{u}_{p,q}^*(p, q), \widetilde{u}_{p,q}^*(q, p)) = \max(\alpha, \beta).$$

Finalmente, considere duas *networks* (X, A_X) e (Y, A_Y) quaisquer e uma função redução de dissimilaridade $\phi : X \to Y$. Sejam $(X, u_X^{NR}) = \mathcal{H}^{NR}(X, A_X)$ e $(Y, u_Y^{NR}) = \mathcal{H}^{NR}(Y, A_Y)$. Denote por $C_X^*(x, x') = [x = x_0, x_1, \ldots, x_l = x']$ a cadeia que atinge o mínimo em $\widetilde{u}_X^*(x, x')$, com $x, x' \in X$ arbitrários e seja $C_Y(\phi(x), \phi(x')) = [\phi(x) = \phi(x_0), \phi(x_1), \ldots, \phi(x_l) = \phi(x')]$ uma cadeia de Y que liga $\phi(x)$ a $\phi(x')$.

Como a função ϕ é redução de dissimilaridade, temos que $A_Y(\phi(x_i), \phi(x_{i+1})) \leq A_X(x_i, x_{i+1})$. Consequentemente,

$$\max_{\phi(x_i) \in C_Y(\phi(x), \phi(x'))} A_Y(\phi(x_i), \phi(x_{i+1})) \leq \max_{i | x_i \in C^*(x, x')} A_X(x_i, x_{i+1}) = \widetilde{u}_X^*(x, x').$$

Como $\widetilde{u}_Y^*(\phi(x), \phi(x'))$ é o custo mínimo de cadeia, segue que

$$\begin{aligned} \widetilde{u}_Y^*(\phi(x), \phi(x')) &\leq \max_{\phi(x_i) \in C_Y(\phi(x), \phi(x'))} A_Y(\phi(x_i), \phi(x_{i+1})) \\ &\leq \max_{i | x_i \in C^*(x, x')} A_X(x_i, x_{i+1}) = \widetilde{u}_X^*(x, x'). \end{aligned} \tag{3.4}$$

Como x e x' são arbitrários, então (3.4) é válido para o par (x', x), ou seja,

$$\widetilde{u}_Y^*(\phi(x'), \phi(x)) \leq \widetilde{u}_X^*(x', x). \tag{3.5}$$

Por (3.2) e (3.4), segue que

$$\begin{aligned} u_Y^{NR}(\phi(x), \phi(x')) &= \max(\widetilde{u}_Y^*(\phi(x), \phi(x')), \widetilde{u}_Y^*(\phi(x'), \phi(x))) \\ &\leq \max(\widetilde{u}_X^*(x, x'), \widetilde{u}_X^*(x', x)) = u_X^{NR}(x, x'), \end{aligned}$$

o que conclui a demonstração. □

Observação 3.5. Podemos observar nas Figuras 3.1 e 3.2 os métodos de *Clustering* recíproco e não-recíproco, respectivamente. No método recíproco, $u_X^R(x, x') \leq \delta$ se existirem cadeias (recíprocas) cujas dissimilaridades máximas são menores ou iguais a δ em ambas as direções. No método não-recíproco, $u_X^{NR}(x, x') \leq \delta$ se x e x' se juntam em ambas as direções com cadeias (possivelmente diferentes) cujas dissimilaridades máximas são menores ou iguais a δ.

Portanto, *clustering* recíproco é um caso particular de *clustering* não-recíproco, e assim, para quaisquer $x, x' \in X$,

$$u_X^{NR}(x, x') \leq u_X^R(x, x'). \tag{3.6}$$

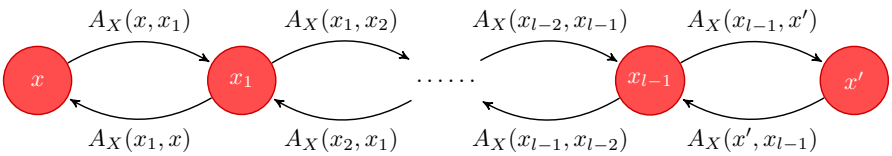


Figura 3.1: Representação de um *clustering* recíproco

O próximo teorema nos diz que, dado um método de *clustering* hierárquico admissível qualquer, tem-se que a ultramétrica proveniente desse método sempre permanece entre as ultramétricas u_X^{NR} e u_X^R.

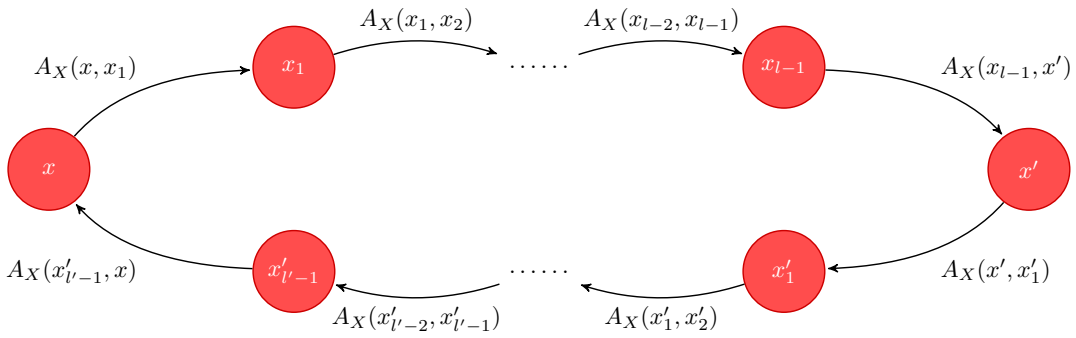


Figura 3.2: Representação de um *clustering* não-recíproco

Teorema 3.6. *Seja $\mathcal{H}$ um método de* clustering *hierárquico que satisfaz os axiomas* (A1) *e* (A2). *Dada uma* network *arbitrária $N = (X, A_X)$, se $(X, u_X) = \mathcal{H}(N)$ então*

$$u_X^{NR}(x, x') \leq u_X(x, x') \leq u_X^{R}(x, x'),$$

para quaisquer nós $x, x' \in X$.

Demonstração. Faremos a demonstração em duas partes, uma para cada desigualdade.

Parte 1. $u_X^{NR}(x, x') \leq u_X(x, x'),\ \forall x, x' \in X.$

Seja $\sim_{NR_X(\delta)}$ a relação de equivalência do *clustering* não-recíproco, dada por

$$x \sim_{NR_X(\delta)} x' \Leftrightarrow u_X^{NR}(x, x') \leq \delta. \tag{3.7}$$

Agora, considere o conjunto $Z = X \bmod \sim_{NR_X(\delta)}$ das classes de equivalência correspondentes à relação $\sim_{NR_X(\delta)}$ e a função $\phi_\delta : X \to Z$ dada por $\phi_\delta(x) = z$, em que z é a classe de equivalência correspondente a x. Logo, se $\phi_\delta(x) = \phi_\delta(x')$, então $x \sim_{NR_X(\delta)} x'$ e, por (3.7), $u_X^{NR}(x, x') \leq \delta$. Assim, podemos escrever

$$\phi_\delta(x) = \phi_\delta(x') \Leftrightarrow u_X^{NR}(x, x') \leq \delta. \tag{3.8}$$

Definimos a *network* $N_Z = (Z, A_Z)$, em que

$$A_Z(z, z') \coloneqq \min_{\substack{x \in \phi_\delta^{-1}(z) \\ x' \in \phi_\delta^{-1}(z')}} A_X(x, x'). \tag{3.9}$$

Observe que, por construção, ϕ_δ é redução de dissimilaridade, isto é,

$$A_X(x, x') \geq A_Z(\phi_\delta(x), \phi_\delta(x')). \tag{3.10}$$

De fato, se $\phi_\delta(x) = \phi_\delta(x')$, temos que $A_Z(\phi_\delta(x), \phi_\delta(x')) = 0$ e se $\phi_\delta(x) \neq \phi_\delta(x')$, temos que $\min_{x \in \phi_\delta^{-1}(z), x' \in \phi_\delta^{-1}(z')} A_X(x, x') \leq A_X(x, x')$. Logo, em ambos os casos, vale (3.10).

Seja $\mathcal{H}$ um método de *clustering* hierárquico que satisfaz os axiomas (A1) e (A2) e denote por $(Z, u_Z) = \mathcal{H}(N_Z)$.

Notemos que de acordo com a definição (3.9), se $z \neq z'$ então $A_Z(z, z') > \delta$ ou $A_Z(z', z) > \delta$. De fato, se ambos $A_Z(z, z') \leq \delta$ e $A_Z(z', z) \leq \delta$, poderemos construir cadeias $C(x, x')$ e $C(x', x)$ com custo máximo menor que δ para quaisquer $x \in \phi_\delta^{-1}(z)$ e $x' \in \phi_\delta^{-1}(z')$.

Para a cadeia $C(x, x')$, denote por $x_o \in \phi_\delta^{-1}(z)$ e $x_i' \in \phi_\delta^{-1}(z')$ os nós que atingem o mínimo em (3.9), ou seja, $A_Z(z, z') = A_X(x_o, x_i')$. Como x e x_o estão na mesma classe de equivalência, existe uma cadeia $C(x, x_o)$ de custo máximo menor ou igual a δ. Da mesma forma, como x' e x_i' estão na mesma classe de equivalência, existe uma cadeia $C(x_i', x')$ de custo máximo menor ou igual a δ. Consequentemente, a cadeia concatenada

$$C(x, x') = C(x, x_o) \uplus [x_o, x_i'] \uplus C(x_i', x') \tag{3.11}$$

tem custo máximo menor ou igual a δ.

Analogamente, podemos construir uma cadeia $C(x', x)$ cujo custo máximo é menor ou igual a δ.

Contudo, a existência dessas duas cadeias implicaria que x e x' estão no mesmo *cluster* no parâmetro δ, contrariando o fato de z ser diferente de z'.

Afirmação: $\text{mlc}(N_Z) > \delta$.

Suponhamos que $\text{mlc}(N_Z) \leq \delta$. Denote por $[z, z', \ldots, z^{(l)}, z]$ um ciclo cujo custo seja menor ou igual a δ. Para quaisquer $x \in \phi_\delta^{-1}(z)$ e $x' \in \phi_\delta^{-1}(z')$, podemos ligar x a x' usando a cadeia concatenada (3.11). Para ligar x' a x, denote por $x_o^{(k)}$ e $x_i^{(k+1)}$ os pontos para os quais $A_Z(z^{(k)}, z^{(k+1)}) = A_X(x_o^{(k)}, x_i^{(k+1)})$. Assim, podemos ligar x_o' e $x_o^{(l)}$ com a seguinte cadeia concatenada,

$$C(x_o', x_o^{(l)}) = \biguplus_{k=1}^{l-i} [x_o^{(k)}, x_i^{(k+1)}] \uplus C(x_i^{(k+1)}, x_o^{(k+1)}).$$

O custo máximo dessa cadeia é menor ou igual a δ, pois o custo máximo na cadeia $C(x_i^{(k+1)}, x_o^{(k+1)})$ é menor ou igual a δ, já que ambos os nós pertencem à mesma classe $z^{(k+1)}$, e $A_X(x_o^{(k)}, x_i^{(k+1)}) \leq \delta$, para todo k. Assim, podemos ligar x' a x com a cadeia concatenada

$$C(x', x) = C(x', x_o') \uplus C(x_o', x_o^{(l)}) \uplus [x_o^{(l)}, x_i] \uplus C(x_i, x),$$

cujo custo máximo é menor ou igual a δ.

Usando as cadeias $C(x, x')$ e $C(x', x)$ construídas acima, podemos concluir que $u_X^{NR}(x, x') \leq \delta$, o que contradiz o fato de que x e x' pertencem a diferentes classes de equivalência. Consequentemente, $\text{mlc}(N_Z) > \delta$, como afirmado.

Segue da Propriedade da Influência (P1) (e da afirmação acima) que

$$u_Z(z, z') > \delta, \ \forall z, z' \in Z, \ z \neq z'.$$

Além disso, por (3.8), (3.10) e (A2), tem-se $u_X(x, x') \geq u_Z(z, z') > \delta$, ou seja, quando x e x' são aplicadas em classes de equivalência distintas, temos que $u_X(x, x') > \delta$, ou ainda, $u_X^{NR}(x, x') > \delta$.

