FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Geração Aleatória de Estruturas Classificatórias

Vasco Manuel Pedro Machado

Licenciado em Matemática – Ramo Educacional pela
Faculdade de Ciências da Universidade do Porto

Dissertação submetida para satisfação parcial dos requisitos do grau
de mestre em
Estatística Aplicada e Modelação

Dissertação realizada sob a supervisão da
Professora Doutora Fernanda Sousa,
do Departamento de Engenharia Civil da
Faculdade de Engenharia da Universidade do Porto

Porto, Janeiro de 2007

Resumo

A Análise Classificatória é uma área da Análise de Dados Multivariados cujo objectivo é agrupar um conjunto de objectos a classificar num número pequeno de classes, que reflitam as relações de semelhança e/ou oposição entre esses objectos. Um método de Classificação produz, sobre estes objectos, um conjunto de classes organizado segundo uma determinada estrutura. Duas metodologias de Classificação serão analisadas: a Classificação Hierárquica e a Classificação Piramidal.

A aplicação de um método de Classificação Piramidal produz estruturas mais complexas do que a aplicação de um método de Classificação Hierárquica. Assim, inicialmente será introduzido um vasto conjunto de definições e propriedades, apresentados os principais algoritmos de Classificação Piramidal Ascendente, serão referidas as suas características e limitações.

A motivação para desenvolver algoritmos de geração de estruturas classificatórias tem-se intensificado entre a comunidade científica, tendo dado origem a diversos estudos. Neste trabalho propõe-se um método de geração aleatória de estruturas piramidais. Sendo o modelo de Classificação Piramidal uma generalização do modelo de Classificação Hierárquica, o método aqui proposto poderá ser visto como uma extensão de trabalhos anteriores de geração aleatória de dendrogramas.

Tendo por objectivo avaliar o desempenho do método proposto, em particular, verificar se o método gera pirâmides aleatória e uniformemente, foi previamente realizado um estudo topológico de pirâmides. Para um número fixo de nós terminais identificam-se os diferentes tipos topológicos de pirâmides, assim como o respectivo número de pirâmides não isomórficas. Em virtude da elevada complexidade desta análise, este objectivo foi apenas conseguido para um número reduzido de nós terminais.

Abstract

The Cluster Analysis is an area of the Multivariate Data Analysis whose the main goal is to group a set of elements in a small number of classes, that they reflect the relationships of similarity and/or opposition among those elements. A method of Classification produces, on these elements, a group of classes organized second a certain structure. Two methodologies of Classification will be analyzed: the Hierarchical Classification and the Pyramidal Classification.

The application of a Pyramidal Classification method produces more complex structures than the application of a Hierarchical Classification method. First, it will be introduced a vast group of definitions and properties and presented the main sort algorithms of Ascendant Pyramidal Classification.

The motivation to develop generation algorithms of classification structures have been intensifying among the scientific community, having created several studies. In this work a random generation method of pyramidal structures is proposed. Being the model of Pyramidal Classification a generalization of the model of Hierarchical Classification, the method proposed appears as an extension of previous works in random generation of dendrograms.

With the purpose to evaluate the performance of the proposed method, in particular, to verify if the method generates random and uniform pyramids, it was previously accomplished a topologic study of the pyramids. For a fixed number of terminal nodes the different topologic types of pyramids were identified, as well as the respective number of non isomorphic pyramids. On account of the high complexity of this analysis, this purpose was just gotten for a reduced number of terminal nodes.

Agradecimentos

Nesta página vou referir aquelas pessoas e instituições que directa ou indirectamente contribuíram para a realização desta dissertação.

À Professora Fernanda Sousa, minha orientadora, um agradecimento muito especial por todo o tempo dispendido, pelas sugestões e por ter compreendido todos os meus atrasos. Gostaria de dizer MUITO OBRIGADA pela total disponibilidade manifestada.

À FEUP, em particular à Direcção do MEAM, quero agradecer pelas condições que me facultaram para que este trabalho pudesse ser realizado.

Ao Jorge, agradeço pela troca de ideias em alguns temas do trabalho e pela ajuda na escrita, em LaTex, da dissertação. À Joana, apesar de durante a realização desta não estarmos muitas vezes juntos, gostava de lhe agradecer a entreajuda e todos os trabalhos realizados durante a parte lectiva.

À minha família, agradeço todo o apoio e incentivo que tornaram possível mais esta etapa da minha vida.

Por fim, um beijo muito especial para a Sílvia que me compreendeu durante estes longos meses. Apesar da minha reduzida disponibilidade e das férias que abdicamos, sempre me apoiou.

Índice

Índice de Figuras

Índice de Tabelas

Capítulo 1

Introdução

1.1 Motivações e Objectivos

O trabalho desenvolvido nesta dissertação insere-se na Análise Classificatória, uma área da Análise de Dados Multivariados.

A aplicação de um método de classificação a um conjunto de dados multivariados produz, sobre os elementos a classificar, um conjunto de classes organizado segundo uma estrutura (uma partição, uma hierarquia, uma cobertura, uma pirâmide, etc.). Esta estrutura depende, não só do conjunto de dados, mas também da natureza intrínseca do método de classificação usado (ver Gordon [21]). Neste trabalho duas dessas metodologias de Classificação serão estudadas: a Classificação Hierárquica e a Classificação Piramidal. Como será visto, a Classificação Piramidal produz estruturas mais complexas que a Classificação Hierárquica.

Os métodos de Classificação Hierárquica e de Classificação Piramidal impõem a escolha de duas funções de comparação: função de comparação entre elementos e função de comparação entre partes do conjunto de elementos a classificar. O resultado obtido da aplicação de um destes métodos classificatórios depende frequentemente desta dupla escolha, levantando naturalmente a questão da escolha indicada em cada caso. Da estrutura obtida por um método classificatório torna-se importante averiguar o grau de responsabilidade exclusiva do critério usado. Estas e outras questões têm sido abordadas em vários trabalhos no âmbito da *Validação em Classificação*. A geração aleatória de estruturas de classificação surgiu como uma ferramenta importante nessa área do conhecimento.

Vários métodos já foram propostos para a geração aleatória de dendrogramas, dos quais se salientam os trabalhos de Lapointe e Legendre [25], Sousa [37] e Podani [32]. A sua extensão a estruturas mais complexas parece relevante.

Assim, o objectivo deste trabalhalho é o de propor métodos de geração aleatória de pirâmides.

A complexidade das estruturas piramidais, relativamente às estruturas hierárquicas, leva à necessidade de introduzir um conjunto vasto de definições e propriedades, para uma melhor compreensão dos algoritmos de Classificação Piramidal Ascendente mais relevantes. Nesta área serão referidos, principalmente, os trabalhos desenvolvidos por Diday [13], Bertrand [4] e Mfoumoune [27].

Nesta dissertação propõe-se, implementa-se e discute-se um algoritmo – ***R****andom Generation* ***A****lgorithm of* ***P****yramids* (***RAP***) – que gera aleatoriamente pirâmides fixado um número qualquer de nós terminais.

No que respeita a aspectos computacionais foram desenvolvidos programas em linguagem Matlab para os diferentes métodos de geração aleatória de dendrogramas, para o método *RAP* de geração aleatória de uma pirâmide e alguns algoritmos para a identificação topológica das estruturas geradas. Por opção, uma listagem dos programas desenvolvidos encontra-se em anexo (Anexos 1 a 8).

Pareceu natural, por comparação com a geração de dendrogramas, verificar se o método *RAP* gerava pirâmides de forma uniforme no sentido de Furnas [19], o que levou à necessidade de identificar os diferentes tipos topológicos das pirâmides. Devido à complexidade das estruturas piramidais esta identificação foi apenas conseguida para um pequeno número de nós terminais. Concluiu-se que, de facto, o método não gera pirâmides uniformemente, embora não se afaste significativamente desta distribuição. Esta característica era já esperada atendendo a um conjunto de limitações que o método apresenta e que são referidas ao longo do trabalho.

1.2 Estrutura do Trabalho

Esta dissertação está organizada em sete capítulos.

Este Capítulo 1 é dedicado a uma pequena introdução, na qual se referem as motivações e objectivos do trabalho e a sua estrutura.

No Capítulo 2, intitulado "Classificação", são introduzidas algumas noções de base, importantes para a compreensão do trabalho aqui desenvolvido. Assim, apresentam-se duas metodologias de Classificação: Classificação Hierárquica e Classificação Piramidal. É ainda apresentado um exemplo para ilustrar as diferenças de actuação dos dois procedimentos.

O Capítulo 3 é dedicado à Geração Aleatória de Dendrogramas. São apresentadas algumas definições sobre árvores em geral e sobre árvores de classificação

(ou dendrogramas) em particular. São referidos e discutidos os métodos propostos na literatura para gerar aleatoriamente dendrogramas, com especial ênfase para os métodos que geram aleatória e uniformemente dendrogramas no sentido de Furnas [19]. Como já foi referido, o objectivo deste trabalho é a proposta de métodos de geração aleatória de pirâmides, no que poderá ser visto como uma extensão dos métodos de geração aleatória de dendrogramas estudados neste capítulo.

Assim, no Capítulo 4 apresentam-se os principais algoritmos, propostos ao longo das últimas décadas, de Classificação Piramidal Ascendente. Inicialmente introduz-se um conjunto vasto de noções necessárias à apresentação desses algoritmos. Na Secção 4.3 introduzem-se os algoritmos de Diday [12] e [13], Bertrand [4] e Mfoumoune [27]. O resto do capítulo é dedicado a alguns comentários sobre as propriedades e/ou limitações desses métodos.

No Capítulo 5 é proposto um método de geração aleatória de pirâmides, denominado ***R**andom Generation **A**lgorithm of **P**yramids* (***RAP***) cuja filosofia base é análoga ao algoritmo de Classificação Piramidal Ascendente proposto por Bertrand [4]. A apresentação do algoritmo proposto é complementada por um exemplo de aplicação. Conclui-se o capítulo tecendo algumas considerações a este método e referindo o possível desenvolvimento de um método alternativo com algumas vantagens relativamente ao aqui proposto.

A análise do desempenho do método de geração de pirâmides é o objectivo do Capítulo 6. Por analogia com o estudo dos métodos de geração aleatória de dendrogramas, a questão da geração uniforme de pirâmides parece pertinente. Neste sentido, torna-se necessário identificar os diferentes tipos topológicos de pirâmides, para um número fixo de nós terminais, bem como o respectivo número de pirâmides não isomórficas. Dada a complexidade do estudo, este objectivo só foi conseguido para 3 e 4 nós terminais. Um estudo de simulação permitiu analisar o desempenho do algoritmo *RAP* no que respeita à geração ser uniforme.

Conclui-se esta dissertação com um capítulo dedicado a algumas considerações finais e propostas de desenvolvimento de trabalho futuro.

Capítulo 2

Classificação

2.1 Introdução

A Análise Classificatória é um área da Análise de Dados Multivariados que tem como objectivo a formação de grupos de elementos que se assemelhem entre si. A aplicação de um método de classificação a um conjunto de dados multivariados produz, sobre os elementos a classificar, um conjunto de classes organizado segundo uma determinada estrutura. Este capítulo irá centrar-se no estudo e comparação de dois desses tipos de métodos de classificação: a Classificação Hierárquica e a Classificação Piramidal, originando as estruturas classificativas hierarquias e pirâmides, respectivamente.

Assim, começar-se-á por apresentar algumas definições e propriedades associadas à Classificação Hierárquica para, de seguida, generalizá-las à Classificação Piramidal. Será visto que as relações existentes entre classes na Classificação Piramidal são mais complexas. Por fim, a aplicação prática de um exemplo académico permitirá ilustrar os diferentes modos de actuação dos algoritmos de classificação.

2.2 Classificação Hierárquica

De um modo geral, a Classificação Hierárquica tem como objectivo produzir, sobre o cunjunto de elementos a classificar, partições ou hierarquias de partições.

Os métodos classificatórios actuam fundamentalmente sobre dois tipos de dados: um quadro de dados ou um quadro de proximidades. Em geral, admite-se que o quadro de dados é do tipo indivíduos-variáveis. Usualmente é representado por uma matriz $X = (x_{ik})$, de dimensões $n \times p$, onde x_{ik} representa o valor da variável de ordem k observada no indivíduo i. Numa primeira fase de um método classifi-

catório, em geral, calcula-se uma matriz de proximidades que pode ser calculada para os n indivíduos, originando uma matriz $n \times n$, onde o elemento (i, j) representa a proximidade entre os indivíduos i e j, ou para as p variáveis, dando então origem a uma matriz $p \times p$, em que o elemento (k, l) representa a proximidade entre as variáveis k e l. Estas matrizes de proximidades são simétricas e podem traduzir semelhanças ou dissemelhanças de acordo com o tipo de função de comparação que lhe dá origem.

Os diferentes métodos de Classificação Hierárquica podem ser agrupados em dois tipos: os ascendente ou aglomerativos, sem dúvida os mais adoptados, e os descendentes ou divisíveis. Seja E o conjunto dos elementos a classificar, um método aglomerativo parte das classes singulares correspondentes aos elementos singulares do conjunto E e, em cada passo do algoritmo, reúne as duas classes mais semelhantes. Este procedimento designa-se por Classificação Hierárquica Ascendente (C.H.A.). Um processo divisível consiste em dicotomias sucessivas do conjunto total E até à obtenção dos elementos isolados que o constituem. Este método designa-se por Classificação Hierárquica Descendente (C.H.D.) e é menos utilizado devido à sua complexidade algorítmica.