Consequentemente, podemos afirmar que $u_X^{NR}(x, x') > \delta$ implica $u_X(x, x') > \delta$, ou seja,

$$\{(x, x') \mid u_X^{NR}(x, x') > \delta\} \subseteq \{(x, x') \mid u_X(x, x') > \delta\}. \tag{3.12}$$

Como (3.12) vale para $\delta > 0$ arbitrário, segue que

$$u_X^{NR}(x, x') \leq u_X(x, x'), \ \forall x, x' \in X.$$

Parte 2. $u_X(x, x') \leq u_X^R(x, x'), \ \forall x, x' \in X.$

Sejam x e x' arbitrários com $u_X^R(x,x') = \delta$. Seja $C(x,x') = [x = x_0, \ldots, x_l = x']$ uma cadeia que atinge o mínimo em (3.2). Assim, podemos escrever

$$\delta = u_X^R(x,x') = \max_i\{\max(A_X(x_i,x_{i+1}), A_X(x_{i+1},x_i))\}. \tag{3.13}$$

Considere a *network* simétrica $\overrightarrow{\Delta}_2(\delta,\delta) = (\{p,q\}, A_{p,q})$ e sua imagem $\mathcal{H}(\overrightarrow{\Delta}_2(\delta,\delta)) = (\{p,q\}, u_{p,q})$. Por (A1), temos que $u_{p,q}(p,q) = \max(\delta,\delta) = \delta$. Sejam $\phi_i : \{p,q\} \to X$ as transformações dadas por $\phi_i(p) = x_i$, $\phi_i(q) = x_{i+1}$, para x_i e x_{i+1} nós consecutivos da cadeia $C(x,x')$. Como $A_X(x_i,x_{i+1}) \leq \delta$ e $A_X(x_{i+1},x_i) \leq \delta$ por (3.13), obtemos

$$A_X(\phi_i(p),\phi_i(q)) \leq A_{p,q}(p,q) = \delta,$$
$$A_X(\phi_i(q),\phi_i(p)) \leq A_{p,q}(q,p) = \delta,$$

o que implica que ϕ_i é redução de dissimilaridade e daí segue de (A2) que

$$u_X(\phi_i(p),\phi_i(q)) \leq u_{p,q}(p,q) = \delta. \tag{3.14}$$

Como (3.14) é válida para todo i e $\phi_i(p) = x_i$, $\phi_i(q) = x_{i+1}$, concluímos que $u_X(x_i,x_{i+1}) \leq \delta$, $\forall i$. Pela desigualdade triangular forte, temos que

$$u_X(x,x') \leq \max_i\{u_X(x_i,x_{i+1})\} \leq \delta = u_X^R(x,x'),\ \forall x,x' \in X. \qquad \square$$

Networks simétricas

Finalizamos este capítulo trabalhando apenas com *networks* simétricas, ou seja, vamos considerar $N = (X, A_X)$ tal que $A_X(x,x') = A_X(x',x)$, para todo $x,x' \in X$.

Notemos que nesse caso os *clusterings* recíproco e não-recíproco são equivalentes. De fato, dados $x,x' \in X$ arbitrários, $\bar{A}_X(x_i,x_{i+1}) = A_X(x_i,x_{i+1}) = A_X(x_{i+1},x_i)$, assim temos que a definição da ultramétrica recíproca se reduz a

$$u_X^R(x,x') = \min_{C(x,x')} \max_{i|x_i \in C(x,x')} \bar{A}_X(x_i,x_{i+1}) = \min_{C(x,x')} \max_{i|x_i \in C(x,x')} A_X(x_i,x_{i+1}) = \widetilde{u}_X^*(x,x').$$

Além disso, note que os custos de uma cadeia $C(x,x') = [x = x_0, \ldots, x_l = x']$ e de sua recíproca $C(x',x) = [x' = x_l, \ldots, x_0 = x]$ são os mesmos, o que implica que $\widetilde{u}_X^*(x,x') = \widetilde{u}_X^*(x',x)$ e, portanto, pela definição de ultramétrica não-recíproca, temos

$$u_X^{NR}(x,x') = \widetilde{u}_X^*(x,x') = u_X^R(x,x'). \tag{3.15}$$

Lembremos de (2.5) que a ultramétrica *single-linkage* para *networks* simétricas satisfaz $u_X^{SL}(x,x') = \widetilde{u}_X^*(x,x')$, e portanto, por (3.15), temos que

$$u_X^{SL}(x,x') = u_X^R(x,x') = u_X^{NR}(x,x'). \tag{3.16}$$

Assim, pelo Teorema 3.6, concluímos que u_X^{SL} é a *única* ultramétrica possível para *networks* simétricas. Mais precisamente:

Corolário 3.7. *Sejam $\mathcal{H} : \mathcal{N} \to \mathcal{U}$ um método de* clustering *hierárquico qualquer, $N = (X, A_X)$ uma* network *simétrica e $\mathcal{H}^{SL}$ o método* single-linkage. *Se $\mathcal{H}$ satisfaz os axiomas* (A1) *e* (A2)*, então $\mathcal{H} = \mathcal{H}^{SL}$.*

Demonstração. Como $\mathcal{H}$ satisfaz as hipóteses do Teorema 3.6, temos que

$$u_X^{NR}(x,x') \leq u_X(x,x') \leq u_X^R(x,x'),$$

para todo $x,x' \in X$. Mas, por (3.16), $u_X^{SL}(x,x') = u_X^R(x,x') = u_X^{NR}(x,x')$, o que implica que $u_X^{SL}(x,x') = u_X(x,x')$, para todo $x,x' \in X$, ou seja, $\mathcal{H} \equiv \mathcal{H}^{SL}$. $\square$

4 Algoritmos

Neste capítulo, interpretaremos as definições de ultramétrica recíproca e não-recíproca por meio de matrizes de modo a resolvermos problemas computacionalmente. Para isso, introduziremos primeiramente o conceito de *dioide*, necessário para tal interpretação. Mais detalhes sobre Álgebra Dioide podem ser encontrados em [5].

4.1 Álgebra Dioide

Seja E um conjunto não-vazio qualquer e $\oplus$ uma operação binária em E. Dizemos que $(E, \oplus)$ é um monoide se $\oplus$ for associativa e tiver um elemento neutro. Se $\oplus$ for comutativa, dizemos que o monoide $(E, \oplus)$ é comutativo.

Um elemento $a \in E$ é idempotente se $a \oplus a = a$. Além disso, $\oplus$ é idempotente se $a \oplus a = a, \forall a \in E$.

Definição 4.1. *Dizemos que $(E, \oplus, \otimes)$ é um dioide se:*

(i) *$(E, \oplus)$ é um monoide comutativo com elemento neutro ϵ;*

(ii) *$(E, \otimes)$ é um monoide com elemento neutro e;*

(iii) *A relação $a \leq b \Leftrightarrow \exists c : b = a \oplus c$ é uma relação de ordem, isto é, se $a \leq b$ e $b \leq a$ então $a = b$;*

(iv) *ϵ é absorvente para $\otimes$, ou seja, para qualquer $a \in E$, $a \otimes \epsilon = \epsilon \otimes a = \epsilon$;*

(v) *$\otimes$ é distributiva com respeito a $\oplus$.*

Considere $(E, \oplus, \otimes)$ um dioide qualquer. Como E é um conjunto ordenado dado na Definição 4.1, item (iii), podemos estabelecer uma topologia para E induzida por essa relação de ordem. Para mais detalhes, veja em [5, Capítulo 3, Seção 2]. Isto posto, podemos falar sobre limite e continuidade.

Definição 4.2. *Sejam $(E, \oplus, \otimes)$ um dioide e $a \in E$ qualquer. O* quasi-inverso *de a, denotado por a^*, é o limite, quando existir, da sequência*

$$a^* := \lim_{k \to \infty} e \oplus a \oplus a^{(2)} \oplus \cdots \oplus a^{(k)},$$

onde e é o elemento neutro de $\otimes$ e $a^{(k)} = \underbrace{a \otimes \cdots \otimes a}_{k \text{ vezes}}$.

Um elemento $a \in E$ é p-estável se

$$e \oplus a \oplus a^{(2)} \oplus \cdots \oplus a^{(p)} = e \oplus a \oplus a^{(2)} \oplus \cdots \oplus a^{(p)} \oplus a^{(p+1)} = \cdots$$

onde e é o elemento neutro de $\otimes$.

A Definição 4.2 pode ser interpretada utilizando-se matrizes, como a seguir.

A *quasi-inversa* de uma matriz A, denotada por A^*, é o limite, quando existir, da sequência de matrizes

$$A^* \coloneqq \lim_{k\to\infty} I \oplus A \oplus A^{(2)} \oplus \cdots \oplus A^{(k)}, \tag{4.1}$$

onde I é a matriz identidade na álgebra dioide, ou seja, I tem o elemento neutro e na diagonal e o elemento neutro ϵ fora da diagonal.

Antes de prosseguirmos, faremos brevemente menção a alguns conceitos e resultados, obtidos com mais detalhes em [5, Capítulo 4, Seção 3.2].

Proposição 4.3. *Se $\oplus$ é idempotente, então $(I \oplus A)^{(k)} = I \oplus A \oplus A^{(2)} \oplus \cdots \oplus A^{(k)}$.*

Demonstração. Temos que

$$(I \oplus A)^{(k)} = \underbrace{(I \oplus A) \otimes \cdots \otimes (I \oplus A)}_{k \text{ vezes}} = I \oplus \bigoplus_{r=1}^{k} C_k^r A^{(r)}, \tag{4.2}$$

onde $C_k^r = \dfrac{k!}{r!(k-r)!}$. Como $\oplus$ é idempotente e $C_k^r \in \mathbb{Z}$, temos que $C_k^r A^{(r)} = A^{(r)}$. Substituindo em (4.2), concluímos a prova. □

Sejam $(E, \oplus, \otimes)$ um dioide, A uma matriz $n \times n$ com entradas em E e $G(A)$ o seu grafo correspondente, ou seja, tendo A como matriz de adjacência.

Um circuito de $G(A)$ é um *circuito baseado* quando um de seus vértices é tratado como origem. Neste caso, o *peso* do circuito baseado $\gamma = \{i_1i_2, i_2i_3, \ldots, i_ki_1\}$ de origem i_1 é $w(\gamma) = a_{i_1i_2} \otimes a_{i_2i_3} \otimes \cdots \otimes a_{i_ki_1}$. Dizemos que um grafo $G(A)$ não tem um *circuito p-absorvente* se o peso de cada circuito baseado é um elemento p-estável de E.

Com isso, podemos apresentar (ver [5, Capítulo 3, Seção 3.3, Teorema 1]):

Teorema 4.4. *Se $G(A)$ não tem um circuito 0-absorvente, então*

$$\begin{aligned} A^* &= \lim_{k\to+\infty} I \oplus A \oplus A^{(2)} \oplus \cdots \oplus A^{(k)} \\ &= I \oplus A \oplus A^{(2)} \oplus \cdots \oplus A^{(n-1)} = I \oplus A \oplus A^{(2)} \oplus \cdots \oplus A^{(n)} = \cdots \end{aligned}$$

Além disso, A^* satisfaz a equação

$$A^* = I \oplus A \otimes A^* = I \oplus A^* \otimes A.$$

4.2 Algoritmo para ultramétricas

Dada uma *network* $N = (X, A_X)$ tal que $|X| = n$, interpretaremos A_X como uma matriz de dissimilaridades $n \times n$. Também, podemos denotar a ultramétrica u_X como uma matriz simétrica $n \times n$ e reinterpretar (3.1) como

$$\bar{A}_X \coloneqq \max(A_X, A_X^T),$$

em que A_X^T é a matriz transposta de A_X e $\max(-,-)$ é tomado ponto-a-ponto, ou seja, resulta em uma matriz de máximos.

Assim, podemos interpretar as definições de ultramétricas recíproca e não-recíproca como matrizes e então resolver problemas computacionalmente. Para isso, usaremos os conceitos vistos na Seção 4.1 para a álgebra dioide $\mathfrak{U} \coloneqq (\mathbb{R}^+ \cup \{+\infty\}, \min, \max)$, conforme em [1].

Nesse caso, a soma $\oplus$ é a operação *mínimo* e o produto $\otimes$ é a operação *máximo*. Mais precisamente, dados $a, b \in \mathbb{R}^+ \cup \{+\infty\}$, então

$$a \oplus b = \min(a, b) \quad \text{e} \quad a \otimes b = \max(a, b).$$

Denotaremos o conjunto $\{1, 2, \ldots, n\}$ por $[1, n]$.

Na álgebra dioide, o produto $A \otimes B$ de duas matrizes A e B é dado pela matriz com entradas

$$[A \otimes B]_{ij} \coloneqq \bigoplus_{k=1}^{n} (A_{ik} \otimes B_{kj}) = \min_{k \in [1,n]} \max(A_{ik}, B_{kj}). \tag{4.3}$$

Para inteiros $k \geq 2$, a k-ésima potência de uma matriz de dissimilaridades A_X relacionada à matriz ultramétrica u_X é dada por

$$A_X^{(k)} \coloneqq A_X \otimes A_X^{(k-1)},$$

com $A_X^{(1)} \coloneqq A_X$. Note que os elementos do produto dioide $u_X^{(2)}$são dados por

$$[u_X^{(2)}]_{ij} = \min_{k \in [1,n]} \max([u_X]_{ik}, [u_X]_{kj}).$$

Como u_X satisfaz a desigualdade triangular forte, então $[u_X]_{ij} \leq \max([u_X]_{ik}, [u_X]_{kj})$, para todo $k \in [1, n]$. Em particular, para $k = j$, temos

$$\max([u_X]_{ij}, [u_X]_{jj}) = \max([u_X]_{ij}, 0) = [u_X]_{ij}.$$

Consequentemente, $[u_X^{(2)}]_{ij} = [u_X]_{ij}$, para todo $i, j \in [1, n]$, e portanto $u_X^{(2)} = u_X$, ou seja, u_X é idempotente.

Na definição de quasi-inversa de uma matriz A, dada em (4.1),

$$A^* \coloneqq \lim_{k \to \infty} I \oplus A \oplus A^{(2)} \oplus \cdots \oplus A^{(k)}.$$

Porém, para o dioide $\mathfrak{U} \coloneqq (\mathbb{R}^+ \cup \{+\infty\}, \min, \max)$, a matriz identidade I tem 0 na diagonal e $+\infty$ fora da diagonal.

A utilidade de uma quasi-inversa para o nosso estudo reside no fato de que, dada uma matriz dissimilaridade A_X, tem-se

$$[A_X^*]_{ij} = \min_{C(x_i, x_j)} \max_{k | x_k \in C(x_i, x_j)} A_X(x_k, x_{k+1}), \tag{4.4}$$

onde A_X^* é a quasi-inversa de A_X. Em outras palavras, os elementos da quasi-inversa A_X^* correspondem aos custos mínimos de cadeia da *network* associada (X, A_X). Para mais detalhes, veja [5, Capítulo 6, Seção 6.1].

Assim, para calcular computacionalmente as ultramétricas recíproca e não-recíproca, temos o seguinte resultado.

Teorema 4.5. *Seja $N = (X, A_X)$ uma* network *com n nós. A ultramétrica recíproca u_X^R pode ser calculada por*

$$u_X^R = \big(\max(A_X, A_X^T)\big)^{(n-1)}$$

e a ultramétrica não-recíproca u_X^{NR} pode ser calculada por

$$u_X^{NR} = \max\big((A_X)^{(n-1)}, (A_X^T)^{(n-1)}\big),$$

onde a operação $(-)^{(n-1)}$ é a $(n-1)$-ésima potência da matriz na álgebra dioide.

Demonstração. Por (4.4), temos $A_X^* = \widetilde{u}_X^*$. Apenas trocando a notação, pela definição da ultramétrica não-recíproca temos que

$$u_X^{NR} = \max(A_X^*, (A_X^*)^T).$$

Analogamente, a quasi-inversa da matriz simetrizada $\bar{A}_X := \max(A_X, A_X^T)$ é

$$[\bar{A}_X^*]_{ij} = \min_{C(x_i,x_j)} \max_{k|x_k \in C(x_i,x_j)} \bar{A}_X(x_k, x_{k+1}).$$

Pela definição de ultramétrica recíproca, segue que

$$u_X^R = \bar{A}_X^* = \max(A_X, A_X^T)^*$$

e portanto, para concluir a demonstração, basta mostrar que $A_X^* = A_X^{(n-1)}$.

Note que na álgebra dioide $\mathfrak{U}$ a operação $\oplus$ é idempotente, ou seja, $a \oplus a = \min(a, a) = a$, para todo a. Nesse caso, pela Proposição 4.3, temos que

$$I \oplus A_X \oplus A_X^{(2)} \oplus \cdots \oplus A_X^{(k)} = (I \oplus A_X)^{(k)}, \tag{4.5}$$

para todo $k \geq 1$. Além disso, como os elementos da diagonal são nulos nas matrizes do lado direito de (4.5) e os elementos fora da diagonal de I são $+\infty$, segue que $I \oplus A_X = A_X$. Consequentemente,

$$I \oplus A_X \oplus A_X^{(2)} \oplus \cdots \oplus A_X^{(k)} = A_X^{(k)}. \tag{4.6}$$

Fazendo $k \to \infty$ em ambos os lados da igualdade (4.6), e pela definição de quasi-inversa, obtemos

$$A_X^* = \lim_{k\to\infty} A_X^{(k)}.$$

Finalmente, pelo Teorema 4.4, segue que $A_X^{(n-1)} = A_X^{(n)}$, provando que o limite em (4.1) existe e, mais importante, que $A_X^* = A_X^{(n-1)}$, como desejado. □

5 Aplicações

Neste capítulo, aplicaremos os métodos de *clustering* hierárquicos desenvolvidos anteriormente, ou seja, os métodos recíproco e não-recíproco, para obter dendrogramas para dois conjuntos de dados de *networks* assimétricas.

Analisaremos a *network* de migração interna entre condados dos Estados Unidos (EUA) para o ano de 2011 e também analisaremos a *network* de migração entre países do mundo todo, no período 2000–2011.

Para calcularmos computacionalmente as ultramétricas para esses dados, apresentamos inicialmente algumas definições.

Denotamos por X o conjunto de condados dos Estados Unidos ou de países do mundo, e definimos a função $\mu\colon X \times X \to \mathbb{R}_+ \cup \{+\infty\}$, em que $\mu(x, x')$ é o número de indivíduos que migraram do condado (ou país) x para o condado (ou país) x', e $\mu(x, x) = +\infty$, para todo $x, x' \in X$.

Assim, construímos uma *network* assimétrica $N_X = (X, A_X)$, onde

$$A_X(x, x') = \begin{cases} 0, & \text{se } x = x', \\ f\Big(\mu(x, x')/\sum_t \mu(t, x')\Big), & \text{se } x \neq x', \end{cases} \tag{5.1}$$

para todo $x, x' \in X$, onde $f\colon [0, 1) \to \mathbb{R}_+$ é qualquer função decrescente. No nosso caso, optamos por considerar $f(x) = 1 - x$.

Analogamente, a função A_X também pode ser definida tomando o somatório sobre a segunda variável, ou seja,

$$A_X(x, x') = \begin{cases} 0, & \text{se } x = x', \\ f\Big(\mu(x, x')/\sum_t \mu(x, t)\Big), & \text{se } x \neq x'. \end{cases} \tag{5.2}$$

A normalização $\mu(x, x')/\sum_t \mu(t, x')$ em (5.1) pode ser interpretada como sendo a probabilidade de que um imigrante do local x' tenha vindo do local x, assim como $\mu(x, x')/\sum_t \mu(x, t)$ em (5.2) pode ser interpretada como sendo a probabilidade de que um imigrante do local x tenha vindo do local x'.

O papel da função f é transformar $\mu(x, x')/\sum_t \mu(t, x')$ e $\mu(x, x')/\sum_t \mu(x, t)$ em dissimilaridades.

5.1 Migração da população dos EUA

Iniciamos essa seção apresentando um resumo dos dados iniciais, a forma como estão e foram organizados. Tais dados estão no arquivo `Net_Gross_US.txt` obtido

em [2]. A partir desse arquivo, filtramos apenas as colunas $2, 3, 4, 5, 6, 8$, produzindo um novo arquivo, `raw_data_usa.csv`, com 6 colunas, a saber:

$$e_A \qquad c_A \qquad e_B \qquad c_B \qquad \mu(c_A, c_B) \qquad \mu(c_B, c_A) \tag{5.3}$$

onde e denota o nome de um dos estados[1], c denota o nome de um condado do estado e, e μ é a migração (orientada) entre dois condados.

O arquivo `prepare_data_states_num_counties.csv` contém informações sobre cada um dos 52 estados, mais precisamente, o nome do estado, sua quantia de condados, e seu código ISO, conforme a Tabela 5.1. O número total de condados é 3220, contando *District of Columbia* e *Puerto Rico*, e foram enumerados por ordem alfabética, primeiramente para os estados e então para os condados deste estado. Por exemplo, o primeiro condado do Alaska (segundo estado) recebe o número 68, pois existem 67 condados no único estado anterior, Alabama.

Estado (em inglês)	**Cond.**	**ISO**	**Estado** (em inglês)	**Cond.**	**ISO**
Alabama	67	AL	Montana	56	MT
Alaska	29	AK	Nebraska	93	NE
Arizona	15	AZ	Nevada	17	NV
Arkansas	75	AR	New Hampshire	10	NH
California	58	CA	New Jersey	21	NJ
Colorado	64	CO	New Mexico	33	NM
Connecticut	8	CT	New York	62	NY
Delaware	3	DE	North Carolina	100	NC
District of Columbia	1	DC	North Dakota	53	ND
Florida	67	FL	Ohio	88	OH
Georgia	159	GA	Oklahoma	77	OK
Hawaii	5	HI	Oregon	36	OR
Idaho	44	ID	Pennsylvania	67	PA
Illinois	102	IL	Puerto Rico	78	PR
Indiana	92	IN	Rhode Island	5	RI
Iowa	99	IA	South Carolina	46	SC
Kansas	105	KS	South Dakota	66	SD
Kentucky	120	KY	Tennessee	95	TN
Louisiana	64	LA	Texas	254	TX
Maine	16	ME	Utah	29	UT
Maryland	24	MD	Vermont	14	VT
Massachusetts	14	MA	Virginia	133	VA
Michigan	83	MI	Washington	39	WA
Minnesota	87	MN	West Virginia	55	WV
Mississippi	82	MS	Wisconsin	72	WI
Missouri	115	MO	Wyoming	23	WY

Tabela 5.1: Conteúdo de `prepare_data_states_num_counties.csv`

A seguir, explicaremos o conteúdo dos programas em Python criados para filtrar os dados, extrair uma matriz de migração em condados, calcular as ultramétricas pelos

[1]Em todo o texto, também chamaremos de estado *District of Columbia* e *Puerto Rico*.

métodos recíproco e não-recíproco e, por fim, criar dendrogramas para cada *network* assim construída.

Os programas em Python e os arquivos de dados estão disponíveis em [3].

5.1.1 Preparando os dados: `prepare_data.py`

Descreveremos abaixo os blocos mais relevantes do programa `prepare_data.py`. Vale ressaltar que as funções abaixo descritas estão em ordem alfabética, não na ordem em que serão executadas no programa.

O programa inicia carregando os módulos necessários: NumPy e *pandas*, versões 1.13.3 e 0.19.2, respectivamente ([11, 9]).

```
1  # coding:utf-8
2  import numpy as np
3  import pandas as pd
```

A função `filter_line()` recebe uma 6-upla como em (5.3), determina as posições i, j de c_A, c_B e define o valor das entradas correspondentes na matriz de migração como sendo $\mu(c_A, c_B)$ e $\mu(c_B, c_A)$, ou seja,

$$M_{i,j} = \mu(c_A, c_B), \quad M_{j,i} = \mu(c_B, c_A).$$

```
4  def filter_line(s1, c1, s2, c2, x, y):
5      print 'filtering data to %s, %s, %s, %s' % (s1, c1, s2, c2)
6      from_ = (s1.decode('latin1'), c1.decode('latin1'))
7      to_ = (s2.decode('latin1'), c2.decode('latin1'))
8      from_idx = list_all_states_counties.index(from_) # defined in line 53
9      to_idx = list_all_states_counties.index(to_)     # defined in line 53
10     migration_matrix[from_idx, to_idx] = y
11     migration_matrix[to_idx, from_idx] = x
```

A função `get_len()` retorna o número de condados de um dado estado.

```
12 def get_len(state):
13     return int(len_states[np.where(len_states == state)[0][0], 1])
```

A função `migration_from_states()` é utilizada para obter os dados de migração a partir de um determinado estado, ou seja, fixado $e \in$ `states`, a função cria dois arquivos:

- `mat_`$\langle e \rangle$`_to_USA.csv`: migração a partir de e para qualquer outro estado, não necessariamente diferente de e. Assim, representa uma matriz de ordem $m \times 3220$, onde m é o número de condados de e (obtido por `get_len(`$\langle e \rangle$`)`)
- `mat_`$\langle e \rangle$`_to_`$\langle e \rangle$`.csv`: migração interna apenas entre os condados de e, representando uma matriz quadrada de ordem m, onde m é como acima.

```
14 def migration_from_states():
15     migration_matrix = np.loadtxt(migration_matrix_file, dtype=int,
       ↪  delimiter=',')
16     for i, state in enumerate(states):
```

```
17          print 'filtering migration data from %s to USA' % state
18          len_state = get_len(state)
19          len_prev = sum([get_len(s) for s in states[:i]])
20          tmp = migration_matrix[len_prev:len_prev+len_state]
21          tmp_single = migration_matrix[len_prev:len_prev+len_state,
            ↪  len_prev:len_prev+len_state]
22          np.savetxt('./migration_matrices/mat_%s_to_USA.csv' % state, tmp,
            ↪  fmt='%d', delimiter=',')
23          np.savetxt('./migration_matrices/mat_%s_to_%s.csv' % (state,
            ↪  state), tmp_single, fmt='%d', delimiter=',')
```

A função `migration_usa()` recursivamente aplica a função `filter_line()` em cada linha do arquivo `raw_data_usa.csv`, salvando o resultado em `migration_matrix.csv`. Uma vez que os valores contidos nesse arquivo são os dados de migração entre dois condados, ele pode ser interpretado como uma matriz quadrada de ordem 3220.

```
24 def migration_usa():
25     global migration_matrix
26     migration_matrix = np.zeros((num_counties, num_counties))
27     with open(raw_data_usa_file, 'r') as f:
28         for line in f:
29             tmp = line.split(',')
30             if tmp[1] != '-' and tmp[3] != '-':
31                 filter_line(tmp[0], tmp[1], tmp[2], tmp[3], tmp[4],
                    ↪  tmp[5])
32         f.close()
33     np.savetxt(migration_matrix_file, migration_matrix, fmt='%d',
       ↪  delimiter=',')
```

A função `print_fields()` é utilizada para filtrar cada linha do `Net_Gross_US.txt` e obter os valores desejados, como em (5.3).

```
34 def print_fields(line):
35     data = ''
36     for cols in fields[used_fields]:
37         val = line[cols[0]-1:cols[1]].strip()
38         if val != '.':
39             data += val+','
40         else:
41             data += val.replace('.', '')+','
42     return data[:-1]
```

A função `use_pandas()` utiliza o módulo Python *pandas* para ler o arquivo de dados `raw_data_usa_file.csv`, eliminar entradas não desejadas, como por exemplo, dados de migração de algum estado para outro país, ou de algum outro continente para algum estado, entre outros.