Para a aplicação de um método de Classificação é fundamental definir à priori dois tipos de funções: a função de comparação entre elementos e a função de comparação entre classes.

2.2.1 Funções de Comparação

Função de Comparação entre Elementos

Seja E o conjunto de elementos a classificar, que podem ser os indivíduos ou as variáveis. Por abuso de linguagem, considere-se o conjunto de elementos a classificar de cardinal n assim definido $E = \{e_1, \ldots, e_n\}$. De salientar que não se refere necessariamente aos indivíduos.

Define-se função de comparaçao entre elementos de E como uma aplicação $\gamma : E \times E \longrightarrow \mathbb{R}_0^+$. Esta função pode ser de dois tipos:

- **dissemelhança** - para dois elementos quaiquer e_i e e_j de E, um pequeno valor de $\gamma(e_i, e_j) = d_{ij}$ significa que e_i e e_j são elementos bastante análogos;

- **semelhança** - para dois elementos quaiquer e_i e e_j de E, um pequeno valor de $\gamma(e_i, e_j) = s_{ij}$ indica que e_i e e_j são elementos com bastantes diferenças.

A função de comparação entre elementos permite transformar o quadro de dados na matriz de proximidades. Se a matriz de proximidades é calculada sobre os indivíduos a função de comparação usada é, em geral, do tipo dissemelhança. Quando esta matriz é obtida para as variáveis a função de comparação utilizada é do tipo semelhança.

Seja $d : E \times E \longrightarrow \mathbb{R}_0^+$ uma aplicação e considerem-se as seguintes condições:

(i) $d(e_i, e_i) = 0 \quad , \quad \forall\, e_i \in E$

(ii) $d(e_i, e_j) = d(e_j, e_i) \quad , \quad \forall\, e_i, e_j \in E$

(iii) $d(e_i, e_j) = 0 \Longrightarrow e_i = e_j \quad , \quad \forall\, e_i, e_j \in E$

(iv) $d(e_i, e_k) \leq d(e_i, e_j) + d(e_j, e_k) \quad , \quad \forall\, e_i, e_j, e_k \in E$

Se d satisfaz (i) e (ii) diz-se um *índice de dissemelhança*.
Se d é um índice de dissemelhança e satisfaz (iii) diz-se um *índice de distância*.
Se d é um índice de distância e satisfaz (iv) diz-se uma *distância*.

Se d é uma distância e satisfaz a seguinte propriedade

$$d(e_i, e_k) \leq max\{d(e_i, e_j), d(e_j, e_k)\} \quad , \quad \forall\, e_i, e_j, e_k \in E \tag{2.1}$$

então a função d diz-se uma ***distância ultramétrica***.

Esta propriedade, chamada de propriedade ultramétrica, é mais restritiva que a desigualdade triangular e tem uma grande aplicabilidade em Classificação Hierárquica.

Obtém-se uma caracterização análoga usando funções do tipo semelhança:

(i) $s(e_i, e_i) = s_{max} \quad , \quad \forall\, e_i \in E$

(ii) $s(e_i, e_j) = s(e_j, e_i) \quad , \quad \forall\, e_i, e_j \in E$

(iii) $s(e_i, e_j) = s_{max} \Longrightarrow e_i = e_j \quad , \quad \forall\, e_i, e_j \in E$

(iv) $s_{max} + s(e_i, e_k) \geq s(e_i, e_j) + s(e_j, e_k) \ , \quad \forall\, e_i, e_j, e_k \in E$

Se s satisfaz (i) e (ii) diz-se um *índice de semelhança*.
Se s é um índice de semelhança e satisfaz (iii) diz-se um *índice de proximidade*.

Se s é um índice de proximidade e satisfaz (iv) diz-se uma *proximidade*.

Se s é uma proximidade e satisfaz a seguinte propriedade

$$s\left(e_i, e_k\right) \geq min\left\{ s\left(e_i, e_j\right), s\left(e_j, e_k\right) \right\}, \quad \forall\, e_i, e_j, e_k \in E \tag{2.2}$$

então a função s diz-se uma ***proximidade ultramétrica***.

A escolha da função de comparação entre elementos é um passo importante e dela pode depender bastante o resultado final. Algumas das medidas de dissemelhança e semelhança mais usadas são apresentadas de seguida. Para mais detalhes ver Bacelar-Nicolau [1], Gordon [21] e referências aí citadas.

Considere-se a matriz de dados iniciais $X = (x_{ik})$, de dimensões $n \times p$. Seja $w_i \in E$, com $i = 1, \ldots, n$, o indivíduo i observado nas p variáveis, $(x_{i1}, x_{i2}, \ldots, x_{ip})$. Assim, para comparar os pares de indivíduos, isto é, os pares de elementos de E, algumas das distâncias mais utilizadas são:

- distância Euclidiana

$$d\left(w_i, w_j\right) = \sqrt{\sum_{k=1}^{p} \left(x_{ik} - x_{jk}\right)^2} \tag{2.3}$$

- distância de Minkowsky

$$d\left(w_i, w_j\right) = \left(\sum_{k=1}^{p} \left| x_{ik} - x_{jk} \right|^{\rho} \right)^{\frac{1}{\rho}} \tag{2.4}$$

  Se $\rho = 1$ obtém-se a distância *city block*;
  Se $\rho = 2$ obtém-se a distância Euclidiana, já referida;
  Se $\rho \longrightarrow \infty$ obtém-se a distância do máximo.

Quando se pretende comparar variáveis deve-se usar semelhanças. Na matriz de dados iniciais considere-se v_k, com $k = 1, \ldots, p$, a k-ésima variável observada nos n indivíduos, $(x_{1k}, x_{2k}, \ldots, x_{nk})$. Duas medidas de correlação são apresentadas em baixo. Ambas medem o cosseno do ângulo entre dois vectores, com os vectores a serem medidos, respectivamente, a partir da origem e a partir da média dos dados.

- separação angular

$$s\left(v_k, v_l\right) = \frac{\sum_{j=1}^{n} x_{jk} x_{jl}}{\sqrt{\left(\sum_{j=1}^{n} x_{jk}^2\right)\left(\sum_{j=1}^{n} x_{jl}^2\right)}} \tag{2.5}$$

- coeficiente de correlação de Pearson

$$s\left(v_k, v_l\right) = \frac{\sum_{j=1}^{n}\left(x_{jk} - \overline{v_k}\right)\left(x_{jl} - \overline{v_l}\right)}{\sqrt{\left(\sum_{j=1}^{n}\left(x_{jk} - \overline{v_k}\right)^2\right)\left(\sum_{j=1}^{n}\left(x_{jl} - \overline{v_l}\right)^2\right)}} \tag{2.6}$$

em que $\overline{v_k}$ e $\overline{v_l}$ são, respectivamente, os valores das médias observadas para as variáveis v_k e v_l.

A escolha do tipo de função de comparação depende essencialmente da natureza das variáveis envolvidas e do tipo de elementos a classificar. Desta escolha depende bastante o resultado final.

Função de Comparação entre Classes

Seja E o conjunto de elementos a classificar de cardinal n, que como já foi referido não são necessariamente os indivíduos.

Uma função de comparação γ entre os elementos de E dá origem a uma matriz de semelhanças ou dissemelhanças, a já referida matriz de proximidades, com $\binom{n}{2} = \frac{n(n-1)}{2}$ elementos relevantes[1]. O objectivo da Classificação é formar classes de elementos o mais semelhantes entre si. É, por isso, natural que se comece por agregar o par de elementos de E onde a função de comparação é mínima (máxima), quando esta é do tipo dissemelhança (semelhança). Precisa-se de introduzir uma função que permita comparar entre si classes de elementos de E, podendo uma classe ser constituída por um único elemento.

Seja $\mathscr{P}(E)$ o conjunto das partes de E e considere-se a função de comparação entre as classes dada por

$$\Gamma : \mathscr{P}(E) \times \mathscr{P}(E) \longrightarrow \mathbb{R}_0^+.$$

Também para a comparação entre as classes existe um número vasto de opções, pelo que neste trabalho se fará referência apenas a algumas dessas funções. Sejam A e B elementos de $\mathscr{P}(E)$. Para γ do tipo dissemelhança, são referidas algumas das funções mais conhecidas:

- Ligação Mínima (S.L. - Single Linkage)

$$\Gamma(A, B) = min\left\{\gamma(a, b) : a \in A, b \in B\right\} \tag{2.7}$$

[1] A matriz de proximidades é quadrada, simétrica e a informação dos elementos da diagonal desprezível.

Este método agrega classes pelos elementos mais próximos. As classes exibem um fenómeno conhecido por cadeia, isto é, à medida que as classes são agregadas há tendência para a formação de classes numerosas e alongadas. O S.L. priveligia o isolamento das classes.

- Ligação Máxima (C.L. - Complete Linkage)

$$\Gamma(A,B) = max\{\gamma(a,b) : a \in A, b \in B\} \tag{2.8}$$

Neste critério a distância entre duas classes é dada pela distância dos dois elementos mais afastados. As classes formadas por esta metedologia são mais compactas e coesas internamente.

- Ligação Média (A.L. - Average Linkage)

$$\Gamma(A,B) = \sum_{a\in A, b\in B} \frac{\gamma(a,b)}{n_A n_B}, \tag{2.9}$$

onde n_A e n_B são os cardinais das classes A e B, respectivamente.

Este método tem um desempenho intermédio entre o S.L. e o C.L. quanto ao tipo de classes formadas. A distância entre duas classes é a distância média para todos os pares $(a,b) \in A \times B$.

- Método de Ward (M.W.)

Este método é aplicado a casos em que a função γ é a distância euclidiana. Os n elementos de E são considerados como uma nuvem de pontos num espaço $\mathbb{R}^p$ e a agregação faz-se no sentido de minimizar a variação da inércia intraclasse. Assim, em cada etapa reunem-se as classes que conduzem a uma perda de inércia interclasses mínima, sendo esta dada por

$$\begin{aligned} \Gamma(A,B) &= n_A d^2(g_A,g) + n_B d^2(g,g_B) - (n_A+n_B)\, d^2(g_{A\cup B},g) \\ &= \frac{n_A n_B}{n_A+n_B} d^2(g_A,g_B) \end{aligned} \tag{2.10}$$

onde g, g_A, g_B e $g_{A\cup B}$ são os centros de gravidade global e das classes A, B e $A \cup B$, respectivamente, e n_A e n_B os cardinais das classes A e B, respectivamente.

Os algoritmos dos critérios de agregação referidos, assim como outros, para coeficientes de dissemelhança, podem ser descritos por uma fórmula de recorrência,

proposta por Lance & Williams [24] e generalizada por Jambu & Lebeaux [23]. Estas fórmulas permitem o cálculo da dissemelhança entre uma classe e uma outra formada pela reunião de duas classes.

2.2.2 Hierarquia: definições e representações

Como já foi referido, uma Classificação Hierárquica tem como objectivo produzir partições e hierarquias de partições sobre os elementos a classificar.

Nas definições que se seguem continua-se a considerar que E é o conjunto dos elementos a classificar e que este tem cardinal n.

Definição 2.1 Uma **Partição** de E é um conjunto de partes não vazias de E, mutuamente disjuntas $(E_i \cap E_j = \emptyset, \forall i, j = 1, 2, \ldots, k,\ i \neq j)$, cuja reunião é E $\left(\bigcup_{i=1}^{k} E_i = E\right)$.
Vamos designar o conjunto das partições de E por $Part\,(E)$.

Definição 2.2 Seja H uma família de partes não vazias de E. Diz-se que H é uma **Hierarquia** sobre E se:

(i) $E \in H$

(ii) $\{a\} \in H, \quad \forall\, a \in E$

(iii) $\forall\, h, h' \in H, \quad h \cap h' \in \{\emptyset, h, h'\}$

Uma herarquia é um conjunto de partições encaixadas.

Definição 2.3 (Sucessor e Predecessor) Seja H uma hierarquia. Diz-se que $h \in H$ é um **sucessor** de $h' \in H$ se:

- $h \subseteq h'$
- $\nexists\, h'' \in H : \quad h \subseteq h'' \subseteq h'$

Diz-se que h' é um **predecessor** de h.

Na Figura 2.1 definida na página 13, h_3 é um predecessor de h_1 e um sucessor de h_4.

Proposição 2.1 *Seja H uma hierarquia. Cada classe $h \in H$ tem, no máximo, um predecessor.*

Demonstração: A justificação deste resultado é trivial. Supondo que $h \in H$ admite dois predecessores, h_1 e h_2, é fácil notar que $h_1 \cap h_2 = h \notin \{\emptyset, h_1, h_2\}$, isto é, não é satisfeita a condição (iii) da Definição 2.2 de hierarquia.

A cada classe da hierarquia pode associar-se um número real positivo que indica o grau de agregação dos elementos dessa classe, dando origem a uma hierarquia indiciada. Considerando que a função de comparação entre os elementos de E é do tipo dissemelhança, apresenta-se a seguinte definição.

Definição 2.4 Uma **hierarqia indiciada** em E é um par (H, f) onde H é uma hierarquia e $f : H \longrightarrow \mathbb{R}_0^+$ é tal que:

(i) $f(h) = 0 \iff \underbrace{\exists\, a \in E : h = \{a\}}_{h \text{ contém um único elemento}}$

(ii) $\forall\, h, h' \in H, \quad h \subset h' \Longrightarrow f(h) \leq f(h')$.

Os valores de f são designados por *índices de nível de agregação* e os valores ordinais correspondentes por *níveis de agregação*. Quando estes são usados a hierarquia indiciada dará origem a uma hierarquia habitualmente designada por hierarquia estratificada ou de níveis.