Como resultado, temos a variável `states` que contém os 52 estados de interesse, bem como `list_all_states_counties`, que contém pares de estados e seus respectivos condados, ou seja,

$$\texttt{list\_all\_states\_counties} = \{(e, c),\ c \in e, e \in \texttt{states}\}.$$

Por fim, o número total de condados é calculado, a saber, 3220 e salvo na variável `num_counties`.

```
43 def use_pandas():
44     global states, list_all_states_counties, num_counties
45     print 'Defining variables states, list_all_states_counties,
       ↪  num_counties'
46     df = pd.read_csv(raw_data_usa_file, encoding='latin1', header=None,
       ↪  delim_whitespace=False, index_col=None, sep=',',
       ↪  names=used_fields)
47     data = df[df[7] != '-'][used_fields]
48     states = data[4].unique().tolist()
49     states = states[:-9]
50     list_all_states_counties = []
51     for state in states:
52         tmp = data[data[4] == state][5].unique().tolist()
53         list_all_states_counties += [(state, cty) for i, cty in
           ↪  enumerate(tmp)] #
54     num_counties = len(list_all_states_counties)
```

A função `write_raw_data()` lê cada linha `l` do arquivo `Net_Gross_US.txt` e executa a função `print_fields(l)`, escrevendo o resultado em uma linha do arquivo `raw_data_usa.csv` (que é utilizado pela função `use_pandas()`).

```
55 def write_raw_data():
56     print 'Generating the file %s.\nWait...' % raw_data_usa_file
57     with open(raw_data_usa_file, 'w') as f:
58         with open(net_gross_file, 'rb') as net:
59             for l in net:
60                 f.write('%s\n' % print_fields(l))
61             net.close()
62         f.close()
```

A seguir, apresentamos a parte principal do programa, na qual algumas variáveis são definidas e as funções acima são executadas.

Primeiro, definimos as variáveis com os nomes dos arquivos que serão lidos (*required files*) e que serão criados (*output files*) por algumas funções. A variável `used_fields` informa quais colunas[2] do arquivo `Net_Gross_US.txt` serão utilizadas.

```
63 """ MAIN CODE """
64
65 """ Required files """
66 net_gross_file = 'Net_Gross_US.txt'
67 fields = np.loadtxt('prepare_data_fields.csv', dtype=int, delimiter=',')
68 len_states = np.loadtxt('prepare_data_states_num_counties.csv',
   ↪  dtype=str, delimiter=',')
69
70 """ Output files """
71 raw_data_usa_file = 'raw_data_usa.csv'
72 migration_matrix_file = 'migration_matrix.csv'
73 used_fields = [4, 5, 6, 7, 8, 10]
```

[2] A numeração das colunas inicia-se em 0.

Por fim, executamos as funções abaixo para obtermos as matrizes de migração, que são salvas nos arquivos `./migration_matrices/*.csv`.

```
74 write_raw_data()
75 use_pandas()
76 migration_usa()
77 migration_from_states()
78
79 """ End of file prepare_data.py """
```

5.1.2 Criando as ultramétricas: `compute_ultrametric.py`

O programa `compute_ultrametric.py`, como o nome diz, calcula as ultramétricas pelos métodos recíproco e não-recíproco, a partir das matrizes de migração, criadas no programa `prepare_data.py` da Seção 5.1.1.

Lembramos que o algoritmo para calcular u^R e u^{NR} está descrito no Capítulo 3.

Iniciamos o programa carregando o módulo NumPy.

```
1 # coding:utf-8
2 import numpy as np
```

A função `loop_ultrametric_inner_state()` itera sobre todos os estados, exceto 'District of Columbia', aplicando a função `ultrametric_inner_state()` em cada um deles que, de fato, calcula a ultramétrica para um estado dado.

```
3 def loop_ultrametric_inner_state():
4     for state in states:
5         if state != 'District of Columbia':
6             ultrametric_inner_state(state) # defined in line 40
```

A função `max_prod()` implementa a operação definida em (4.3), que recebe duas matrizes quadradas de ordem $n \times n$ e calcula as entradas (i, j) e (j, i) da matriz resultante, por meio da função `min_max()`. Vale observar que, para o cálculo da ultramétrica, as duas matrizes de entrada `mat1`, `mat2` são iguais.

```
7 def max_prod(mat1, mat2):
8     n = len(mat1)
9     tmp = np.zeros((n, n), dtype=float)
10    for i in range(n):
11        for j in range(i, n):
12            tmp[i, j] = min_max(mat1, mat2, i, j)
13            tmp[j, i] = min_max(mat2, mat1, j, i)
14    return tmp
```

A função `min_max()` implementa a operação definida em (4.3). Graças ao módulo Numpy, essa operação pode ser feita simplesmente com uma linha de comando, usando `min()` e `maximum()`.

```
15 def min_max(mat1, mat2, i, j):
16     return min(np.maximum(mat1[i, :], mat2[:, j]))
```

A função `power()` recebe uma matriz A quadrada de ordem n e retorna a potência A^{n-1}, onde o produto é definido em (4.3). Devido à estabilidade do produto, como visto no Teorema 4.4, basta calcularmos as potências A^{2k} até $2^k \leq n-1$.

```
17 def power(mat):
18     n = len(mat)
19     k = 1
20     while 2 ** (k - 1) < n:
21         mat = max_prod(mat, mat)
22         k += 1
23     return mat
```

As duas seguintes funções, `u_nr()` e `u_r()` calculam as ultramétricas, pelos métodos não-recíproco e recíproco, respectivamente, a partir de uma matriz quadrada. Ambas funções possuem o argumento opcional `output_file`, cujos valores padrão são as *strings* `'uNR.csv'` e `'uR.csv'`, que são os nomes dos arquivos `.csv` que conterão a matriz da ultramétrica e serão criados na pasta `./ultra_metrics`.

As funções implementam os cálculos descritos no Teorema 4.5.

```
24 def u_nr(mat, output_file='uNR.csv'):
25     print 'Computing non-reciprocal ultra metric. Wait... ',
26     tmp1 = power(mat)
27     tmp2 = tmp1.T
28     tmp = np.maximum(tmp1, tmp2)
29     np.savetxt('ultra_metric/'+output_file.replace(' ', '_'), tmp,
       ↪  fmt='%f', delimiter=',')
30     print 'saved in %s' % output_file.replace(' ', '_')
31     return tmp
32
33 def u_r(mat, output_file='uR.csv'):
34     print 'Computing     reciprocal ultra metric. Wait... ',
35     a_bar = np.maximum(mat, mat.T)
36     tmp = power(a_bar)
37     np.savetxt('ultra_metric/'+output_file.replace(' ', '_'), tmp,
       ↪  fmt='%f', delimiter=',')
38     print 'saved in %s' % output_file.replace(' ', '_')
39     return tmp
```

A função `ultrametric_inner_state()`, como o nome diz, trata da migração interna de um estado. Mais precisamente, dado um estado, carrega seu arquivo de migração interna e executa as funções `u_r()` e `u_nr()` nas duas matrizes retornadas pela função `weight_fc()`. Como consequência, as matrizes da ultramétrica são salvas nos arquivos `.csv` correspondentes.

```
40 def ultrametric_inner_state(state): #
41     print '\n** %s' % state
42     mat = np.loadtxt('migration_matrices/mat_%s_to_%s.csv' % (state,
       ↪  state), dtype=float, delimiter=',')
43     w_mat1, w_mat2 = weight_fc(mat) # defined in line 56
44     u_r(w_mat1, output_file='col_uR_%s.csv' % state)
45     u_nr(w_mat1, output_file='col_uNR_%s.csv' % state)
46     u_r(w_mat2, output_file='row_uR_%s.csv' % state)
```

```
47      u_nr(w_mat2, output_file='row_uNR_%s.csv' % state)
```

A função `ultrametric_national()`, assim como a anterior, executa quatro vezes as funções que calculam as ultramétricas, porém, neste caso, a matriz é para o fluxo migratório nacional, ou seja, engloba também os dados entre diferentes estados. Vale observar que, devido à grande dimensão (3220^2) das matrizes, esta função **pode levar horas para ser concluída**.

```
48 def ultrametric_national():
49     print 'This function needs time to finish.'
50     mat = np.loadtxt(migration_matrix_file, dtype=float, delimiter=',')
51     w_mat1, w_mat2 = weight_fc(mat)
52     u_r(w_mat1, output_file='col_uR.csv')
53     u_nr(w_mat1, output_file='col_uNR.csv')
54     u_r(w_mat2, output_file='row_uR.csv')
55     u_nr(w_mat2, output_file='row_uNR.csv')
```

A função `weight_fc()` implementa os cálculos descritos em (5.1) e (5.2), para aplicar uma *função peso* à matriz de entrada. No código abaixo, duas funções são utilizadas, de modo que o processo retorna duas matrizes. As funções peso são $f_1(a_{ij}) = 1 - a_{ij}/\sum_k a_{ik}$ e $f_2(a_{ij}) = 1 - a_{ij}/\sum_k a_{kj}$.

```
56 def weight_fc(mat): #
57     n = len(mat)
58     tmp1 = np.zeros((n, n), dtype=float)
59     tmp2 = np.zeros((n, n), dtype=float)
60     for j in range(n):
61         sum_col = max(mat[:, j].sum(), 1)
62         sum_row = max(mat[j, :].sum(), 1)
63         for i in range(n):
64             tmp1[i, j] = 1 - float(mat[i, j]) / sum_col
65             tmp2[i, j] = 1 - float(mat[i, j]) / sum_row
66         tmp1[j, j] = 0
67         tmp2[j, j] = 0
68     return tmp1, tmp2
```

Por fim, executamos as duas principais funções do programa, a partir do arquivo de dados de migração 'migration_matrix.csv'.

```
69 migration_matrix_file = 'migration_matrix.csv'
70 states = np.loadtxt('prepare_data_states_num_counties.csv', dtype=str,
   ↪  delimiter=',')[:, 0]
71
72 loop_ultrametric_inner_state()
73
74 """ This function could take hours to finish. """
75 ultrametric_national()
76
77 """ End of file compute_ultrametric.py """
```

5.1.3 Criando os dendrogramas: `plot_dendrogram.py`

Nesta seção, apresentamos o último programa do projeto de migração nacional. O arquivo `plot_dendrogram.py` faz uso das matrizes de ultramétricas do diretório `ultra_metric` e cria no diretório `dendrograms` quatro dendrogramas para cada estado, e mais quatro para a migração nacional.

Como de costume, carregamos os módulos necessários, dos quais Matplotlib [6] e Scipy [8] não haviam sido usados até então. O primeiro, fornece ferramentas para criarmos/exportarmos figuras e/ou gráficos; o segundo, fornece funções para *clustering*.

```
1  # coding:utf-8
2
3  import matplotlib.pyplot as plt
4  import numpy as np
5  import pandas as pd
6  import sys
7  from scipy.cluster.hierarchy import dendrogram, linkage
8
9  sys.setrecursionlimit(2000) # to produce national dendrogram
```

A função `get_cmap()` depende de n e para cada $0 \leq i \leq n$ retorna uma cor RGB distinta do mapa de cores `name`, cujo padrão é `'hsv'`. Tais cores serão usadas nas legendas dos dendrogramas.

```
10 def get_cmap(n, name='hsv'):
11     return plt.cm.get_cmap(name, n)
```

A função `get_iso()` retorna o código ISO [7] de um dado estado, que será usado nas legendas dos dendrogramas.

```
12 def get_iso(state):
13     return len_states[np.where(len_states == state)[0][0], 2]
```

A função `get_len()` retorna o número de condados em um dado estado.

```
14 def get_len(state):
15     return int(len_states[np.where(len_states == state)[0][0], 1])
```

A função `loop_plot_dendrogram_by_state()` itera sobre todos os estados (exceto 'District of Columbia') e executa a função `plot_dendrogram_by_state()` neste estado, que gera os quatro dendrogramas para diferentes métodos e funções peso.

```
16 def loop_plot_dendrogram_by_state():
17     for state in states:
18         if state != 'District of Columbia':
19             plot_dendrogram_by_state(state) # defined in line 51
```

A função `plot_dendrogram_by_mat()`, a partir de uma matriz de distâncias ou ultramétrica, aplica técnicas de *clustering* para criar um dendrograma. Os argumentos `weight_fc` e `method` são responsáveis pelas informações necessárias para criarmos o dendrograma, e seus possíveis valores são `'col'` ou `row'`, e `'R'` ou `'NR'`, respectivamente. O argumento `state` recebe o nome do estado, se for o caso.

```
20 def plot_dendrogram_by_mat(mat, weight_fc='', method='', state='',
   ↪ label=np.array([])):
21     t = np.triu(mat, k=1)
22     tu = t[t > 0]
23     z = linkage(tu)
24     fig_width = 15
25 
26     if len(mat) > 200:
27         fig_width = 25
28     else:
29         if len(mat) > 100:
30             fig_width = 20
31     if state == 'USA':
32         fig_width = 400
33 
34     fig = plt.figure(figsize=(fig_width, 8))
35     fig.add_subplot(111)
36     plt.title('Hierarchical Clustering Dendrogram of %s (wf: %s, meth:
       ↪ %s)' % (state, weight_fc, method))
37     dendrogram(
38         z,
39         color_threshold=0.9 * max(z[:, 2]),
40         leaf_rotation=90,
41         leaf_font_size=7,
42         labels=label,
43     )
44     ax = plt.gca()
45     xlbls = ax.get_xmajorticklabels()
46     label_colors = {i: j for i, j in zip(label, colors)}
47     for lbl in xlbls:
48         lbl.set_color(label_colors[lbl.get_text()])
49     plt.savefig('dendrograms/%s_u%s_%s.pdf' % (weight_fc, method, state),
       ↪ bbox_inches="tight")
50     plt.close(fig)
```

A função `plot_dendrogram_by_state()` recebe um estado e itera sobre os métodos recíproco e não-recíproco e sobre as funções peso descritas em (5.1) e (5.2). Em cada iteração, carrega a matriz ultramétrica correspondente e executa a função `plot_dendrogram_by_mat()`.

```
51 def plot_dendrogram_by_state(state): #
52     print 'Creating all dendrograms for %s' % state
53     for method in ['R', 'NR']:
54         for weight_fc in ['col', 'row']:
55             mat = np.loadtxt('ultra_metric/%s_u%s_%s.csv' % (weight_fc,
               ↪ method, state.replace(' ', '_')), dtype=float,
               ↪ delimiter=',')
56             plot_dendrogram_by_mat(mat, weight_fc=weight_fc,
               ↪ method=method, state=state, label=dict_states_cts[state])
```

Analogamente, a função `plot_dendrogram_national()` itera sobre os métodos recíproco e não-recíproco e sobre as funções peso e executa `plot_dendrogram_by_mat()`

para os dados de migração nacional. Vale observar que, devido à quantia de condados, a figura do dendrograma precisa ser extremamente larga. Também, **limitações de memória do computador podem interromper o procedimento**.

```
57 def plot_dendrogram_national():
58     print 'Creating the national migration dendrogram'
59     for method in ['R', 'NR']:
60         for weight_fc in ['col', 'row']:
61             mat = np.loadtxt('ultra_metric/%s_u%s.csv' % (weight_fc,
                   ↪  method), dtype=float, delimiter=',')
62             plot_dendrogram_by_mat(mat, weight_fc=weight_fc,
                   ↪  method=method, state='USA', label=all_counties)
```

A função `use_pandas()`, como o nome sugere, utiliza o módulo *pandas* para gerar as variáveis `dict_states_cts` e `states`. A primeira é um dicionário Python, cuja chave é o nome de um estado e seu valor é uma lista com os condados deste estado. A segunda, é uma lista Python com os nomes dos estados.

```
63 def use_pandas():
64     global dict_states_cts, states
65     print 'Defining variables dict_states_cts, states'
66     df = pd.read_csv(raw_data_usa_file, encoding='latin1', header=None,
        ↪  delim_whitespace=False, index_col=None, sep=',',
        ↪  names=used_fields)
67     data = df[df[7] != '-'][used_fields]
68     states = data[4].unique().tolist()[:-9]
69
70     dict_states_cts = {}
71     for state in states:
72         tmp = data[data[4] == state][5].unique()
73         tmp = [cty.replace(' County', '').replace(' Municipio',
            ↪  '').replace(' city', '').replace(' Parish', '') for cty in
            ↪  tmp]
74         dict_states_cts[state] = tmp
```

Finalmente, após definidas as funções acima, o código a seguir carrega dois arquivos `.csv`, cria algumas variáveis (destacamos as listas Python `all_counties` e `color`) e executa as principais funções `use_pandas()`, `loop_plot_dendrogram_by_state()` e `plot_dendrogram_national()`.

```
75 raw_data_usa_file = 'raw_data_usa.csv'
76 len_states = np.loadtxt('prepare_data_states_num_counties.csv',
   ↪  dtype=str, delimiter=',')
77 used_fields = [4, 5, 6, 7, 8, 10]
78
79 use_pandas()
80
81 """ List of ISO codes + county names,
82     with commom words deleted to make them shorter """
83 all_counties = []
84 for state in states:
85     for county in dict_states_cts[state]:
```

```
86          tmp = u'%s %s' % (get_iso(state), county)
87          tmp = tmp.replace(' County', '').replace(' Municipio',
            ↪ '').replace(' city', '').replace(' Parish', '').replace('
            ↪ Census Area', '')
88          all_counties.append(tmp)
89
90  colors = []
91  cmap = get_cmap(len(states))
92  for i, state in enumerate(states):
93      colors += [cmap(i)] * get_len(state)
94
95  # required files: 'ultra_metric/{col,row}_u{R,NR}_{state}.csv'
96  loop_plot_dendrogram_by_state()
97
98  # required files: 'ultra_metric/{col,row}_u{R,NR}.csv'
99  plot_dendrogram_national()
```

5.2 Migração da população mundial

Nesta seção, faremos uma análise dos dados de migração mundial, criando ultramétricas e dendrogramas, como na seção anterior. O o programa `world-migration.py` é baseado nos descritos acima, com apenas alguns ajustes, pois os dados iniciais encontram-se em formato diferente. Assim, explicaremos somente os blocos de código que não foram já descritos.

5.2.1 Resumo dos dados iniciais

A análise será feita a partir dos dados do arquivo `worldbankdata.csv` obtido em [4]. Este arquivo possui sete colunas, das quais filtramos $1, 2, 5, 6, 7$, contendo:

país A cód. ISO país B cód. ISO fluxo de A para B

O número de países é 231, separados em 7 regiões (que chamaremos de continentes, por abuso de terminologia), coloridas na legenda do dendrograma, do seguinte modo:

- AF Africa
- AS Asia
- CA Central America
- EU Europe
- NA North America
- OC Oceania
- SA South America

Os dados de fluxo de A para B (e de B para A) são obtidos e guardados no arquivo `migration_matrix.csv`, que pode ser tratado como uma matriz quadrada de ordem 231. A seguir, com as mesmas funções do programa da seção anterior, as ultramétricas são calculadas para os dois métodos, recíproco e não-recíproco, e para as duas funções peso, como em (5.1) e (5.2). Por fim, quatro dendrogramas são criados.

5.2.2 Programa `world-migration.py`

A seguir, apresentamos na íntegra o programa `world-migration.py`, apenas fazendo observações nos pontos relevantes e diferentes.

```
# coding:utf-8

import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from scipy.cluster.hierarchy import dendrogram, linkage
```

A função `delete_zero_col_row()` é utilizada para eliminar linhas e/ou colunas nulas de uma matriz.

```
def delete_zero_col_row(mat):
    idx = []
    for i in range(len(mat)):
        if max(mat[i]) == 0:
            idx.append(i)

    mat = np.delete(mat, idx, 0)
    mat = np.delete(mat, idx, 1)
    return mat
```

O bloco a seguir apresenta as funções já explicadas na seção anterior e portanto não requer mais detalhes.

```
def max_prod(mat1, mat2):
    n = len(mat1)
    tmp = np.zeros((n, n), dtype=float)
    for i in range(n):
        for j in range(i, n):
            tmp[i, j] = min_max(mat1, mat2, i, j)
            tmp[j, i] = min_max(mat2, mat1, j, i)
    return tmp

def min_max(mat1, mat2, i, j):
    return min(np.maximum(mat1[i, :], mat2[:, j]))

def plot_dendrogram_by_mat(mat, weight_fc='', method='',
↪   label=np.array([])):
    t = np.triu(mat, k=1)
    tu = t[t > 0]
    z = linkage(tu)
    fig_width = 15

    if len(mat) > 200:
        fig_width = 25
    else:
        if len(mat) > 100:
            fig_width = 20

    fig = plt.figure(figsize=(fig_width, 8))
    fig.add_subplot(111)
    plt.title('Hierarchical Clustering Dendrogram (wf: %s, meth: %s)' %
    ↪   (weight_fc, method))
```

```
43      dendrogram(
44          z,
45          color_threshold=0.9 * max(z[:, 2]),
46          leaf_rotation=90,
47          leaf_font_size=7,
48          labels=label,
49      )
50      ax = plt.gca()
51      xlbls = ax.get_xmajorticklabels()
52      label_colors = {i: j for i, j in zip(label, colors)}
53      for lbl in xlbls:
54          lbl.set_color(label_colors[lbl.get_text()])
55      plt.savefig('%s_u%s.pdf' % (weight_fc, method), bbox_inches="tight")
56      plt.close(fig)
57
58  def plot_dendrogram_international():
59      print 'Creating the international migration dendrogram'
60      for method in ['R', 'NR']:
61          for weight_fc in ['col', 'row']:
62              mat = np.loadtxt('%s_u%s.csv' % (weight_fc, method),
                ↪  dtype=float, delimiter=',')
63              plot_dendrogram_by_mat(mat, weight_fc=weight_fc,
                ↪  method=method, label=iso_codes)
64
65  def power(mat):
66      tmp = mat
67      n = len(tmp)
68      k = 1
69      while 2 ** (k - 1) < n:
70          tmp = max_prod(tmp, tmp)
71          k += 1
72      return tmp
73
74  def u_nr(mat, output_file='uNR.csv'):
75      print 'Computing non-reciprocal ultra metric. Wait... ',
76      tmp1 = power(mat)
77      tmp2 = tmp1.T
78      tmp = np.maximum(tmp1, tmp2)
79      np.savetxt(output_file.replace(' ', '_'), tmp, fmt='%f',
        ↪  delimiter=',')
80      print 'saved in %s' % output_file.replace(' ', '_')
81      return tmp
82
83  def u_r(mat, output_file='uR.csv'):
84      print 'Computing     reciprocal ultra metric. Wait... ',
85      a_bar = np.maximum(mat, mat.T)
86      tmp = power(a_bar)
87      np.savetxt(output_file.replace(' ', '_'), tmp, fmt='%f',
        ↪  delimiter=',')
88      print 'saved in %s' % output_file.replace(' ', '_')
```

```
    return tmp

def ultrametric_international(mat):
    w_mat1, w_mat2 = weight_fc(mat)
    u_r(w_mat1, output_file='col_uR.csv')
    u_nr(w_mat1, output_file='col_uNR.csv')
    u_r(w_mat2, output_file='row_uR.csv')
    u_nr(w_mat2, output_file='row_uNR.csv')

def weight_fc(mat):
    n = len(mat)
    tmp1 = np.zeros((n, n), dtype=float)
    tmp2 = np.zeros((n, n), dtype=float)
    for j in range(n):
        sum_col = max(mat[:, j].sum(), 1)
        sum_row = max(mat[j, :].sum(), 1)
        for i in range(n):
            tmp1[i, j] = 1 - float(mat[i, j]) / sum_col
            tmp2[i, j] = 1 - float(mat[i, j]) / sum_row
        tmp1[j, j] = 0
        tmp2[j, j] = 0
    return tmp1, tmp2
```

O bloco a seguir define várias variáveis, as mais importantes sendo `raw_data_file` e `migration_matrix`. As demais partes são detalhes técnicos, porém de fácil compreensão.

```
raw_data_file = 'worldbankdata.csv'
df = pd.read_csv(raw_data_file, header=0, delim_whitespace=False,
↪  index_col=0, sep=',')
iso_codes = df['Country Origin Code'].unique()
iso_codes_continents = np.loadtxt('codes.csv', dtype=str, delimiter=',')
iso_continents = sorted(list(set(iso_codes_continents[:, 1])))
num_countries = len(iso_codes)
num_continents = len(iso_continents)
migration_matrix =
↪  df['value'].values.astype(float).reshape(num_countries,
↪  num_countries)
migration_matrix = delete_zero_col_row(migration_matrix)
np.savetxt('migration_matrix.csv', migration_matrix, fmt='%d',
↪  delimiter=',')

colors = []
palette = ['orange', 'red', 'black', 'green', 'cyan', 'magenta', 'blue']
for i, country in enumerate(iso_codes):
    cont = iso_codes_continents[i, 1]
    j = iso_continents.index(cont)
    colors.append(palette[j])
```

Por fim, executamos as duas funções principais, que por sua vez, fazem uso das demais funções acima. A função `ultrametric_international()` recebe a variável `migration_matrix` e cria as ultramétricas de fluxo migratório internacional. A fun-

ção `plot_dendrogram_international()`, como o nome já diz, cria arquivos `.pdf` dos quatro dendrogramas.

```
128 ultrametric_international(migration_matrix)
129 plot_dendrogram_international()
```

5.3 *Ranking*

5.3.1 Programa `ranking_world_migration.py`

Neste programa, ainda sobre migração mundial, faremos uma análise para determinar quais países foram mais frequentes no fluxo migratório.

Mais precisamente, dado um país P, filtramos cinco países $B_1, \ldots, B_5$ para os quais o fluxo de migração $\mu(P, B_i)$ de P para B_i é máximo, ou seja, filtramos os cinco países que mais receberam população vinda de P. Analogamente, dualizamos o processo para obtermos $A_1, \ldots, A_5$ para os quais o fluxo de migração $\mu(A_i, P)$ de A_i para P é máximo.

Um resumo dos dados obtidos pode ser visto na Tabela C.1 da página 81.

Por fim, a partir da matriz de distâncias, criamos uma filtração F_δ para $\delta \in \{\frac{7}{10}, \frac{8}{10}, \frac{9}{10}\}$, onde

$$F_\delta = \{(A, B);\ \mu(A, B) \leq \delta\}.$$

Assim, podemos analisar se os países mais próximos com relação à migração também são mais próximos no sentido geográfico, ou seja, se possuem fronteira comum ou se pertencem a um mesmo continente.