Definição 2.5 Uma **hierarquia estratificada** ou **de níveis** é uma hierarquia indiciada (H, f) onde f é uma aplicação da hierarquia H num intervalo de inteiros $[0..m]$. O inteiro $m = f(E)$ é chamado o nível de agregação máximo.

Se a aplicação f é injectiva então o nível de agregação m da hierarqiua estratificada será igual a $n - 1$, isto é, em todos os níveis da hierarquia há a junção de apenas duas classes.

Seja (H, f) uma hierarquia indiciada. Um *indice de dissemelhança induzido* por H, d_H, é uma aplicação de $E \times E$ em $\mathbb{R}_0^+$ definida por:

$$d_H(x, y) = min\{f(h) :\ h \in H,\ x, y \in h\}. \tag{2.11}$$

d_H é dado pelo valor da função de comparação entre duas classes, cuja reunião contém pela primeira vez os elementos x e y. d_H é uma distância ultramétrica sobre E. Uma Classificação Hierárquica produz uma hierarquia indiciada cuja

representação gráfica mais comum é um dendrograma. A altura do traço horizontal dos vértices do dendrograma, que representam as classes, é proporcional ao valor positivo $f(h)$. No exemplo que se segue são utilizados os níveis de agregação. Na Figura 2.1 podem observar-se as partições para os vários níveis de agregação e o respectivo dendrograma.

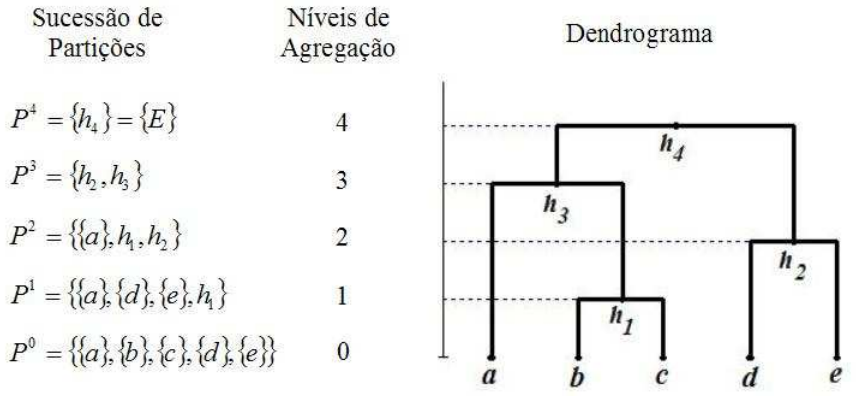


Figura 2.1: Representação de uma hierarquia.

A hierarquia obtida para este exemplo é

$$H = \bigcup_{i=0}^{4} P^i = \{\{a\}, \{b\}, \{c\}, \{d\}, \{e\}, h_1, h_2, h_3, E\}. \tag{2.12}$$

com $h_1 = \{b, c\}$, $h_2 = \{d, e\}$ e $h_3 = \{a, b, c\}$.

Dado $E = \{e_1, \ldots, e_n\}$ de cardinal n, a representação de uma hierarquia pode ainda ser feita através de uma matriz, designada por *matriz ultramétrica*. Notando por $U = (u_{i,j})_{i,j \in \{1,\ldots,n\}}$ essa matriz, o elemento $u_{i,j}$ é o valor $d_H(e_i, e_j)$, com $e_i, e_j \in E$, ou então o valor ordinal correspondente. U é uma matriz quadrada, de dimensão n, simétrica e são suficientes $\frac{n(n-1)}{2}$ valores para a definir. Apresenta-se a seguir a matriz ultramétrica associada ao dendrograma da Figura 2.1.

$$\begin{array}{c} \\ a \\ b \\ c \\ d \\ e \end{array} \begin{array}{c} \begin{array}{ccccc} a & b & c & d & e \end{array} \\ \begin{pmatrix} 0 & 3 & 3 & 4 & 4 \\ & 0 & 1 & 4 & 4 \\ & & 0 & 4 & 4 \\ & & & 0 & 2 \\ & & & & 0 \end{pmatrix} \end{array} \tag{2.13}$$

Na obtenção de uma hierarquia é produzida uma ordem sobre os elementos de E, designada por ordem induzida. A definição de ordem será relembrada mais à frente, na Subsecção 4.2.2, mas poderá ser entendida como a posição física que os elementos do conjunto E ocupam, como nós terminais, no dendrograma. Na Figura 2.1 a ordem sobre os elementos é (a, b, c, d, e). A ordem induzida sobre os elementos de E pela hierarquia não é única. É possível permutar os elementos de todas as classes da hierarquia e obter uma ordem diferente. Pode-se, por exemplo, verificar que se tem a mesma hierarquia com a ordem (a, c, b, d, e), que consiste numa inversão dos elementos da classe h_1, ou ainda com a ordem (e, d, c, b, a), que consiste em várias inversões. Bertrand [4] mostra que, numa hierarquia, existem 2^{n-1} ordens possíveis sendo n o cardinal de E.

Teorema 2.1 (Johnson - Benzécri) *Existe uma bijecção entre o conjunto das hierarquias indiciadas e o conjunto das distâncias ultramétricas.*

A demonstração deste resultado consiste em verificar que a cada hierarquia indiciada (H, f) está associado um índice de dissemelhança induzido d_H, que é uma distância ultramétrica, e que esta aplicação é bijectiva. Para uma demonstração ver, por exemplo, Diday et al. (1982) [14].

2.2.3 Classificação Hierárquica Ascendente (C.H.A.)

Existem alguns algoritmos que permitem construir uma hierarquia indiciada a partir de uma função γ de comparação entre os elementos definida à priori sobre E. O mais utilizado é o da Classificação Hierárquica Ascendente (algoritmo C.H.A.). Definida a função Γ de comparação entre classes de $\mathscr{P}(E)$, ou seja, o critério de agregação, o algoritmo de C.H.A. pode ser enunciado da seguinte forma:

Algoritmo de C.H.A.

Uma condição prévia à aplicação de um algoritmo de C.H.A. é a definição das duas funções de comparação, γ e Γ, e consequente determinação do conjunto $\{\gamma(x, y) : x, y \in E\}$.

Passo 1 (Inicialização)
Definir a partição inicial constituída pelas classes singulares de E.

Passo 2 (Agregação)
Determinar o par de classes mais próximas que verificam o critério de agregação associado a Γ, reunir essas classes numa só e actualizar os valores de Γ entre esta nova classe e todas as restantes.

Passo 3 (Critério de Paragem)

Se a classe formada não é igual a E repete-se o **Passo 2**, caso contrário o algoritmo termina.

A agregação de um número de classes superior a dois num dado nível tem fácil resolução. Porém, para simplificar a escrita, considera-se que em cada passo apenas um par de classes se agrega. Num algoritmo de C.H.A. podem ser introduzidos critérios de paragem que não impliquem a actuação do algoritmo até à formação da classe E. Por exemplo, pode-se impôr um número mínimo fixo de classes para a última partição ou limitar o valor da função de comparação entre as classes.

2.3 Classificação Piramidal

A Classificação Piramidal, introduzida por Diday (1984) [12] e Bertrand (1986) [4], é uma generalização da Classificação Hierárquica. Tal como a hierarquia, uma pirâmide é uma colecção de classes de um conjunto de elementos E a classificar. No entanto, a Classificação Piramidal dá normalmente origem a relações mais complexas entre as classes. Em particular e contrariamente ao modelo hierárquico, duas classes cuja intersecção não é vazia, não são necessariamente sobrepostas, isto é, uma não tem necessariamente que conter a outra. Para além disso, outra propriedade interessante das representações piramidais é a capacidade de produzir um número pequeno de ordens, sobre o conjunto dos elementos a classificar, respeitando as condições de proximidade entre esses elementos.

2.3.1 Pirâmide: definições, propriedades e representações

Considere-se novamente, nas definições que se seguem, o conjunto de elementos a classificar E, de cardinal n.

Definição 2.6 Diz-se que p é uma **Parte Conexa** de E de acordo com a ordem θ se p é um intervalo dessa ordem.
Mais precisamente,

$$\begin{aligned} \{a \in p\} &\iff \{a \text{ está entre o menor e o maior elemento de } p \text{ de acordo com } \theta\} \\ &\iff \{p \text{ é conexo}\} \end{aligned}$$

Diz-se ainda que a ordem θ é compatível com P se todo $p \in P$ é conexo de acordo com θ.

Definição 2.7 Uma **Cobertura** do conjunto E é um conjunto de partes não vazias de E cuja reunião é E.
Vamos designar o conjunto das coberturas de E por $Cob\,(E)$.

Definição 2.8 Seja P uma família de partes não vazias de E. Diz-se que P é uma **Pirâmide** sobre E se:

(i) $E \in P$

(ii) $\{a\} \in P, \quad \forall\, a \in E$

(iii) $\forall\, p, p' \in P, \quad p \cap p' = \emptyset \quad$ ou $\quad p \cap p' \in P$

(iv) existe uma ordem θ compatível com P tal que $\forall p \in P$, p é um intervalo dessa ordem

Uma pirâmide é um sucessão de coberturas encaixadas.

Considere-se $E = \{a, b, c\}$ e $P = \left\{ \{a\}, \{b\}, \{c\}, \{a, b\}, \{b, c\}, E \right\}$ uma pirâmide. Verifica-se facilmente que $P' = P \cup \{a, c\}$ não é uma pirâmide uma vez que não é satisfeita a quarta condição da definição, isto é, não é possível construir uma ordem θ compatível com P'.

Notando que uma partição é um caso particular de uma cobertura é quase imediata a propriedade que se segue.

Proposição 2.2 *O conjunto das hierarquias está incluído no conjunto das pirâmides.*

Demonstração: Pretende verificar-se que toda a hierarquia H é ainda uma pirâmide, isto é, dada uma qualquer hierarquia H então ela sastifaz as quatro condições da Definição 2.8. (i) e (ii) coincidem nas Definições 2.2 e 2.8. Se H é uma hierarquia então $\forall\, h, h' \in H,\ h \cap h' \in \{\emptyset, h, h'\}$ o que implica que $h \cap h' = \emptyset$ ou $h \cap h' \in H$. Portanto, a terceira condição da Definição 2.8 é satisfeita. Verifica-se que existe também uma ordem θ sobre E compatível com H (ver, por exemplo, Diday [13]). Portanto, H é uma pirâmide.

Definição 2.9 (Sucessor e Predecessor) Seja P uma pirâmide. Diz-se que $p \in P$ é um **sucessor** de $p' \in P$ se:

- $p \subseteq p'$
- $\nexists\, p'' \in P : \quad p \subseteq p'' \subseteq p'$

Diz-se que p' é um **predecessor** de p.

As noções de sucessor e predecessor são equivalentes às dadas em hierarquias. Na Figura 2.2, que será definida nesta subsecção, p_4 e p_5 são predecessores de p_3.

Uma propriedade importante e que diferencia as pirâmides das hierarquias é a que se enuncia a seguir.

Proposição 2.3 *Seja P uma pirâmide. Cada classe $p \in P$ tem, no máximo, dois predecessores.*

Demonstração: Vai-se provar por redução ao absurdo. Suponha-se que $p \in P$ admite, pelo menos, três predecessores distintos, p_1, p_2 e p_3.

Como as classes p_i $(i = 1, 2, 3)$ são predecessores da mesma classe p, tem-se que

$$p = p_1 \cap p_2 = p_1 \cap p_3 = p_2 \cap p_3. \tag{2.14}$$

Seja θ a ordem compatível com P. As classes de P são intervalos relativamente à ordem θ. Então as classes p_i podem ser escritas sob a forma

$$p_i = [a_i, b_i] \quad (i = 1, 2, 3)$$

onde a_i e b_i são elementos de E.

Sem perda de generalidade, podemos supor que os elementos a_i estão "ordenados" da seguinte maneira

$$a_1 \leq a_2 \leq a_3 \ .$$

A segunda igualdade de (2.14) pode ser escrita da forma

$$[a_2, min\,(b_1, b_2)] = [a_3, min\,(b_1, b_3)]\,.$$

Consequentemente $a_2 = a_3$, pelo que existe uma relação de inclusão entre p_2 e p_3, o que é contraditório. Logo, a classe p admite, no máximo, dois predecessores.

Em Classificação Hierárquica dois conceitos importantes são o de *hierarquia* e *hierarquia indiciada*. Da mesma forma, em Classificação Piramidal as definições de *pirâmide* e *pirâmide indiciada* são fundamentais. Assim, a cada classe da pirâmide será associado um valor que mede o grau de agregação dos elementos nesse patamar.

Definição 2.10 Uma **pirâmide indiciada** sobre E é um par (P, f) onde P é uma pirâmide e $f : P \longrightarrow \mathbb{R}_0^+$ é tal que:

(i) $f\,(p) = 0 \iff \underbrace{\exists\, a \in E : p = \{a\}}_{p \text{ contém um único elemento}}$

(ii) $\forall\, p, p' \in P, \quad p \subset p' \Longrightarrow f\,(p) \leq f\,(p')$

Uma pirâmide é indiciada em ***sentido lato*** se:

$$\underbrace{p \subset p'}_{estritamente} \quad \text{e} \quad f(p) = f(p') \quad \Longrightarrow \quad p \text{ tem dois predecessores}^2$$

Uma pirâmide é dita indiciada em ***sentido estrito*** se:

$$\underbrace{p \subset p'}_{estritamente} \quad \Longrightarrow \quad f(p) < f(p')$$

Analogamente ao que foi dito para hierarquias, os valores de f serão designados *índices de nível de agregação* e os valores ordinais correspondentes por *níveis de agregação*. Assim, uma **pirâmide estratificada** ou **de níveis** é uma pirâmide indiciada (P, f) onde f é uma aplicação da pirâmide P num intervalo de inteiros $[0..m]$.