Como de costume, carregamos os módulos necessários.

```
1 # coding:utf-8
2
3 import numpy as np
4 import pandas as pd
```

A função `filtration()` utiliza a matriz de distâncias para criar a filtração F_δ citada acima. Também, retorna o número de pares de países na filtração que pertencem a um mesmo continente.

```
5  def filtration(mat):
6      delta_partition = [.7, .8, .9]
7      for delta in delta_partition:
8          same_continent = 0
9          idx, idy = np.where(mat <= delta)
10         pairs = zip(idx, idy)
11         for p in pairs:
12             tmp = ''
13             if p[0] < p[1]:
14                 if iso_codes[p[0], 1] == iso_codes[p[1], 1]:
15                     same_continent += 1
16                     tmp = '** '
17                 print '%sdist %s: %s - %s  %s-%s' % (tmp, mat[p[0],
                    ↪  p[1]], countries[p[0]], countries[p[1]],
                    ↪  iso_codes[p[0], 1], iso_codes[p[1], 1])
```

```
18          print '%d/%d = %.02f belongs to same continent\n' %
            ↪  (same_continent, len(pairs)/2,
            ↪  float(same_continent)/(len(pairs)/2))
```

A função `loop_top_migration_by_country()` possui um laço que executa a função `top_world_migration_by_country()` para cada país `country` na variável `countries`.

```
19 def loop_top_migration_by_country():
20     for country in countries:
21         print 'Ranking migration flow for %s' % country
22         top_world_migration_by_country(country)
```

A função `top_list()` é interessante e requer uma explicação mais detalhada.

Ao executarmos a função `top_world_migration_by_country()`, duas variáveis `top_in_list` e `top_out_list` são atualizadas, ou seja, cada uma recebe os cinco países de maior fluxo migratório com o país `country` em questão, como exemplificado nas Figuras 5.1a e 5.1b.

Após o laço realizado para cada país, as duas variáveis acima guardam todos os países que apareceram dentre os cinco primeiros. É aqui que a função `top_list()` faz o seu papel retornando, tanto para imigração quanto para emigração, os três países que mais aparecem dentre os *top five*.

Informalmente, podemos pensar que, após todas as rodadas, são premiados nas categorias 'Imigração' e 'Emigração' com Ouro, Prata e Bronze os que mais frequentaram o pódio durante a competição (com o pódio sendo formado por cinco competidores).

```
23 def top_list():
24     unique, counts = np.unique(top_in_list, return_counts=True)
25     print 'top in:\n', sorted(zip(counts, unique))[-3:]
26     unique, counts = np.unique(top_out_list, return_counts=True)
27     print 'top out:\n', sorted(zip(counts, unique))[-3:]
```

A função `top_world_migration_by_country()`, para cada país `country`, determina os (cinco) países com maior fluxo de migração *para* `country` e *a partir* de `country`. Ainda, salva os resultados nos arquivos `ranking/top_from_<country>.csv` e `ranking/top_to_<country>.csv`.

```
28 def top_world_migration_by_country(country, save_csv=False):
29     from_ = data[data['from'] == country]
30     top_from = from_.sort_values(['value'],
       ↪  ascending=False).head(5)[['from', 'to', 'value']]
31     to_ = data[data['to'] == country]
32     top_to = to_.sort_values(['value'], ascending=False).head(5)[['to',
       ↪  'from', 'value']]
33
34     for val in top_from['to'].values:
35         top_in_list.append(val)
36     for val in top_to['from'].values:
37         top_out_list.append(val)
38
39     if save_csv:
```

```
40          top_from.to_csv('ranking/top_from_%s.csv' % country, sep=',',
            ↪  encoding='utf-8', index=False, header=False)
41          top_to.to_csv('ranking/top_to_%s.csv' % country, sep=',',
            ↪  encoding='utf-8', index=False, header=False)
```

A função `top_world_migration()` simplesmente retorna os *top ten*, ou seja, os dez países com maior fluxo de imigração e emigração. Ainda, salva os resultados nos arquivos `ranking/top_out.csv` e `ranking/top_in.csv`.

```
42 def top_world_migration(save_csv=False):
43     """ This function saves the top 10 countries
44     with higher incoming/outcoming flow. """
45     top_out = data.groupby(['from'],
       ↪  as_index=False).sum().sort_values(['value'],
       ↪  ascending=False).head(10)[['from', 'value']].astype(object)
46     top_in = data.groupby(['to'],
       ↪  as_index=False).sum().sort_values(['value'],
       ↪  ascending=False).head(10)[['to', 'value']].astype(object)
47     if save_csv:
48         top_out.to_csv('ranking/top_out.csv', sep=',', encoding='utf-8',
           ↪  index=False, header=False)
49         top_in.to_csv('ranking/top_in.csv', sep=',', encoding='utf-8',
           ↪  index=False, header=False)
```

A seguir, consta a parte principal do programa, onde o arquivo `worldbankdata.csv` de dados bruto ([4]) é carregado e as funções descritas acima são executadas. Também, é carregado o arquivo `row_uNR.csv` (ou os outros três possíveis) da matriz da ultramétrica, que é utilizada pela função `filtration()`.

```
50 """ Source: Global Bilateral Migration
51     Data from 2000
52     Last Updated: 06/28/2011 """
53 raw_data_file = 'worldbankdata.csv'
54 col_names = ['from', 'iso_from', 'tmp1', 'tmp2', 'to', 'iso_to', 'value']
55 df = pd.read_csv(raw_data_file, encoding='utf8', header=0,
   ↪  delim_whitespace=False, index_col=None, sep=',', names=col_names)
56 data = df[['from', 'iso_from', 'to', 'iso_to', 'value']]
57 countries = df['from'].unique().tolist()
58 iso_codes = np.loadtxt('codes.csv', delimiter=',', dtype=str)
59 top_in_list = []
60 top_out_list = []
61
62 top_world_migration()
63 loop_top_migration_by_country()
64 top_list()
65
66 mat = np.loadtxt('row_uNR.csv', delimiter=',')
67 filtration(mat)
```

5.3.2 Classificando os dados: `ranking_USA_migration.py`

Neste programa, faremos a mesma análise da Seção 5.3.1 para determinar quais estados foram mais frequentes no fluxo migratório.

Nesse caso, dado um estado S, filtramos cinco estados $E_1, \ldots, E_5$ para os quais o fluxo de migração $\mu(S, E_i)$ de S para E_i é máximo. Analogamente, dualizamos o processo para obtermos $E_1, \ldots, E_5$ para os quais o fluxo de migração $\mu(E_i, S)$ de E_i para S é máximo.

Como o nosso banco de dados `raw_data_usa.csv` ([2]) é muito maior, não faremos a filtração para analisar se os estados próximos com relação à migração também são mais próximos no sentido geográfico.

Primeiro, carregamos os módulos necessários.

```
1  # coding:utf-8
2
3  import numpy as np
4  import pandas as pd
```

A função `loop_top_migration_by_state()` é análoga à função da seção anterior `loop_top_migration_by_country()`.

```
5  def loop_top_migration_by_state():
6      for state in states:
7          print 'Ranking migration flow for %s' % state
8          top_migration_by_state(state)
```

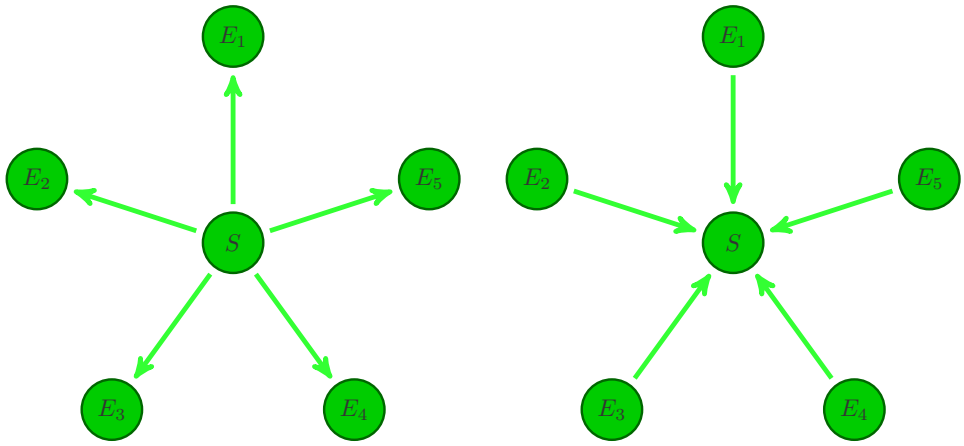
A função `top_list()` retorna os três estados de maior fluxo migratório.

```
9  def top_list():
10     unique, counts = np.unique(top_from_list, return_counts=True)
11     print 'top out:\n', sorted(zip(counts, unique))[-3:]
12     unique, counts = np.unique(top_to_list, return_counts=True)
13     print 'top in:\n', sorted(zip(counts, unique))[-3:]
```

A função `top_migration_by_state()` é análoga à função `top_world_migration_by_country()` e salva os resultados nos arquivos `ranking/top_from_<state>.csv` e `ranking/top_to_<state>.csv`.

```
14 def top_migration_by_state(state, save_csv=False):
15     tmp = data[data[4] == state]
16     top_from = tmp.groupby([6], as_index=False).sum().sort_values([10],
       ↪  ascending=False).head(5)[[6, 10]]
17     top_to = tmp.groupby([6], as_index=False).sum().sort_values([8],
       ↪  ascending=False).head(5)[[6, 8]]
18
19     for val in top_from[6].values:
20         top_from_list.append(val)
21     for val in top_to[6].values:
22         top_to_list.append(val)
23
24     if save_csv:
25         top_from.to_csv('ranking/top_from_%s.csv' % state, sep=',',
           ↪  encoding='utf-8', index=False, header=False)
```

```
26          top_to.to_csv('ranking/top_to_%s.csv' % state, sep=',',
            ↪  encoding='utf-8', index=False, header=False)
```



(a) `top_to_list` (b) `top_from_list`

Figura 5.1: Representação da função `top_migration_by_state()`

A função `top_migration_national()` retorna os *top ten* dos estados com maior fluxo migratório, listados na Tabela C.3 da página 82. Ainda salva os resultados nos arquivos `ranking/top_in_USA.csv` e `ranking/top_out_USA.csv`.

```
27 def top_migration_national(save_csv=False):
28     print 'Ranking national migration...',
29     top_in = data.groupby([6, 7], as_index=False).sum().sort_values([10],
       ↪  ascending=False).head(10)[[6, 7, 10]]
30     top_out = data.groupby([6, 7], as_index=False).sum().sort_values([8],
       ↪  ascending=False).head(10)[[6, 7, 8]]
31     if save_csv:
32         top_in.to_csv('ranking/top_in_USA.csv', sep=',',
           ↪  encoding='utf-8', index=False, header=False)
33         top_out.to_csv('ranking/top_out_USA.csv', sep=',',
           ↪  encoding='utf-8', index=False, header=False)
34         print 'done.'
```

Finalmente, a parte principal do programa, onde os arquivos `raw_data_usa.csv` e `prepare_data_states_num_counties.csv` são carregados e as funções acima são executadas.

```
35 """ MAIN CODE """
36
37 raw_data_usa_file = 'raw_data_usa.csv'
38 states = np.loadtxt('prepare_data_states_num_counties.csv', dtype=str,
   ↪  delimiter=',')[:, 0]
39
40 used_fields = [4, 5, 6, 7, 8, 10]
41 df = pd.read_csv(raw_data_usa_file, encoding='latin1', header=None,
   ↪  delim_whitespace=False, index_col=None, sep=',', names=used_fields)
42 data = df[(df[5] != '-') & (df[7] != '-')][used_fields]
43
```

```
top_from_list = []
top_to_list = []

top_migration_national()
loop_top_migration_by_state()
top_list()
```

Referências

[1] G. Carlsson, F. Mémoli, A. Ribeiro, and S. Segarra. Axiomatic construction of hierarchical clustering in asymmetric networks. *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference*, pages 5219–5223, 2013.

[2] County-to-county migration flows: 2011-2015 acs, 2018.

[3] T. de Melo and B. Barreiro. Python program to compute ultra-metric for migration networks, 2018.

[4] Global Bilateral Migration Database, 2018. [Online; acessado em 14 de dezembro de 2018].

[5] M. Gondran and M. Minoux. *Graphs, dioids and semi rings: New models and algorithms*. Springer, New York, 2008.

[6] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing In Science & Engineering*, 9(3):90–95, 2007.

[7] Iso 3166-1 — Wikipedia, the free encyclopedia, 2018. [Online; acessado em 14 de dezembro de 2018].

[8] E. Jones, T. Oliphant, P. Peterson, et al. SciPy: Open source scientific tools for Python, 2001–. [Online; acessado em 14 de dezembro de 2018].

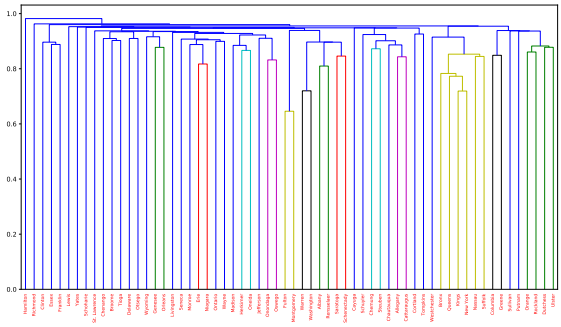
[9] W. McKinney. Data structures for statistical computing in python. Proceedings of the 9th Python in Science Conference, 2010–. [Online; acessado em 14 de dezembro de 2018].

[10] Migracão — Wikipedia, the free encyclopedia, 2019. [Online; acessado em 11 de janeiro de 2019].

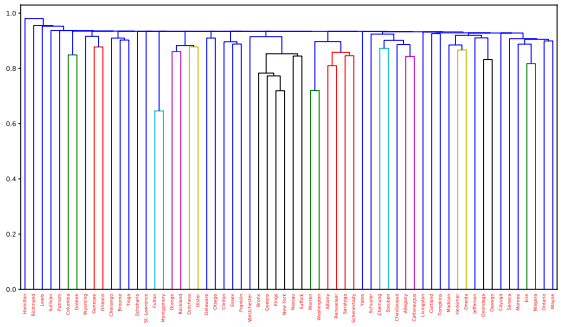
[11] T. Oliphant. NumPy: A guide to NumPy. USA: Trelgol Publishing, 2006–. [Online; acessado em 14 de dezembro de 2018].

A Dendrogramas

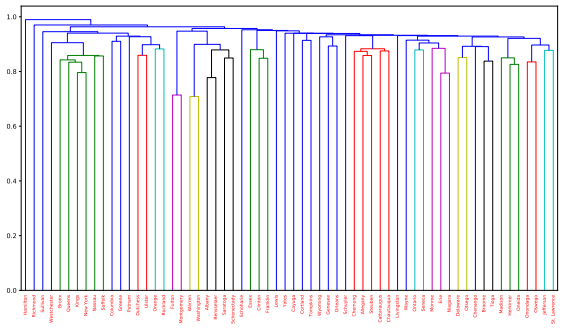
Alguns dos vários dendrogramas criados com os programas do Capítulo 5.



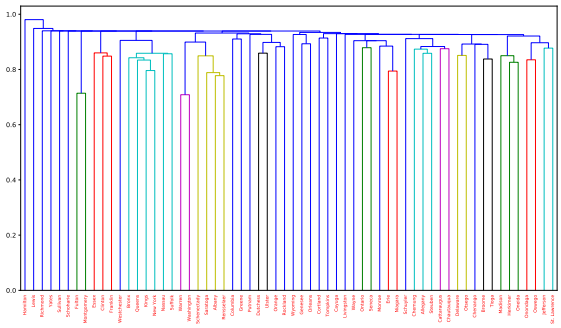
(a) Método recíproco, função peso f_1



(b) Método não-recíproco, função peso f_1



(c) Método recíproco, função peso f_2



(d) Método não-recíproco, função peso f_2

Figura A.1: Fluxo migratório do estado de Nova Iorque

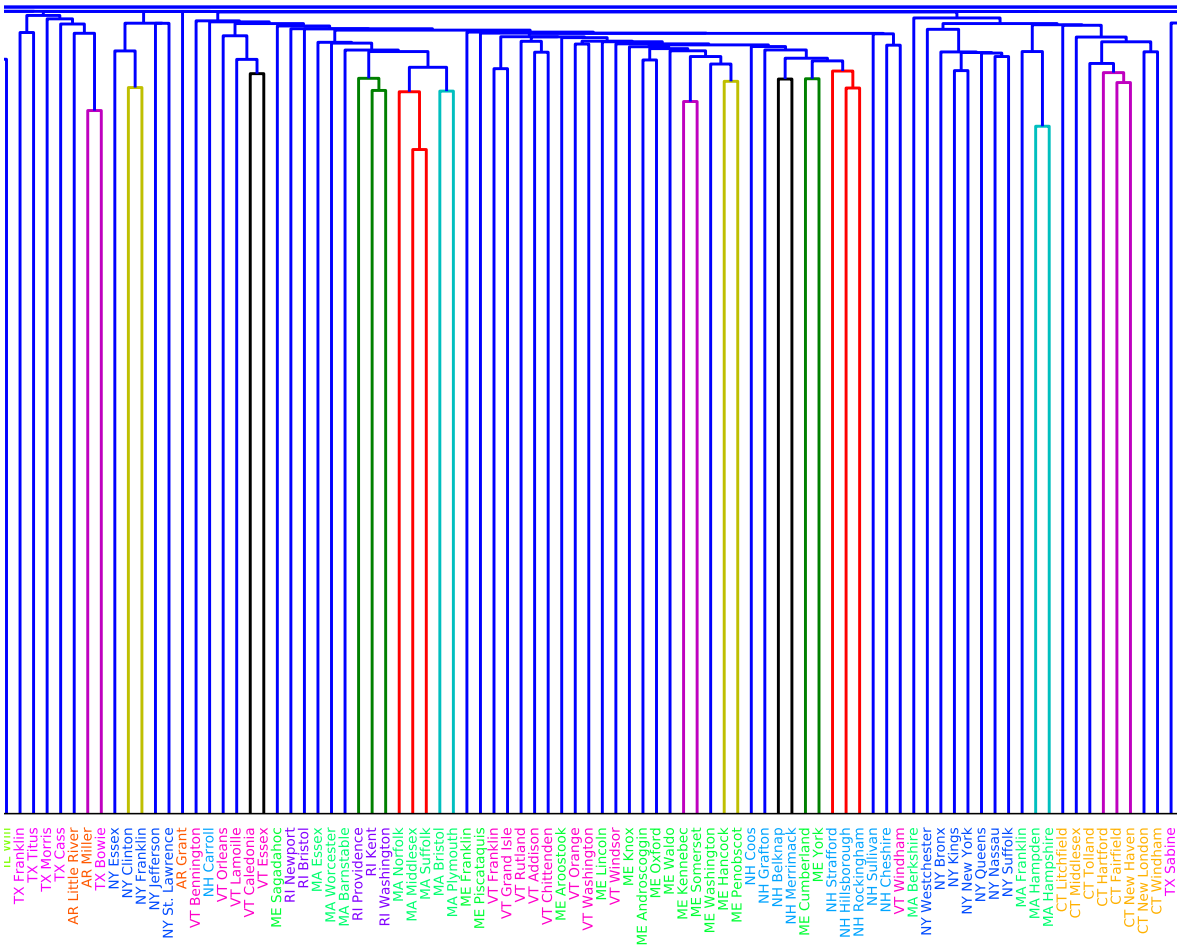


Figura A.2: Dendrograma (parcial) de migração nacional dos EUA, método recíproco, função peso f_2