A Figura 2.2 ilustra o resultado de uma classificação piramidal sobre o conjunto de elementos $E = \{a, b, c, d, e\}$. É apresentada a sucessão de coberturas para os vários níveis de agregação e a representação piramidal.

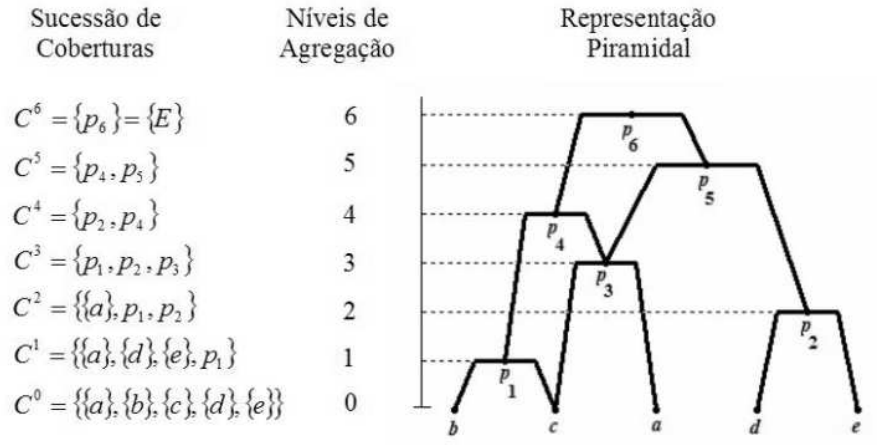


Figura 2.2: Representação de uma pirâmide.

A pirâmide obtida para este exemplo é:

$$P = \bigcup_{i=0}^{6} C^i = \{\{a\}, \{b\}, \{c\}, \{d\}, \{e\}, p_1, p_2, p_3, p_4, p_5, E\}. \tag{2.15}$$

onde $p_1 = \{b, c\}$, $p_2 = \{d, e\}$, $p_3 = \{a, c\}$, $p_4 = \{a, b, c\}$ e $p_5 = \{a, c, d, e\}$.

As ordens compatíveis com a classificação piramidal obtida são:

[2] $\exists\, p_1, p_2 \in P, \quad p_1 \neq p \quad \text{e} \quad p_2 \neq p : \quad p = p_1 \cap p_2$.

- $\theta_1 = (b, c, a, d, e)$, a representada na figura;
- $\theta_2 = (b, c, a, e, d)$, que resulta da inversão da classe p_2;
- $\theta_3 = (e, d, a, c, b)$, que resulta da inversão da classe p_6;
- $\theta_4 = (d, e, a, c, b)$, que resulta da inversão da classe p_2 e p_6.

Definição 2.11 (Matriz de Robinson) Uma matriz é de Robinson se e só se os termos das linhas e das colunas **nunca diminuem** quando nos afastamos da diagonal principal, em qualquer das direcções.
A matriz de Robinson será denotada por $R = (r_{i,j})_{i,j\in\{1,\ldots,n\}}$.

R é uma matriz quadrada, de dimensão n, simétrica e são suficientes $\frac{n(n-1)}{2}$ valores para a definir. Podem considerar-se apenas os elementos da matriz triangular superior, acima da diagonal principal, pois contêm toda a informação. Numa matriz de Robinson, para todo $i < j$, tem-se que $(r_{i,j}) \leq (r_{i,j+1})$ e $(r_{i,j}) \leq (r_{i-1,j})$. De seguida apresenta-se a matriz de Robinson associada à representação piramidal da Figura 2.2.

$$\begin{array}{c} \\ b \\ c \\ a \\ d \\ e \end{array} \begin{array}{c} \begin{array}{ccccc} b & c & a & d & e \end{array} \\ \begin{pmatrix} 0 & 1 & 4 & 6 & 6 \\ & 0 & 3 & 5 & 5 \\ & & 0 & 5 & 5 \\ & & & 0 & 2 \\ & & & & 0 \end{pmatrix} \end{array} \tag{2.16}$$

Considere-se (P, f) uma hierarquia indiciada. Um *índice de dissemelhança induzido* por P, d_P, é uma aplicação de $E \times E$ em $\mathbb{R}_0^+$ definida por:

$$d_P(x, y) = min\{f(p) : p \in P,\ x, y \in p\}. \tag{2.17}$$

d_P é um *índice piramidal* sobre E que se define, formalmente, a seguir.

Definição 2.12 (Índice Piramidal) Um índice de dissemelhança d é dito piramidal se:

(i) $d(x, y) = 0 \iff x = y$

(ii) existir uma ordem θ tal que:

$$\forall\, x, y, z \in E,\ x <_\theta y <_\theta z \implies d(x, z) \geq max\{d(x, y), d(y, z)\} \tag{2.18}$$

Proposição 2.4 *O conjunto das distâncias ultramétricas está incluído no conjunto dos índices piramidais.*

Demonstração: Seja d uma qualquer distância ultramétrica. A condição (i) da Definição 2.12 é trivialmente satisfeita. Por outro lado, a d pode-se associar uma ordem θ tal que $M(d,\theta)$[3] é uma matriz ultramétrica e, por conseguinte, uma matriz de Robinson, o que verifica (ii).

Proposição 2.5 *Se d é um índice piramidal então as condições seguintes são equivalentes:*

(i) *θ é compatível com d*

(ii) *$M(d,\theta)$ é uma matriz de Robinson*

(iii) *para todo o par de elementos x e y incluídos (em sentido lato), de acordo com a ordem θ, entre dois elementos a e b verifica-se que:*

$$d(a,b) \geq d(x,y)$$

Teorema 2.2 (extensão do Teorema de Johnson - Benzécri) *Existe uma bijecção entre o conjunto das pirâmides indiciadas (em sentido lato)* Π *e o conjunto dos índices piramidais D.*

É dada uma demonstração do teorema em Diday [13], que consiste em verificar que existe uma aplicação $\Phi : \Pi \longrightarrow D$ e uma aplicação $\Psi : D \longrightarrow \Pi$ tal que Φ e Ψ são a inversa uma da outra.

2.3.2 Classificação Piramidal Ascendente (C.P.A.)

Tal como no método de C.H.A., na aplicação de um método de C.P.A. a um conjunto E, de cardinal n, estão implícitas duas escolhas: a função de comparação entre os elementos de E , γ, e a função de comparação entre as classes, Γ. A função de comparação entre elementos produz uma matriz de proximidades que permite relacionar entre si, em termos de semelhança, qualquer par de elementos de E, enquanto que a segunda permite relacionar pares de partes de E. No método de C.P.A. as funções de comparação entre as classes sofrem algumas modificações em relação ao método de C.H.A., uma vez que em cada nível tem-se uma cobertura.

[3]Considera-se que $M(d,\theta)$ é a matriz induzida pelo índice d associada à ordem θ.

A filosofia de construção de uma C.P.A é, em tudo, análoga à de uma C.H.A.. O princípio fundamental da C.P.A. pode ser formulado da seguinte forma:

"São agregadas sucessivamente as classes mais próximas, de acordo com as funções de comparação definidas à priori, entre as classes que ainda não foram agregadas duas vezes e que não estão contidas estritamente em alguma classe já formada."

Assim definido, este princípio representa um algorimo informal para a construção de uma pirâmide. Serão vistas no Capítulo 4 as dificuldades inerentes a este princípio do algoritmo, nomeadamente a actualização dos pares de classes agregáveis e a construção de uma ordem sobre os elementos compatível com a pirâmide. Nesse capítulo serão também apresentados alguns algoritmos de C.P.A. propostos.

Existem na literatura várias funções para definir índices de agregação Γ. Bertrand [4] propôs as seguintes generalizações das funções de comparação entre as classes utilizadas na C.H.A..

Sejam $A, B \in \mathscr{P}(E)$. Assim,

- Ligação Mínima (S.L. - Single Linkage)

$$\Gamma(A,B) = min\{\gamma(a,b) : a \in A - B, b \in B - A\} \tag{2.19}$$

- Ligação Máxima (C.L. - Complete Linkage)

$$\Gamma(A,B) = max\{\gamma(a,b) : a \in A, b \in B\} \tag{2.20}$$

- Ligação Média (A.L. - Average Linkage)

$$\Gamma(A,B) = \sum_{a \in A, b \in B} \frac{\gamma(a,b)}{n_A n_B - n_{A \cap B}}, \tag{2.21}$$

onde n_A, n_B e $n_{A \cap B}$ são os cardinais das classes A, B e $A \cap B$, respectivamente.

- Método de Ward (M.W.) Utiliza-se a relação seguinte sobre as inércias:

$$I_g(A \cup B) = I_g(A) + I_g(B) + \frac{n_A n_B}{n_A + n_B} d^2(g_A, g_B) \tag{2.22}$$

O índice de agregação Γ representa o aumento da inércia resultante da reunião de A e B, que é igual a:

– Se $A \cap B = \emptyset$, então

$$\Gamma(A, B) = \frac{n_A n_B}{n_A + n_B} d^2(g_A, g_B) \tag{2.23}$$

– Se $A \cap B \neq \emptyset$, então

$$\begin{aligned}\Gamma(A, B) = {} & n_A d^2(g_A, g) + n_B d^2(g, g_B) - \\ & - n_{A \cup B - A \cap B} d^2(g_{A \cup B - A \cap B}, g) - n_{A \cap B} d^2(g_{A \cap B}, g)\end{aligned} \tag{2.24}$$

2.4 Exemplo Ilustrativo

Nesta secção serão aplicados algoritmos de C.H.A. e C.P.A. a um mesmo conjunto de dados com o objectivo de ilustrar os seus modos de actuação e, portanto, ajudar a uma melhor compreensão.

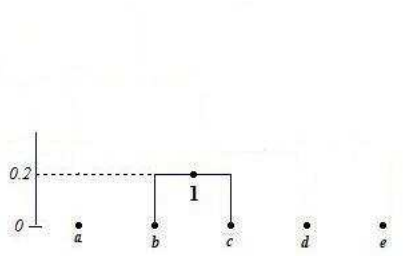
Seja $E = \{a, b, c, d, e\}$ o conjunto a classificar. Considere-se a *matriz de proximidades* (2.25) obtida após a definição de um índice de dissemelhança γ, isto é, a matriz que nos permite visualizar a distância entre todos os pares de elementos de E $\left(\gamma(i, j),\ \forall i, j \in E \text{ com } i \neq j\right)$.

$$\begin{array}{c} \\ a \\ b \\ c \\ d \\ e \end{array} \begin{array}{c} \begin{array}{ccccc} a & b & c & d & e \end{array} \\ \begin{pmatrix} 0 & 0.6 & 0.4 & 0.8 & 0.8 \\ & 0 & \boxed{0.2} & 1 & 1 \\ & & 0 & 0.7 & 0.7 \\ & & & 0 & 0.3 \\ & & & & 0 \end{pmatrix} \end{array} \tag{2.25}$$

Usando o critério da Ligação Máxima (Complete Linkage) tanto na C.H.A. como na C.P.A., definidos respectivamente por (2.8) e (2.20), vai-se obter o dendrograma e pirâmide associados.

2.4.1 Algoritmo de C.H.A

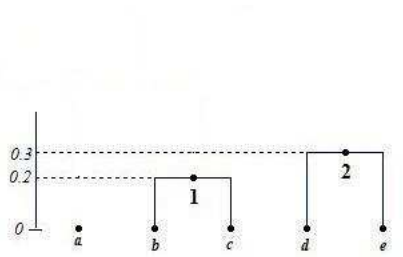
Iteração 1 O menor valor de γ na matriz de dissemelhanças é $\gamma(b,c) = 0.2$. Uma nova classe chamada $\mathbf{1} = \{b\} \cup \{c\} = \{b,c\}$ é criada. Na matriz actulizada, são retiradas as classes $\{b\}$ e $\{c\}$ e é acrescentada esta nova classe. A actualização dos valores na matriz é feita usando o critério referido, ou seja, $\Gamma(A,B) = max\{\gamma(i,j) : i \in A, j \in B\}$.
Assim, $\Gamma(\{a\},\mathbf{1}) = max\{\gamma(a,b),\gamma(a,c)\} = 0.6$. Analogamente, $\Gamma(\mathbf{1},\{d\}) = 1$ e $\Gamma(\mathbf{1},\{e\}) = 1$. Na Figura 2.3 pode visulizar-se a construção do respectivo dendrograma, bem como a matriz actualizada.



$$\begin{array}{c} \\ a \\ \mathbf{1} \\ d \\ e \end{array} \begin{array}{c} \begin{array}{cccc} a & \mathbf{1} & d & e \end{array} \\ \begin{pmatrix} 0 & 0.6 & 0.8 & 0.8 \\ & 0 & 1 & 1 \\ & & 0 & \boxed{0.3} \\ & & & 0 \end{pmatrix} \end{array}$$

Figura 2.3: Dendrograma e matriz actualizada (formação da classe $\mathbf{1}$).

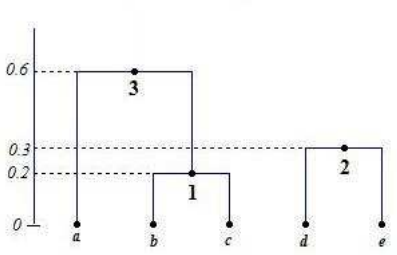
Iteração 2 O menor valor de dissemelhança na matriz actualizada é $\gamma(d,e) = 0.3$. São portanto agregados os ojectos d e e formando uma classe nova: a classe $\mathbf{2} = \{d,e\}$. São agora retiradas da matriz as classes $\{d\}$ e $\{e\}$ e acrescentada a nova classe $\mathbf{2}$. Tem-se que $\Gamma(\{a\},\mathbf{2}) = 0.8$ e
$\Gamma(\mathbf{1},\mathbf{2}) = max\{\gamma(b,d),\gamma(b,e),\gamma(c,d),\gamma(c,e)\} = 1$, usando a informação inicial, ou $\Gamma(\mathbf{1},\mathbf{2}) = max\{\gamma(\mathbf{1},d),\gamma(\mathbf{1},e) = 1$, usando a informação da matriz actualizada. A actualização do dendrograma e da matriz vem na Figura 2.4.



$$\begin{array}{c} \\ a \\ \mathbf{1} \\ \mathbf{2} \end{array} \begin{array}{c} \begin{array}{ccc} a & \mathbf{1} & \mathbf{2} \end{array} \\ \begin{pmatrix} 0 & \boxed{0.6} & 0.8 \\ & 0 & 1 \\ & & 0 \end{pmatrix} \end{array}$$

Figura 2.4: Dendrograma e matriz actualizada (formação da classe $\mathbf{2}$).

Iteração 3 O menor valor de dissemelhança na matriz actualizada é $\Gamma(\{a\}, \mathbf{1}) = 0.6$. As classes $\{a\}$ e **1** são agregadas formando a classe $\mathbf{3} = \{a, b, c\}$. Restam agora duas classes para agregar: a classe **2** e a classe **3**. Podemos visualizar na Figura 2.5 o dendrograma e a matriz que indica apenas a distância entre estas duas classes, de acordo com o critério adoptado.



$$\begin{array}{c} \\ \mathbf{2} \\ \mathbf{3} \end{array} \begin{array}{c} \begin{array}{cc} \mathbf{2} & \mathbf{3} \end{array} \\ \begin{pmatrix} 0 & \boxed{1} \\ & 0 \end{pmatrix} \end{array}$$

Figura 2.5: Dendrograma e matriz actualizada (formação da classe **3**).

Iteração 4 Por fim, tem-se $\Gamma(\mathbf{2}, \mathbf{3}) = 1$, isto é, são agregadas as classes **2** e **3** formando a classe $\mathbf{4} = \{a, b, c, d, e\} = E$. Na Figura 2.6 tem-se o dendrograma final.

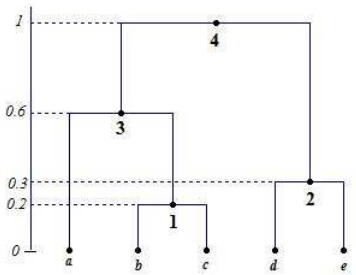
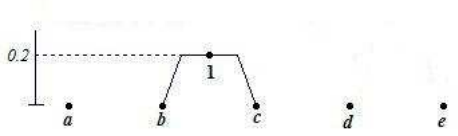


Figura 2.6: Dendrograma final (formação da classe **4**).

2.4.2 Algoritmo de C.P.A

Na aplicação de uma C.P.A. à mesma matriz de disemelhança (2.25) e usando o mesmo critério de comparação entre classes – Ligação Máxima (2.20) – obter-se-á a representação piramidal do conjunto E. Fixa-se, à partida, uma ordem compatível inicial, por exemplo $\theta = (a, b, c, d, e)$.

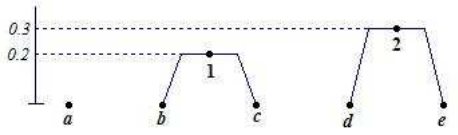
Iteração 1 O menor valor da matriz é $\gamma(b,c) = 0.2$. Uma nova classe é formada, $\mathbf{1} = \{b,c\}$, e acrescentada à matriz de dissemelhança, usando o critério adoptado. Na matriz actualizada, os pares de classes $(\{b\},\{c\})$, $(\{b\},\mathbf{1})$ e $(\{c\},\mathbf{1})$ são agora inválidos e, por isso, assinalados com $\times$. A ordem compatível actualizada mantém-se. Assim, a representação piramidal obtida e a respectiva matriz actualizada encontram-se na Figura 2.7.



$$\begin{array}{c} \\ a \\ b \\ c \\ d \\ e \\ \mathbf{1} \end{array} \begin{array}{c} \begin{array}{cccccc} a & b & c & d & e & \mathbf{1} \end{array} \\ \begin{pmatrix} 0 & 0.6 & 0.4 & 0.8 & 0.8 & 0.6 \\ & 0 & \times & 1 & 1 & \times \\ & & 0 & 0.7 & 0.7 & \times \\ & & & 0 & \boxed{0.3} & 1 \\ & & & & 0 & 1 \\ & & & & & 0 \end{pmatrix} \end{array}$$

Figura 2.7: Pirâmide e matriz actualizada (formação da classe $\mathbf{1}$).

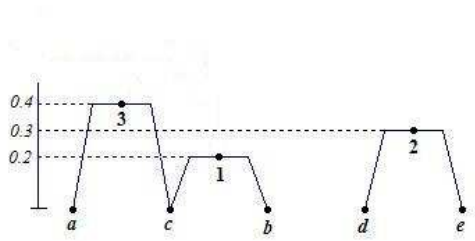
Iteração 2 O menor valor da matriz é $\gamma(d,e) = 0.3$. As classes $\{d\}$ e $\{e\}$ são agregadas dando origem à classe $\mathbf{2} = \{d,e\}$. Analogamente, os pares de classes $(\{d\},\{e\})$, $(\{d\},\mathbf{2})$ e $(\{e\},\mathbf{2})$ são agora inválidos. A ordem compatível mantém-se e obtemos, na Figura 2.8, a representação piramidal e a matriz actualizada.



$$\begin{array}{c} \\ a \\ b \\ c \\ d \\ e \\ \mathbf{1} \\ \mathbf{2} \end{array} \begin{array}{c} \begin{array}{ccccccc} a & b & c & d & e & \mathbf{1} & \mathbf{2} \end{array} \\ \begin{pmatrix} 0 & 0.6 & \boxed{0.4} & 0.8 & 0.8 & 0.6 & 0.8 \\ & 0 & \times & 1 & 1 & \times & 1 \\ & & 0 & 0.7 & 0.7 & \times & 0.7 \\ & & & 0 & \times & 1 & \times \\ & & & & 0 & 1 & \times \\ & & & & & 0 & 1 \\ & & & & & & 0 \end{pmatrix} \end{array}$$

Figura 2.8: Pirâmide e matriz actualizada (formação da classe $\mathbf{2}$).

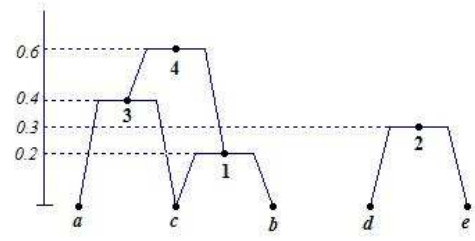
Iteração 3 O menor valor da matriz é $\gamma(a,c) = 0.4$. A classe $\{c\}$ é novamente agregada, agora com a classe $\{a\}$, formando a classe $\mathbf{3} = \{a,c\}$. A ordem compatível actualizada passa a ser $\theta = \{a,c,b,d,e\}$. Uma vez que o elemento c se agregou duas vezes é afastado da matriz. Na Figura 2.9 podemos visualizar a construção da pirâmide e a matriz actualizada.



$$\begin{array}{c} \\ a \\ b \\ d \\ e \\ \mathbf{1} \\ \mathbf{2} \\ \mathbf{3} \end{array} \begin{array}{c} \begin{array}{ccccccc} a & b & d & e & \mathbf{1} & \mathbf{2} & \mathbf{3} \end{array} \\ \begin{pmatrix} 0 & \times & 0.8 & 0.8 & \times & 0.8 & \times \\ & 0 & 1 & 1 & \times & 1 & \times \\ & & 0 & \times & 1 & \times & 0.8 \\ & & & 0 & 1 & \times & 0.8 \\ & & & & 0 & 1 & \boxed{0.6} \\ & & & & & 0 & 0.8 \\ & & & & & & 0 \end{pmatrix} \end{array}$$

Figura 2.9: Pirâmide e matriz actualizada (formação da classe **3**).

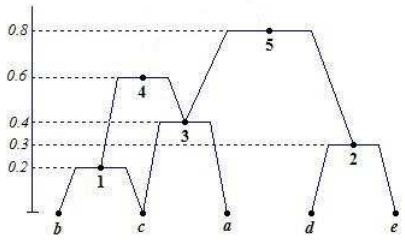
Iteração 4 O menor valor da matriz é $\Gamma(\mathbf{1},\mathbf{3}) = 0.6$. É formada uma nova classe: $\mathbf{4} = \mathbf{1} \cup \mathbf{3} = \{a,c,b\}$. A ordem compatível mantém-se. A representação piramidal e a matriz actualizada podem ser vistas na Figura 2.10.



$$\begin{array}{c} \\ a \\ b \\ d \\ e \\ \mathbf{1} \\ \mathbf{2} \\ \mathbf{3} \\ \mathbf{4} \end{array} \begin{array}{c} \begin{array}{cccccccc} a & b & d & e & \mathbf{1} & \mathbf{2} & \mathbf{3} & \mathbf{4} \end{array} \\ \begin{pmatrix} 0 & \times & 0.8 & 0.8 & \times & 0.8 & \times & \times \\ & 0 & 1 & 1 & \times & 1 & \times & \times \\ & & 0 & \times & 1 & \times & 0.8 & 1 \\ & & & 0 & 1 & \times & 0.8 & 1 \\ & & & & 0 & 1 & \times & \times \\ & & & & & 0 & \boxed{0.8} & 1 \\ & & & & & & 0 & \times \\ & & & & & & & 0 \end{pmatrix} \end{array}$$

Figura 2.10: Pirâmide e matriz actualizada (formação da classe **4**).

Iteração 5 O menor valor é $\Gamma(\mathbf{2},\mathbf{3}) = 0.8$. Então, são agregadas as classe **2** e **3**, formando uma nova classe que designamos por **5**. A ordem compatível é actulizada para $\theta = \{b,c,a,d,e\}$. Na Figura 2.11 podemos ver a representação piramidal e a matriz actualizada com apenas duas classes já que mais nenhuma agregação é possível.



$$\begin{array}{c} \\ \mathbf{4} \\ \mathbf{5} \end{array} \begin{array}{c} \begin{array}{cc} \mathbf{4} & \mathbf{5} \end{array} \\ \begin{pmatrix} 0 & \boxed{1} \\ & 0 \end{pmatrix} \end{array}$$

Figura 2.11: Pirâmide e matriz actualizada (formação da classe **5**).

Iteração 6 Por fim, tem-se $\Gamma(\mathbf{4}, \mathbf{5}) = 1$. São agregadas as classes **4** e **5** formando a classe $\mathbf{6} = \{a, b, c, d, e\} = E$. O processo iterativo termina e a pirâmide obtida pode ser vista na Figura 2.12.

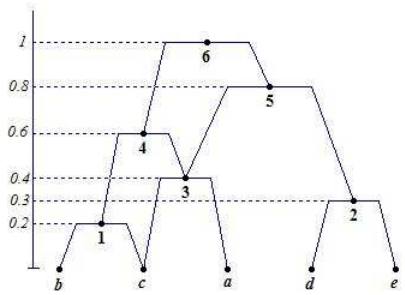


Figura 2.12: Pirâmide final (formação da classe **6**).

2.5 Considerações Finais

Vários autores têm vindo a apresentar outros tipos de estruturas classificatórias, para além das estudadas neste capítulo. De entre estas referem-se de seguida as que parecem mais relevantes neste contexto.

2.5.1 Métodos Alternativos de Classificação

Fichet em 1984 [17] introduziu a noção de Pseudo-Hierarquia, mostrando que é uma estrutura equivalente a uma pirâmide.

Bandelt e Dress em 1989 [3] introduziram as Hierarquias Fracas, também desenvolvidas por Diatta e Fichet em 1994 [11].

A intersecção de três classes pertencentes a uma Hierarquia Fraca é sempre a intersecção entre duas delas.

Formalmente, $\mathcal{F}$ é uma Hierarquia Fraca se:

(i) $E \in \mathcal{F}$

(ii) $\{a\} \in \mathcal{F}, \quad \forall\, a \in E$

(iii) $\mathcal{F}$ é estável para intersecções não vazias, isto é,

$$\forall\, A, B \in \mathcal{F}, \quad A \cap B = \emptyset \quad \text{ou} \quad A \cap B \in \mathcal{F}$$

(iv) $\forall\, A, B, C \in \mathcal{F}$ tem-se $A \cap B \cap C \in \{A \cap B, A \cap C, B \cap C\}$

As Quase-Hierarquias surgem como uma normalização das Hierarquias Fracas.

A Classificação Hierárquica Ascendente *2-3* (C.H.A. *2-3*), introduzida graças aos trabalhos de Bertrand (2002) [6] e [10], generaliza a Classificação Hierarquica Ascendente. Sob certas condições, permite que cada classe possa ser agregada duas vezes, como no caso das pirâmides.

Diz-se que duas classes A e B se *intersectam propriamente* se

$$A \cap B \notin \{\emptyset, A, B\}.$$

Caso contrário, diz-se que o par de classes $\{A, B\}$ é hierárquico.