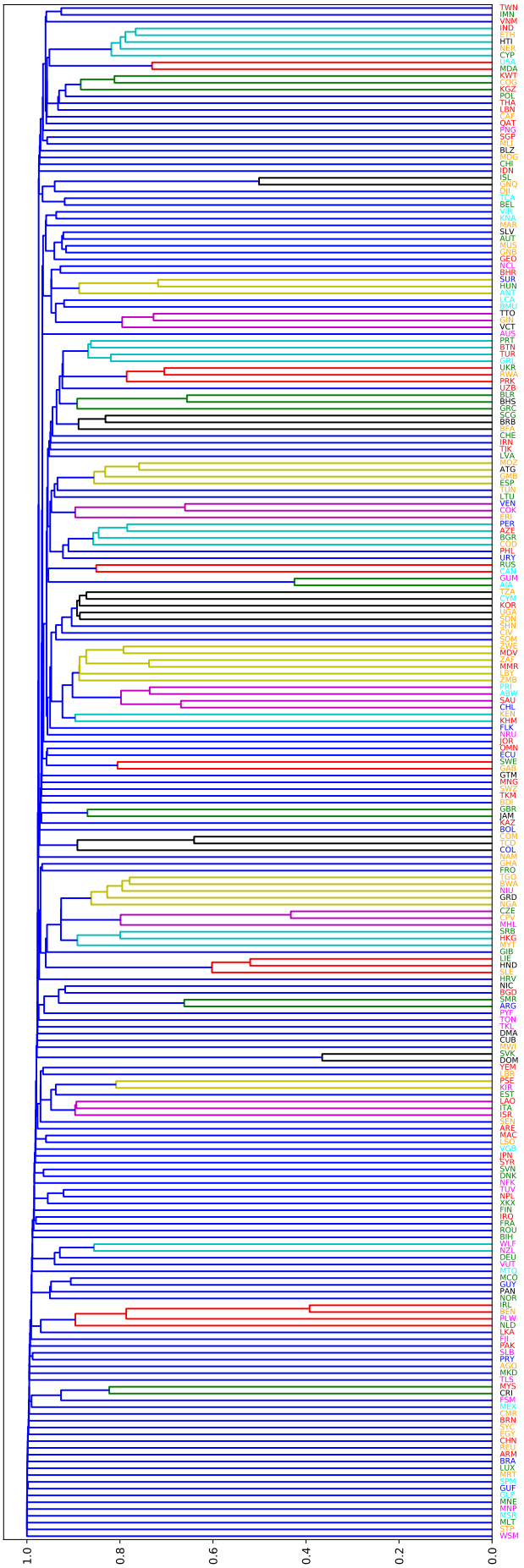


Figura A.3: Migração mundial, método recíproco, função peso f_1

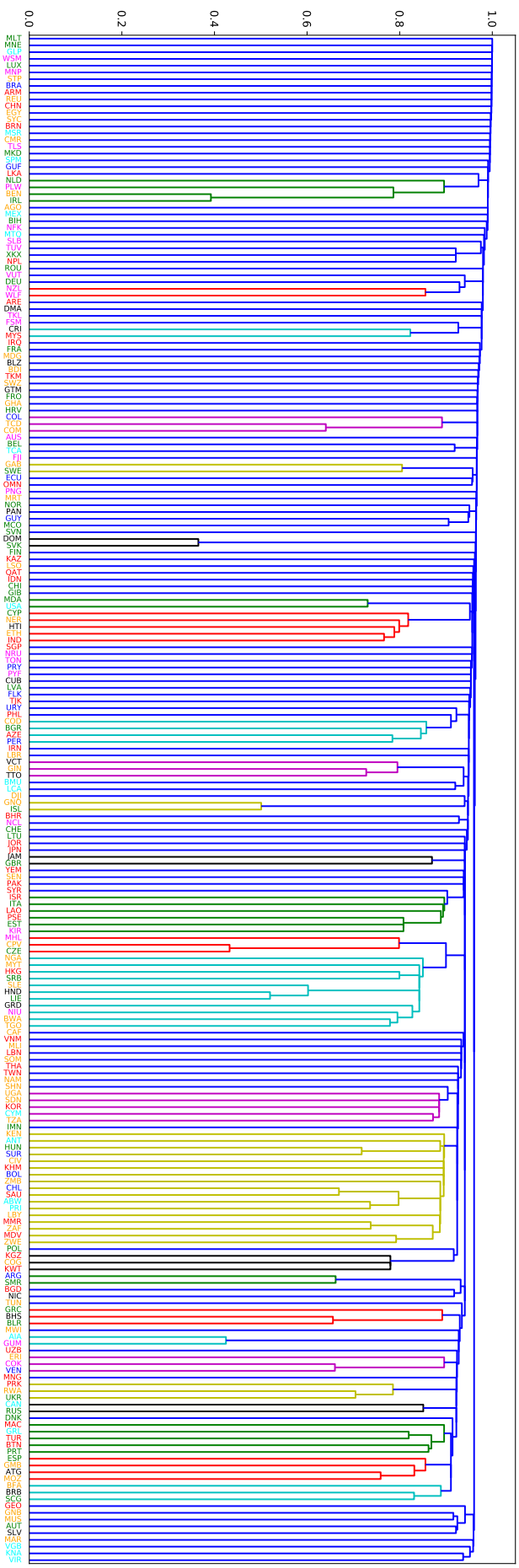


Figura A.4: Migração mundial, método não-recíproco, função peso f_1

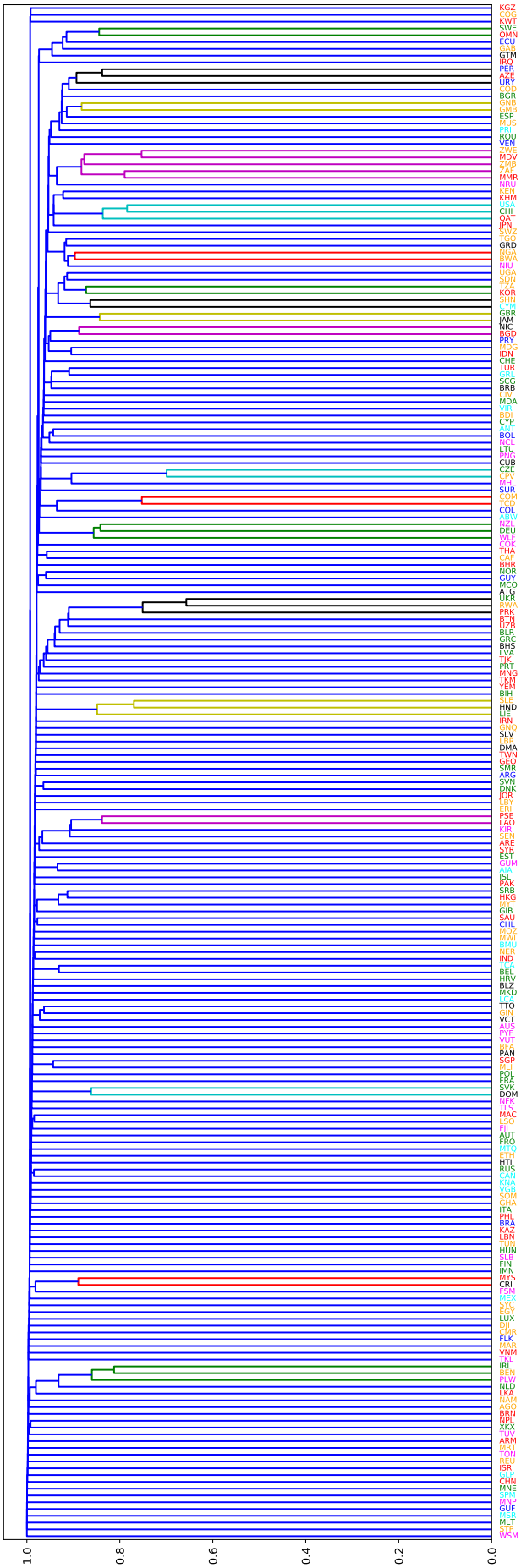


Figura A.5: Migração mundial, método recíproco, função peso f_2

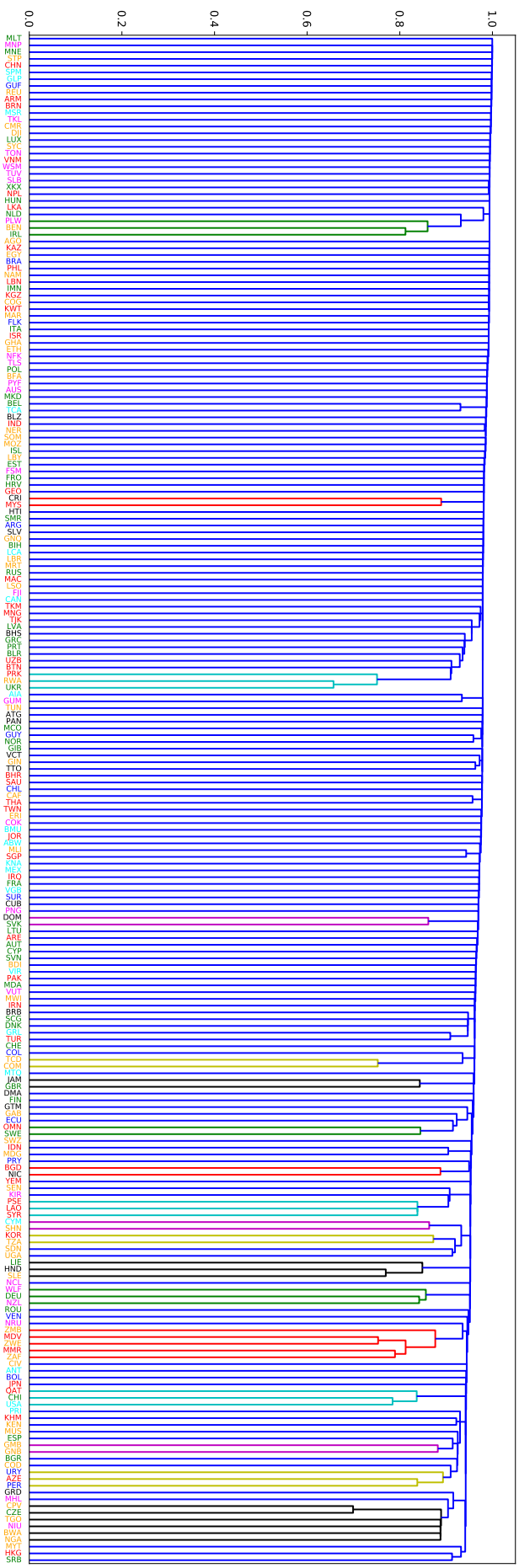


Figura A.6: Migração mundial, método não-recíproco, função peso f_2

B Países, códigos ISO e continentes

País (em inglês)	**ISO**	**Cont.**	**País** (em inglês)	**ISO**	**Cont.**
Afghanistan	AFG	AS	Luxembourg	LUX	EU
Albania	ALB	EU	Macau	MAC	AS
Algeria	DZA	AF	Macedonia	MKD	EU
American Samoa	ASM	OC	Madagascar	MDG	AF
Andorra	AND	EU	Malawi	MWI	AF
Angola	AGO	AF	Malaysia	MYS	AS
Anguilla	AIA	NA	Maldives	MDV	AS
Antigua & Barbuda	ATG	CA	Mali	MLI	AF
Argentina	ARG	SA	Malta	MLT	EU
Armenia	ARM	AS	Marshall Islands	MHL	OC
Aruba	ABW	NA	Martinique	MTQ	NA
Australia	AUS	OC	Mauritania	MRT	AF
Austria	AUT	EU	Mauritius	MUS	AF
Azerbaijan	AZE	AS	Mayotte	MYT	AF
Bahamas	BHS	CA	Mexico	MEX	NA
Bahrain	BHR	AS	Micronesia	FSM	OC
Bangladesh	BGD	AS	Moldova	MDA	EU
Barbados	BRB	CA	Monaco	MCO	EU
Belarus	BLR	EU	Mongolia	MNG	AS
Belgium	BEL	EU	Montenegro	MNE	EU
Belize	BLZ	CA	Montserrat	MSR	NA
Benin	BEN	AF	Morocco	MAR	AF
Bermuda	BMU	NA	Mozambique	MOZ	AF
Bhutan	BTN	AS	Myanmar	MMR	AS
Bolivia	BOL	SA	Namibia	NAM	AF
Bosnia	BIH	EU	Nauru	NRU	OC
Botswana	BWA	AF	Nepal	NPL	AS
Brazil	BRA	SA	Netherlands	NLD	EU
Brunei	BRN	AS	Netherlands Antilles	ANT	NA
Bulgaria	BGR	EU	New Caledonia	NCL	OC
Burkina Faso	BFA	AF	New Zealand	NZL	OC
Burundi	BDI	AF	Nicaragua	NIC	CA
Cambodia	KHM	AS	Niger	NER	AF
Cameroon	CMR	AF	Nigeria	NGA	AF
Canada	CAN	NA	Niue	NIU	OC

Cape Verde	CPV	AF
Cayman Islands	CYM	NA
Central African Republic	CAF	AF
Chad	TCD	AF
Channel Islands	CHI	EU
Chile	CHL	SA
China	CHN	AS
Colombia	COL	SA
Comoros	COM	AF
Congo–Kinshasa	COD	AF
Congo–Brazzaville	COG	AF
Cook Islands	COK	OC
Costa Rica	CRI	CA
Côte d'Ivoire	CIV	AF
Croatia	HRV	EU
Cuba	CUB	CA
Cyprus	CYP	EU
Czechia	CZE	EU
Denmark	DNK	EU
Djibouti	DJI	AF
Dominica	DMA	CA
Dominican Republic	DOM	CA
Ecuador	ECU	SA
Egypt	EGY	AF
El Salvador	SLV	CA
Equatorial Guinea	GNQ	AF
Eritrea	ERI	AF
Estonia	EST	EU
Ethiopia	ETH	AF
Faroe Islands	FRO	EU
Falkland Islands	FLK	SA
Fiji	FJI	OC
Finland	FIN	EU
France	FRA	EU
French Guiana	GUF	SA
French Polynesia	PYF	OC
Gabon	GAB	AF
Gambia	GMB	AF
Georgia	GEO	AS
Germany	DEU	EU
Ghana	GHA	AF
Gibraltar	GIB	EU
Greece	GRC	EU
Greenland	GRL	NA
Grenada	GRD	CA
Guadeloupe	GLP	NA
Guam	GUM	OC
Guatemala	GTM	CA
Norfolk Island	NFK	OC
Northern Mariana Islands	MNP	OC
North Korea	PRK	AS
Norway	NOR	EU
Oman	OMN	AS
Pakistan	PAK	AS
Palau	PLW	OC
Panama	PAN	CA
Papua New Guinea	PNG	OC
Paraguay	PRY	SA
Peru	PER	SA
Philippines	PHL	AS
Poland	POL	EU
Portugal	PRT	EU
Puerto Rico	PRI	NA
Qatar	QAT	AS
Réunion	REU	AF
Romania	ROU	EU
Russia	RUS	EU
Rwanda	RWA	AF
St. Helena	SHN	AF
St. Pierre & Miquelon	SPM	NA
Samoa	WSM	OC
San Marino	SMR	EU
São Tomé & Príncipe	STP	AF
Saudi Arabia	SAU	AS
Senegal	SEN	AF
Serbia	SRB	EU
Serbia and Montenegro	SCG	EU
Seychelles	SYC	AF
Sierra Leone	SLE	AF
Singapore	SGP	AS
Slovakia	SVK	EU
Slovenia	SVN	EU
Solomon Islands	SLB	OC
Somalia	SOM	AF
South Africa	ZAF	AF
South Korea	KOR	AS
Spain	ESP	EU
Sri Lanka	LKA	AS
St. Kitts & Nevis	KNA	NA
St. Lucia	LCA	NA
St. Vincent & Grenadines	VCT	CA
Sudan	SDN	AF
Suriname	SUR	SA
Swaziland	SWZ	AF
Sweden	SWE	EU
Switzerland	CHE	EU

Guinea	GIN	AF	Syria	SYR	AS
Guinea-Bissau	GNB	AF	Taiwan	TWN	AS
Guyana	GUY	SA	Tajikistan	TJK	AS
Haiti	HTI	CA	Tanzania	TZA	AF
Honduras	HND	CA	Thailand	THA	AS
Hong Kong	HKG	AS	Timor-Leste	TLS	OC
Hungary	HUN	EU	Togo	TGO	AF
Iceland	ISL	EU	Tokelau	TKL	OC
India	IND	AS	Tonga	TON	OC
Indonesia	IDN	AS	Trinidad & Tobago	TTO	CA
Iran	IRN	AS	Tunisia	TUN	AF
Iraq	IRQ	AS	Turkey	TUR	AS
Ireland	IRL	EU	Turkmenistan	TKM	AS
Isle of Man	IMN	EU	Turks & Caicos Islands	TCA	NA
Israel	ISR	AS	Tuvalu	TUV	OC
Italy	ITA	EU	Uganda	UGA	AF
Jamaica	JAM	CA	Ukraine	UKR	EU
Japan	JPN	AS	United Arab Emirates	ARE	AS
Jordan	JOR	AS	UK	GBR	EU
Kazakhstan	KAZ	AS	US	USA	NA
Kenya	KEN	AF	Uruguay	URY	SA
Kiribati	KIR	OC	Uzbekistan	UZB	AS
Kosovo	XKX	EU	Vanuatu	VUT	OC
Kuwait	KWT	AS	Venezuela	VEN	SA
Kyrgyzstan	KGZ	AS	Vietnam	VNM	AS
Laos	LAO	AS	U.S. Virgin Islands	VIR	NA
Latvia	LVA	EU	British Virgin Islands	VGB	NA
Lebanon	LBN	AS	Wallis & Futuna	WLF	OC
Lesotho	LSO	AF	Palestine	PSE	AS
Liberia	LBR	AF	Yemen	YEM	AS
Libya	LBY	AF	Zambia	ZMB	AF
Liechtenstein	LIE	EU	Zimbabwe	ZWE	AF
Lithuania	LTU	EU			

Tabela B.1: Lista de países, códigos ISO e continentes

C *Ranking* de migração

C.1 Migração mundial

De acordo com [10], os termos *imigração* e *emigração* referem-se à entrada e saída de pessoas em um determinado país, respectivamente.

top ten de imigração			*top ten* de emigração		
País (em inglês)	***in***	**Cont.**	**País** (em inglês)	***out***	**Cont.**
United States	34.814.064	NA	Russian Federation	10.375.787	AS
Russian Federation	12.051.167	AS	Mexico	9.550.629	NA
Germany	11.134.583	EU	India	9.516.831	AS
France	6.278.721	EU	Ukraine	5.915.970	EU
India	6.235.774	AS	China	5.814.587	AS
Canada	5.555.024	NA	Poland	5.147.176	EU
Ukraine	5.206.456	EU	Bangladesh	4.987.708	AS
Saudi Arabia	5.130.955	AS	United Kingdom	4.061.775	EU
United Kingdom	4.891.311	EU	Pakistan	3.812.237	AS
Australia	4.027.479	OC	Germany	3.602.196	EU

Tabela C.1: Lista de dez países com maior fluxo migratório de imigrantes (esquerda) e de emigrantes (direita)

AF Africa	AS Asia
CA Central America	EU Europe
NA North America	OC Oceania
SA South America	

Tabela C.2: Códigos ISO e cores das regiões continentais

C.2 Migração dos EUA

top ten de imigração			*top ten* de emigração		
Condado (inglês)	***in***	**Estado**	**Condado** (inglês)	***out***	**Estado**
Los Angeles	217.592	California	Los Angeles	307.170	California
Maricopa	174.378	Arizona	Cook	205.190	Illinois
Harris	166.462	Texas	Harris	164.985	Texas
Cook	142.078	Illinois	Maricopa	159.248	Arizona
San Diego	120.373	California	San Diego	151.968	California
Riverside	114.665	California	Dallas	132.260	Texas
Dallas	113.831	Texas	Orange	122.239	California
Orange	111.285	California	Kings	121.804	New York
San Bernardino	110.644	California	New York	121.404	New York
King	109.650	Washington	King	104.179	Washington

Tabela C.3: Lista de dez condados com maior fluxo migratório de imigrantes (esquerda) e de emigrantes (direita)

UNIVERSIDADE ESTADUAL PAULISTA
"JÚLIO DE MESQUITA FILHO"
Campus de São José do Rio Preto

TERMO DE REPRODUÇÃO XEROGRÁFICA

Autorizo a reprodução xerográfica do presente Trabalho de Conclusão, na íntegra ou em partes, para fins de pesquisa.

São José do Rio Preto, 18/02/2019

Bianca Barreiro

Assinatura do autor