Formalmente, $\mathcal{H}_{2\text{-}3}$ é uma Hierarquia *2-3* se:

(i) $E \in \mathcal{H}_{2\text{-}3}$

(ii) $\{a\} \in \mathcal{H}_{2\text{-}3}, \quad \forall\, a \in E$

(iii) $\mathcal{F}$ é estável para intersecções não vazias, isto é,

$$\forall\, A, B \in \mathcal{H}_{2\text{-}3}, \quad A \cap B = \emptyset \quad \text{ou} \quad A \cap B \in \mathcal{H}_{2\text{-}3}$$

(iv) Cada classe de $\mathcal{H}_{2\text{-}3}$ intersecta propriamente no máximo uma outra classe de $\mathcal{H}_{2\text{-}3}$.

Do ponto (iv) resulta que $\mathcal{H}_{2\text{-}3}$ é uma Hierarquia *2-3* se e só se qualquer que seja o tripleto de classes consideradas, 2 no mínimo de 3 pares de classes possíveis são hierárquicas, o que justifica o termo Hierarquia *2-3*.

2.5.2 Resumo

Este capítulo centrou-se na comparação de dois tipos de estruturas classificatórias: as hierarquias e as pirâmides. Verificou-se que a Classificação Piramidal, generalização da Classificação Hierárquica, origina relações entre as classes mais complexas. Assim, um elemento do conjunto a classificar, num mesmo patamar, pode pertencer a mais do que uma classe. Para além disso, uma pirâmide produz um pequeno número de ordens, quando comparado com as hierarquias, sobre o conjunto de elementos a classificar.

Algumas definições, propriedades e representações foram apresentadas para ambos os tipos de classificações. O exemplo ilustrativo da Secção 2.4 permite acompanhar os diferentes modos de actuação dos algoritmos de C.H.A. e C.P.A. quando aplicados a um mesmo conjunto de elementos, usando as mesmas funções de comparação entre elementos e entre classes.

Capítulo 3

Geração Aleatória de Dendrogramas

3.1 Introdução

A geração aleatória de uma estrutura classificatória tem diversas aplicações em Classificação. Numa Classificação Hierárquica Ascendente é preciso definir, à partida, as funções de comparação entre elementos e entre classes a utilizar. Assim, diferentes escolhas de funções de comparação conduzem frequentemente a diferentes hierarquias. Torna-se necessário escolher que funções de comparação utilizar e avaliar as que melhor se adequam aos dados. Obtida uma classificação, tentar conhecer a sua qualidade, fiabilidade e robustez. É também necessário decidir qual a partição a reter da hierarquia de partições, isto é, qual o número de classes associado ao conjunto de elementos a classificar. Estes e outros problemas, de difícil resposta, dizem respeito a uma área da Classificação: a **Validação em Classificação**. É neste contexto que a geração aleatória de dendrogramas surge como uma ferramenta útil para contribuir na resolução de vários problemas de validação (ver Sousa [37]).

A abordagem da geração aleatória de dendrogramas e a discussão dos vários métodos disponíveis na literatura é fundamental para uma melhor compreensão dos capítulos que se seguem, isto é, da extensão destes métodos de geração ao caso de uma estrutura piramidal.

Neste capítulo, será feita uma descrição de alguns aspectos importantes para o desenvolvimento do tema. Após uma breve referência às árvores de classificação, será definido um dendrograma, que será visto como uma representação gráfica de uma hierarquia. Para um número pequeno de objectos, as fórmulas para a obtenção do número de dendrogramas e do número de tipos topológicos distintos serão apresentadas, assim como a identificação e respectivas proporções dos diferentes tipos topológicos.

Serão descritos diferentes métodos de geração aleatória de dendrogramas. Começar-se-á por analisar a *Geração Aleatória e Uniforme* de dendrogramas no sentido de Furnas [19], descrevendo três métodos distintos mas que produzem resultados idênticos. Um método de geração aleatória não uniforme será também referido, o da geração de dendrogramas com *Parâmetro de Forma*, que permite a obtenção de árvores de classificação tendencialmente de um certo tipo prefixado.

3.2 Definições associadas aos vários tipos de árvores

3.2.1 Árvores de Classificação

Existem, sobre um conjunto de elementos, várias representações de árvores de classificação. No entanto, um grafo é a que melhor ilustra as relações existentes. A estrutura de árvore é aplicada em diversas áreas científicas pelo que é possível encontrar diferentes terminologias e definições na literatura sobre este tema.

Todos os tipos de árvores estão cobertos pela seguinte definição: *uma árvore é um grafo conexo sem ciclos*. Isto é, para dois elementos quaisquer de um conjunto de objectos há exactamente um caminho que os liga.

Numa árvore os vértices designam-se por *nós* (ou *vértices*) e as aresta por *ramos*. O *grau* de um nó da árvore é o número de ramos ligados a ele. Um *nó terminal* ou *folha* tem sempre grau um. Um nó *interior* tem sempre grau superior a um.

Uma árvore tem *raiz* quando um dos seus nós é classificado de raiz de maneira a introduzir direcção nos ramos da árvore.

Uma árvore diz-se *binária* se nenhum nó interior tem grau superior a três. Se todos os nós interiores têm exactamente grau três, então a árvore diz-se *completamente binária*[1].

Quando aos nós de uma árvore se associam etiquetas esta diz-se *etiquetada*. Nas árvores *C-etiquetadas*, ou *completamente etiquetadas*, a todos os seus nós são atribuídas etiquetas. Se apenas os nós terminais são etiquetados a árvore diz-se *T-etiquetada*, ou *terminalmente etiquetada*. A árvore denomina-se *N-etiquetada*, ou *não etiquetada*, se nenhum tipo de nó é etiquetado.

Quando são atribuídos valores aos vários ramos de uma árvore permitindo associar uma função definida sobre o conjunto dos pares de nós diz-se que a árvore é *ponderada*. Um exemplo de árvores ponderadas são as árvores aditivas ou árvores

[1]Em geral, ao referir-se o termo árvore binária admite-se que ela é completamente binária.

com comprimento de caminho, nas quais se associa a qualquer par de nós o comprimento do caminho que os une, dado pela soma dos pesos dos ramos que constituem esse caminho.

Neste capítulo considera-se um tipo particular de árvores — os *dendrogramas* — que tem bastante interesse pois um dendrograma é a representação gráfica mais frequente de uma Classificação Hierárquica, como se viu na Secção 2.2.

3.2.2 Dendrogramas

Um dendrograma é uma árvore ponderada com raiz, T-etiquetada, em que todos os caminhos dos nós terminais à raiz têm o mesmo comprimento. Ao longo do trabalho consideram-se dendrogramas completamente binários, que por simplificação designar-se-ão por *dendrogramas binários*, isto é, todos os índices de nível ou níveis de agregação são distintos.

Os índices de nível são, muitas vezes, substituídos pelos níveis de agregação ou fusão, que são os valores ordinais dos índices de nível. Dá-se assim mais importância à posição relativa dos nós, em detrimento dos valores reais das distâncias entre eles.

Define-se um par de dendrogramas como isomorfo se os dendrogramas diferem apenas na ordem dos braços dos seus nós. A representação de um dendrograma tem um elevado grau de indeterminação, uma vez que a ordenação esquerda-direita dos braços de cada nó pode mudar. Na Secção 2.2 tinha-se visto que existem 2^{n-1} maneiras diferentes de representar um mesmo dendrograma binário de n objectos.

Um dendrograma pode ser completamente descrito por três propriedades formais:

- **topologia** — tem em conta apenas a forma, ou seja, ignora as etiquetas e os pesos atribuídos aos diferentes ramos. Desta forma, dois dendrogramas são distintos se possuírem sistemas de bifurcação diferentes.
  O número de topologias de um dendrograma é função do número n de nós terminais. O número de tipos topológicos é igual ao número de dendrogramas binários não etiquetados e não ponderados (Harding [22], Murtagh [29]). Existem t_n topologias distintas para dendrogramas binários de ordem n, com t_n obtido pela seguinte recorrência

$$t_n = \begin{cases} \sum_{k=1}^{\frac{n}{2}-1} t_k t_{n-k} + \frac{t_{\frac{n}{2}}\left(t_{\frac{n}{2}}+1\right)}{2} , & \text{se } n \text{ par} \\ \sum_{k=1}^{\frac{n-1}{2}} t_k t_{n-k} \quad , & \text{se } n \text{ ímpar} \end{cases} \tag{3.1}$$

e $t_1 = t_2 = 1$.

Na Tabela (3.1) pode observar-se o número de tipos topológicos distintos para alguns valores de n.

- **etiquetas das folhas** — fixada uma topologia, diferentes fomas de etiquetar os nós terminais conduzem a dendrogramas distintos. Naturalmente que topologias distintas originam dendrogramas distintos, independentemente das etiquetas atribuídas.

  O número de combinações, topologias e etiquetagem, distintas para dendrogramas binários de ordem n é dado pela função a_n (Harding [22], Murtagh [29]), em que:

$$a_n = \frac{(2n-3)!}{2^{n-2}(n-2)!} , \quad \forall n \geq 2. \tag{3.2}$$

  Na Tabela (3.1) pode observar-se também alguns valores de a_n.

- **níveis de agregação** ou **índices de nível** — dois dendrogramas com a mesma topologia e etiquetagem podem diferir nos níveis de agregação ou índices de nível.

  O número d_n de dendogramas binários distinguíveis, isto é, com todos os níveis de fusão distintos, com n nós terminais é dado por (Saporta [35], Podani [32]):

$$d_n = \frac{n!(n-1)!}{2^{n-1}}. \tag{3.3}$$

  Esta fórmula é obtida pela recorrência:

$$\begin{cases} d_1 = 1 \\ d_n = \binom{n}{2} d_{n-1}, \ n \geqslant 2 \end{cases} ,$$

  logo

$$\begin{aligned} d_n &= \tbinom{n}{2}\tbinom{n-1}{2}\tbinom{n-2}{2}\cdots\tbinom{3}{2}\tbinom{2}{2} \\ &= \tfrac{n(n-1)}{2} \times \tfrac{(n-1)(n-2)}{2} \times \tfrac{(n-2)(n-3)}{2} \times \cdots \times \tfrac{3\cdot 2}{2} \times \tfrac{2\cdot 1}{2} \\ &= \tfrac{n!(n-1)!}{2^{n-1}}. \end{aligned} \tag{3.4}$$

  Na tabela (3.1) pode observar-se o número de dendrogramas distintos d_n com n nós terminais, para alguns valores de n.

n	t_n	a_n	d_n
1	1	1	1
2	1	1	1
3	1	3	3
4	2	15	18
5	3	105	180
6	6	945	2 700
7	11	10 395	56 700
8	23	135 135	1 587 600
9	46	2 027 025	57 153 600
10	98	34 459 425	2 571 912 000
15	4 850	2.13×10^{14}	5.14×10^{18}
20	293 547	8.21×10^{21}	1.53×10^{29}
30	1.41×10^{9}	4.95×10^{38}	4.58×10^{52}
40	8.10×10^{12}	1.01×10^{57}	4.83×10^{78}
50	5.15×10^{16}	2.75×10^{76}	$> 10^{100}$

Tabela 3.1: Número de árvores distintas segundo diferentes critérios.

Estas características são usadas numa classificação de dendrogramas muito referenciada na literatura (Sibson [36] e Podani [32]). De acordo com esta classificação, há três categorias de dendrogramas:

- **dendrograma parcialmente ordenado** ou **invariante de ordem local** (LOI – Local Order Invariant): não se atribuem pesos aos nós interiores do dendrograma, ou seja, tem-se em conta apenas a topologia e a etiquetagem. O factor ordem nos nós interiores só se coloca dentro do mesmo ramo, isto é, não se comparam nós pertencentes a ramos diferentes. A informação nível de fusão é local.
- **dendrograma completamente ordenado** ou **invariante de ordem global** (GOI – Global Order Invariant): a cada nó está associado o nível de agregação correspondente. Assim, para além da topologia e etiquetagem, todos os nós interiores estão ordenados, podendo comparar-se nós internos pertencentes a ramos distintos. Portanto, tem-se em conta a topologia, etiquetagem e os níveis de agregação.
- **dendrograma ponderado**: é um dendrograma GOI em que a cada nó interior está associado um índice de nível que é uma variável aleatória contínua cuja escala depende do critério de classificação e do coeficiente de comparação entre elementos do conjunto a classificar.

Um dendrograma ponderado torna-se num dendrograma completamente ordenado quando os índices de nível são substituídos pelas suas ordens. Este, por sua vez, torna-se num dendrograma parcialmente ordenado quando a posição relativa dos níveis em diferentes ramos é irrelevante. Neste trabalho considera-se geração de dendrogramas completamente ordenados. A extensão para dendrogramas ponderados é simples mas necessita da definição da distribuição da variável aleatória índice de nível. Esta definição é contudo delicada porque esta distribuição depende das funções de comparação usadas na construção dos respectivos dendrogramas. A opção de considerar apenas as ordens dos índices de nível não deve ser vista como uma restrição, mas sim como uma atitude de robustez. Em Tendeiro [38] foram feitos estudos de geração aleatória de dendrogramas ponderados para diferentes escolhas de distribuição da variável aleatória índice de nível.

3.2.3 Topologia dos Dendrogramas

O número de dendrogramas distintos cresce exponencialmente à medida que aumenta o número de nós terminais, como pode ser visto na Tabela 3.1. Assim, para 7 objectos tem-se já 56700 dendrogramas distinguíveis e 11 tipos topológicos. Portanto, para uma melhor compreensão e visualização não se consideram valores de n (número de nós terminais) superiores a 6.

Podemos identificar os tipos topológicos para valores de n pequenos e determinar a percentagem de cada tipo topológico no conjunto total de dendrogramas distinguíveis. Para $n = 3$, o número de dendrogramas distintos é 3 e a topologia é única. Quer isto dizer que quando se têm 3 nós terminais a decomposição é única, apenas varia a etiquetagem.

O número de topologias distintas que é possível definir nos dendrogramas de 4 objectos, assim como a proporção de cada tipo topológico, pode ser melhor entendido através da representação esquemática da Figura 3.1. Neste esquema e nos que se seguirão adopta-se uma metodologia aglomerativa, isto é, parte-se das classes singulares até à classe composta por todos os elementos através de fusões sucessivas.

Os quatro uns, $\boxed{1111}$, representam as quatro classes singulares iniciais. Para a primeira agregação tem-se $\binom{4}{2} = 6$ formas distintas de agregar duas classes, o que dará origem a uma classe com 2 elementos e duas classes singulares, representadas por $\boxed{211}$. Na fusão seguinte existem duas agregações possíveis:

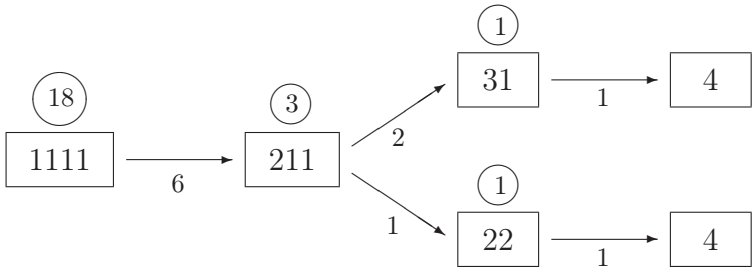


Figura 3.1: Representação esquemática dos dendrogramas para $n = 4$.

31 pode-se agregar a classe com dois elementos com uma das duas com um elemento, originando uma classe com três elementos e uma classe singular. Existem dois modos diferentes de fazer esta fusão;

22 podem-se também agregar as duas classes com um só elemento, formando duas classes com dois elementos cada. Esta agregação é feita de forma única.

Note-se que é neste passo que ficam determinados os tipos topológicos, bem como a sua proporção. Por fim, obtém-se na última agregação quatro objectos numa só classe, representada no esquema por 4 .

A contagem do número de dendrogramas é agora feito em sentido inverso. Assim, o número de dendrogramas em cada passo, que se encontra dentro de uma circunferência, é dado pelo somatório do produto entre o número de agregações possíveis e o número de dendrogramas existentes no passo imediatamente antes (este também representado dentro de uma circunferência). Como se mostra no esquema, resulta que $③ = 2 \times ① + 1 \times ①$ e $⑱ = 6 \times ③$.

Portanto, para $n = 4$, o número de dendrogramas distinguíveis é 18 e os tipos topológicos são dois. Pelo esquema anterior, vê-se que a probabilidade de um dendrograma com 4 nós terminais ter a topologia I é o dobro da probabilidade de ter a topologia II. Na Tabela 3.2 pode observar-se a decomposição topológica, onde 4 representa a classe com quatro objectos e 31 e 22 a decomposição possível desta classes. Isto é, na decomposição 31 podem ser atribuídos 3 elementos a uma classe e 1 a outra; na decomposição 22 são atribuídos 2 elementos a cada uma das classes.

Topologias	Decomposição
I	4 — 31
II	4 — 22

Tabela 3.2: Decomposição topológica para $n = 4$.

Analisa-se de seguida a forma como podem ser deduzidas as proporções dos vários tipos topológicos de dendrogramas gerados. Assim, num nó com k objectos, $(k > 3)$[2], seja i o número de objectos atribuídos ao ramo da direita e, por conseguinte, $k - i$ ao ramo da esquerda. Notando que a situação de serem atribuídos $k - i$ objectos ao ramo da direita e i ao ramo da esquerda, quando $i \neq \frac{k}{2}$, gera dendrogramas topologicamente semelhantes, a probabilidade p_i^k, de se ter a decomposição $(k - i, i)$, depende de k ser par ou ímpar. Assim,

- se k par, tem-se:

$$p_i^k = \frac{2}{k-1}\ , \quad i = 1, \ldots, \frac{k}{2} - 1$$

  e

$$p_{\frac{k}{2}}^k = \frac{1}{k-1}$$

- se k ímpar, tem-se:

$$p_i^k = \frac{2}{k-1}\ , \quad i = 1, \ldots, \frac{k-1}{2}$$

Nos dendrogramas de 4 objectos, a identificação do tipo topológico fica logo determinada na decomposição da raiz. Então a decomposição será efectuada apenas no nó com 4 elementos, $k = 4$. A proporção de dendrogramas do tipo I é dada pela probabilidade de ter a decomposição $(3, 1)$, isto é, $p_1^4 = \frac{2}{3}$. A proporção de dendrogramas do tipo II é dada pela probabilidade de ter a decomposição $(2, 2)$ que é $p_2^4 = \frac{1}{3}$.

Na Figura 3.2 pode ser visualizada a representação destas duas topologias.

[2]Para valores de k menores ou iguais a três o tipo de divisão é único

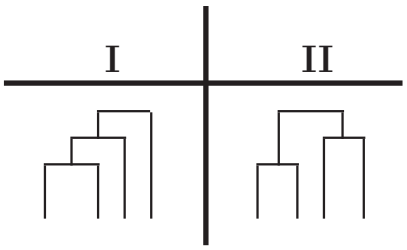


Figura 3.2: Tipos de dendrogramas para $n = 4$.

Para $n = 5$ tem-se, na Figura 3.3, a representação esquemática da obtenção dos diferentes dendrogramas.

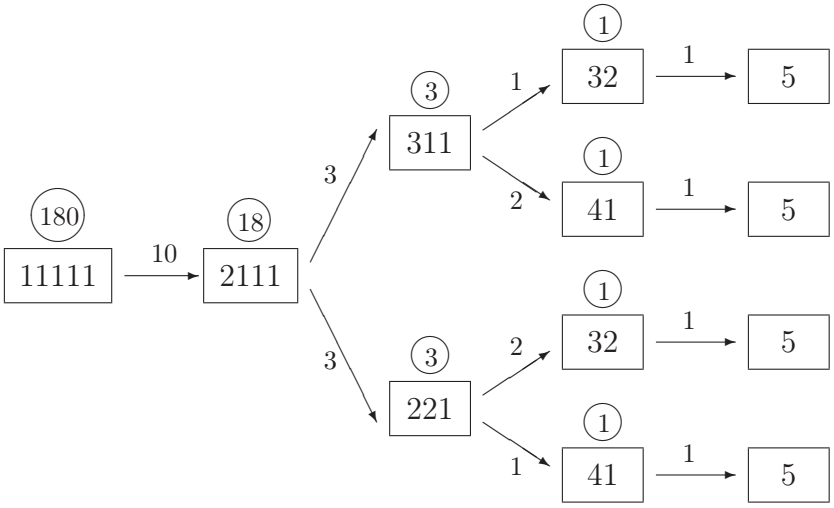


Figura 3.3: Representação esquemática dos dendrogramas para $n = 5$.

Analogamente, pode observar-se, neste esquema, as diferentes formas de agregação para dendrogramas de 5 nós terminais. As cinco classes singulares podem ser agregadas de $\binom{5}{2} = 10$ formas distintas. A agregação seguinte pode ser feita de duas formas: a classe composta com dois elementos ser agregada com uma das classes singulares, dando origem à formação de uma classe com três elementos, ou então a agregação, novamente, de duas classes singulares, que originam a formação de duas classes compostas por dois elementos e uma singular. Ambas podem ser realizadas de 3 formas distintas. Até à formação da classe com cinco elementos pode seguir-se o esquema, que é de fácil compreensão e análogo ao apresentado anteriormente.

Para $n = 5$ o número de dendrogramas distintos é 180 e os tipos topológicos são três. Na tabela 3.3 pode-se observar a respectiva decomposição topológica.

Topologias	Decomposição
I	5 — 41 — 311
II	5 — 32
III	5 — 41 — 221

Tabela 3.3: Decomposição topológica para $n = 5$.

Numa árvore com 5 elementos a determinação das proporções dos diferentes tipos topológicos é ainda simples e permite perceber melhor como se deve proceder na obtenção das probabilidades dos tipos topológicos para árvores com um número de elementos superior. Então, a decomposição da raiz poderá ser feita de duas formas — $(4, 1)$ e $(3, 2)$ — com $p_1^5 = p_2^5 = \frac{2}{4} = 0,5$. A identificação da topologia II fica desde logo determinada com a decomposição $(3, 2)$ e, portanto, a probabilidade de uma árvore ser do tipo II é $p_2^5 = 0,5$. Quando se obtém a decomposição $(4, 1)$, é necessário proceder a uma nova partição do nó com 4 elementos. Como já se viu, tem-se $p_1^4 = \frac{2}{3}$ e $p_2^4 = \frac{1}{3}$. Uma árvore gerada aleatória e uniformemente tem topologia I quando se obtém as decomposições $(4, 1)$ na raiz e $(3, 1)$ no nó com 4 elementos. Portanto, a proporção de árvores com topologia I é dada por $p_1^5 \cdot p_1^4 = \frac{1}{3}$. Analogamente se verifica que a probabilidade da árvore ter topologia III é dada por $p_1^5 \cdot p_2^4 = \frac{1}{6}$.

Na Figura 3.4 tem-se a representação destas três topologias.

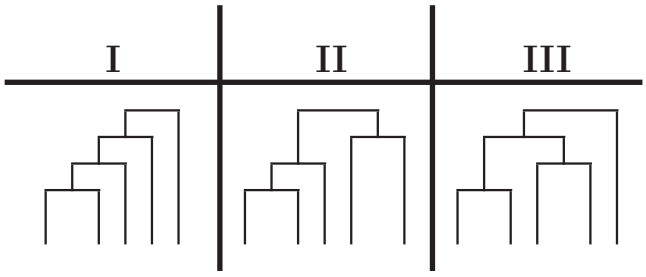


Figura 3.4: Tipos de dendrogramas para $n = 5$.

Para $n = 6$ tem-se, na Figura 3.5, a representação esquemática da obtenção dos diferentes dendrogramas.

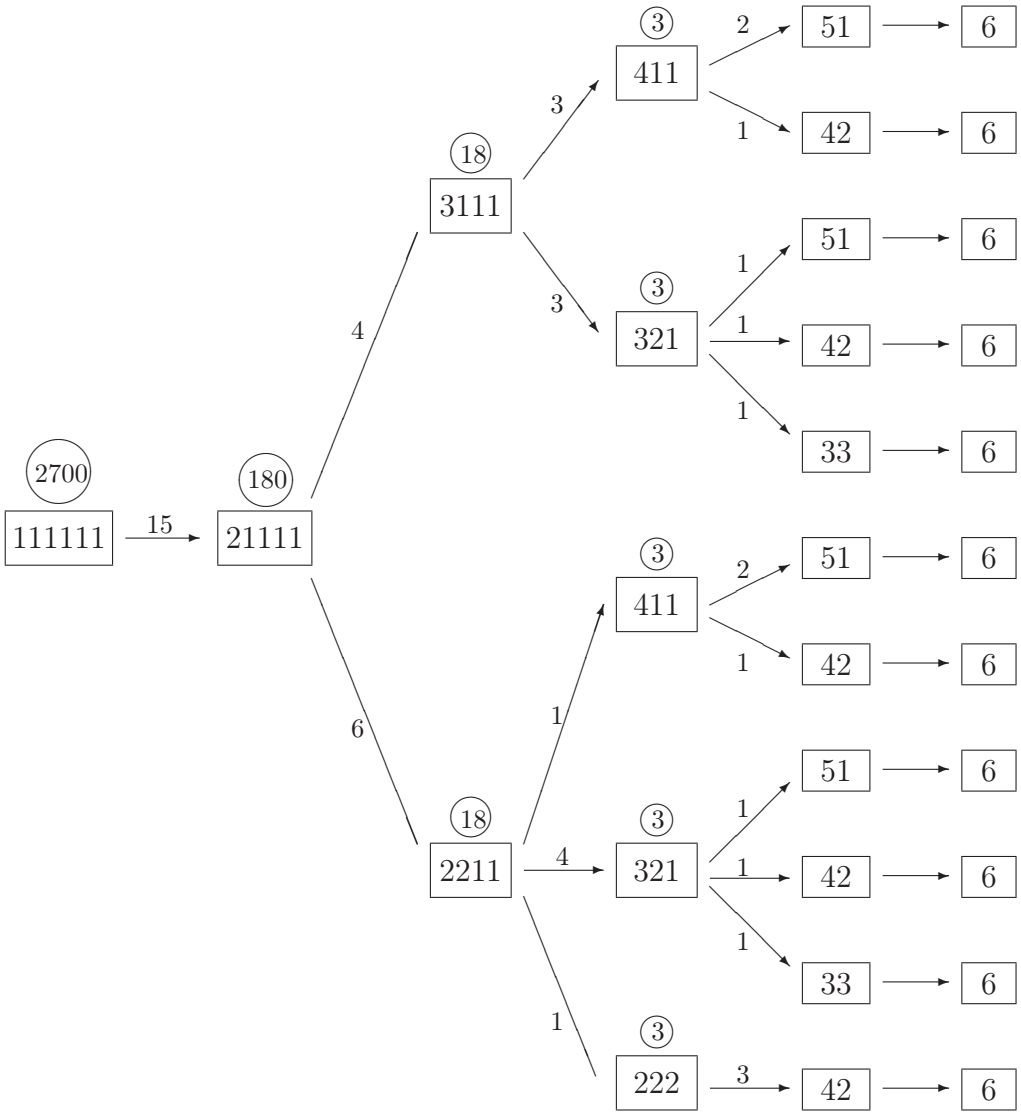


Figura 3.5: Representação esquemática dos dendrogramas para $n = 6$.

Como nos casos anteriores, inicialmente tem-se seis classes singulares, representadas pelos 6 uns. Estas podem ser agregadas de $\binom{6}{2} = 15$ maneiras distintas. A fusão seguinte pode ser realizada de duas formas diferentes: a nova classe com dois elementos ser agregada a uma das restantes quatro classes singulares e, por conseguinte, de quatro maneiras; ou a agregação de duas classes singulares, podendo esta ser feita de $\binom{4}{2} = 6$ formas distintas. A contagem do número de dendrogramas distintos segue como nos esquemas apresentados, para $n = 4, 5$.

Assim, para $n = 6$ o número de dendrogramas distintos é já de 2700 e os tipos topológicos são seis. Na tabela 3.4 pode observar-se a respectiva decomposição topológica.

Topologias	Decomposição
I	6 — 51 — 411 — 3111
II	6 — 51 — 411 — 2211
III	6 — 51 — 321
IV	6 — 42 — 312
V	6 — 42 — 222
VI	6 — 33

Tabela 3.4: Decomposição topológica para $n = 6$.

Para dendrogramas com seis elementos serão também apresentadas as respectivas proporções dos diferentes tipos topológicos. A decomposição da raiz poderá ser feita de três formas distintas — $(5, 1)$, $(4, 2)$ e $(3, 3)$ — com $p_1^6 = p_2^6 = \frac{2}{5} = 0,4$ e $p_3^6 = \frac{1}{5} = 0,2$. As proporções dos diferentes tipos topológicos são dadas por:

Topologia I — $p_1^6 \cdot p_1^5 \cdot p_1^4 = \frac{2}{5} \cdot \frac{2}{4} \cdot \frac{2}{3} = \frac{2}{15} = 0,1333$;

Topologia II — $p_1^6 \cdot p_1^5 \cdot p_2^4 = \frac{2}{5} \cdot \frac{2}{4} \cdot \frac{1}{3} = \frac{1}{15} = 0,0667$;

Topologia III — $p_1^6 \cdot p_2^5 = \frac{2}{5} \cdot \frac{2}{4} = \frac{1}{5} = 0,2$;

Topologia IV — $p_2^6 \cdot p_1^4 = \frac{2}{5} \cdot \frac{2}{3} = \frac{4}{15} = 0,2667$;

Topologia V — $p_2^6 \cdot p_2^4 = \frac{2}{5} \cdot \frac{1}{3} = \frac{2}{15} = 0,1333$;

Topologia VI — $p_3^6 = \frac{1}{5} = 0,2$;

Na Figura 3.6 pode ver-se a representação destas seis topologias.

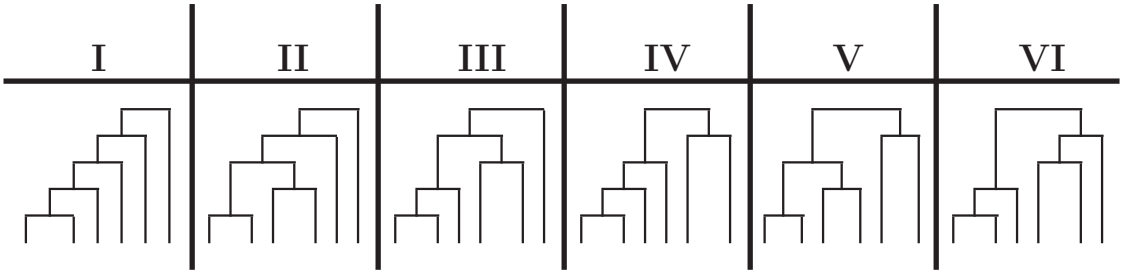


Figura 3.6: Tipos de dendrogramas para $n = 6$.

3.3 Métodos de Geração Aleatória de Dendrogramas

Existem vários métodos de geração de árvores. Nesta subsecção serão analisados apenas métodos de geração aleatória de dendrogramas. Portanto, os três aspectos que caracterizam um dendrograma são gerados aleatoriamente, isto é, as árvores geradas são aleatórias quanto à topologia, a etiquetagem terminal e os níveis de agregação (ou índices de nível).

Alguns métodos para a geração deste tipo de estruturas foram desenvolvidos: o método de Permutação Dupla, proposto por Lapointe e Legendre (1991) [25], o método de Geração Uniforme, apresentado por Sousa (2000) [37], e o método RA (***R**andom **A**glomeration*), desenvolvido por Podani (2000) [32]. Estes três métodos geram dendrogramas aleatória e uniformemente no sentido de Furnas [19]. Quer isto dizer que cada um dos dendrogramas possíveis é gerado equiprovavelmente, isto é, com probabilidade $\frac{1}{d_n}$, onde $d_n = \frac{n!(n-1)!}{2^{n-1}}$ é o número de dendrogramas distintos com n nós terminais.

O método de Permutação Dupla consiste em gerar aleatoriamente a matriz ultramétrica associada a um dendrograma. O processo de Geração Uniforme, assim como o método RA, actuam directamente na construção do dendrograma. No primeiro, o Uniforme, os dendrogramas são gerados partindo da raiz até aos nós terminais, ou seja, adopta-se um procedimento *top-down*. No segundo, o RA, é gerada uma Classificação Hierárquica Ascendente com uma única fusão aleatória em cada passo, ou seja, os dendrogramas são gerados partindo dos nós terminais até à raiz. Para detalhes, estudos comparativos e referências sobre estes métodos ver Sousa [37] e Tendeiro [38].

Sousa [37] propôs também o método de geração aleatória de dendrogramas com Parâmetro de Forma que permite a obtenção de árvores de classificação predominantemente de um certo tipo prefixado. Neste método escolhe-se um valor de $\delta \in]\,0\,;\,0,5\,]$, denominado *parâmetro de forma*, que permite, de forma controlada, variar as probabilidades de ocorrência dos diferentes tipos topológicos.

3.3.1 Método de Permutação Dupla

Este método foi proposto por Lapointe e Legendre (1991) [25].

Como já vimos a um dendrograma está sempre associada uma matriz ultramétrica. Assim, o problema da geração aleatória de dendogramas é equivalente à geração aleatória de matrizes ultramétricas.

Dendrogramas não etiquetados podem ser definidos por um vector de níveis de fusão associado aos seus nós interiores. $n-1$ valores são necessários e suficientes para definir este vector. Usando a propriedade ultramétrica (2.1) pode-se construir a matriz ultramétrica correspondente ao vector dos níveis de fusão. Toda a ordem de permutação deste vector de níveis de fusão corresponde a uma topologia. Todas as topologias são a imagem de pelo menos uma ordem de permutação, mas mais do que uma ordem pode representar a mesma topologia. Pode-se, então, gerar todas as topologias aleatoriamente por uma permutação uniforme do vector de níveis de fusão. A geração uniforme de dendrogramas é completada quando as topologias são etiquetadas aleatoriamente.

Uma matriz ultramétrica aleatória, obtida desta forma, representa um dendrograma aleatório completamente ordenado que é uma árvore com níveis de fusão aleatórios, topologia aleatória e posição das etiquetas aleatórias. Para uma melhor compreensão do método, a sua exposição será ilustrada com um exemplo. O algoritmo foi elaborado em Matlab. Fixado o número de objectos n, para gerar estas matrizes procede-se em três passos:

(1) **Gerar um vector aleatório dos níveis de fusão**[3] de dimensão $n-1$, denominado por $v_{\text{níveis}}$. Para tal é usada uma função geradora uniforme de números pseudo-aleatórios[4] para gerar os $n-1$ primeiros inteiros e permutá-los. É esta permutação que determina aleatoriamente a topologia do dendrograma.

(2) **Preencher a matriz ultramétrica**. Usando apenas os níveis de fusão do vector aleatório pode-se construir uma matriz ultramétrica. Então, numa matriz quadrada de ordem n, o vector obtido no passo anterior é colocado na subdiagonal acima da diagonal principal[5].
A partir desta subdiagonal, as restantes entradas são calculadas fazendo uso da propriedade (2.1), dando origem à matriz preenchida que se denomina por M_{fill}.

(3) **Etiquetar a matriz ultramétrica** (ou, equivalentemente, o dendrograma). São etiquetadas aleatoriamente as folhas da matriz. Esta operação consiste

[3]Poderia, por sua vez, ser gerado um vector aleatório dos índices de nível, mas como foi já referido existem alguns problemas relacionados com a distribuição dos índices de nível a considerar.

[4]No algoritmo desenvolvido em Matlab é usada a função *randperm*.

[5]É suficiente trabalhar com a matriz triangular superior acima da diagonal principal uma vez que contém toda a informação relevante.

em permutar o objecto etiquetado aleatoriamente. Para isso, gera-se aleatoriamente um vector, v_{pos}, de inteiros de 1 até n, das posições das etiquetas. As entradas deste vector correspondem às novas posições dos objectos na matriz. Trata-se, portanto, da segunda permutação implícita no título do método. O que o algoritmo faz é construir uma nova matriz M_{perm}, matriz permutada, que resulta da aplicação $M_{\text{perm}} = M_{\text{fill}}(v_{\text{pos}})$. Esta matriz continua a manter a propriedade ultramétrica como pode ser visto em Tendeiro [38].

Para uma melhor compreensão do método, a sua exposição será acompanhada de um exemplo com 5 objectos.

Exemplo 3.1 No primeiro passo é gerado aleatoriamente o vector $v_{\text{níveis}}$ dos níveis de fusão, de cardinal 4. Obteve-se o vector $v_{\text{níveis}} = (2, 4, 3, 1)$.

No segundo passo, este vector é colocado na subdiagonal acima da diagonal principal,

$$\begin{array}{c} \\ a \\ b \\ c \\ d \\ e \end{array} \begin{array}{c} \begin{array}{ccccc} a & b & c & d & e \end{array} \\ \begin{pmatrix} 0 & 2 & & & \\ & 0 & 4 & & \\ & & 0 & 3 & \\ & & & 0 & 1 \\ & & & & 0 \end{pmatrix} \end{array} \tag{3.5}$$

O preenchimento da matriz é agora feito satisfazendo a propriedade ultramétrica (2.1).

$$M_{\text{fill}} = \begin{array}{c} \\ a \\ b \\ c \\ d \\ e \end{array} \begin{array}{c} \begin{array}{ccccc} a & b & c & d & e \end{array} \\ \begin{pmatrix} 0 & 2 & 4 & & \\ & 0 & 4 & & \\ & & 0 & 3 & \\ & & & 0 & 1 \\ & & & & 0 \end{pmatrix} \end{array} \succ \begin{array}{c} \\ a \\ b \\ c \\ d \\ e \end{array} \begin{array}{c} \begin{array}{ccccc} a & b & c & d & e \end{array} \\ \begin{pmatrix} 0 & 2 & 4 & 4 & \\ & 0 & 4 & & \\ & & 0 & 3 & \\ & & & 0 & 1 \\ & & & & 0 \end{pmatrix} \end{array} \succ$$

$$\succ \begin{array}{c} \\ a \\ b \\ c \\ d \\ e \end{array} \begin{array}{c} \begin{array}{ccccc} a & b & c & d & e \end{array} \\ \begin{pmatrix} 0 & 2 & 4 & 4 & 4 \\ & 0 & 4 & & \\ & & 0 & 3 & \\ & & & 0 & 1 \\ & & & & 0 \end{pmatrix} \end{array} \succ \begin{array}{c} \\ a \\ b \\ c \\ d \\ e \end{array} \begin{array}{c} \begin{array}{ccccc} a & b & c & d & e \end{array} \\ \begin{pmatrix} 0 & 2 & 4 & 4 & 4 \\ & 0 & 4 & 4 & \\ & & 0 & 3 & \\ & & & 0 & 1 \\ & & & & 0 \end{pmatrix} \end{array} \succ$$

$$\succ \begin{array}{c} \\ a \\ b \\ c \\ d \\ e \end{array} \begin{array}{c} \begin{array}{ccccc} a & b & c & d & e \end{array} \\ \begin{pmatrix} 0 & 2 & 4 & 4 & 4 \\ & 0 & 4 & 4 & 4 \\ & & 0 & 3 & \\ & & & 0 & 1 \\ & & & & 0 \end{pmatrix} \end{array} \quad \succ \begin{array}{c} \\ a \\ b \\ c \\ d \\ e \end{array} \begin{array}{c} \begin{array}{ccccc} a & b & c & d & e \end{array} \\ \begin{pmatrix} 0 & 2 & 4 & 4 & 4 \\ & 0 & 4 & 4 & 4 \\ & & 0 & 3 & 3 \\ & & & 0 & 1 \\ & & & & 0 \end{pmatrix} \end{array} \tag{3.6}$$

Por razões algorítmicas é feita a simetrização da matriz. Por fim, falta etiquetar aleatoriamente os nós terminais do dendrograma. É gerado aleatoriamente e permutado um vector de inteiros de 1 até 5 das posições das etiquetas[6], seja $v_{\text{pos}} = (4, 3, 2, 5, 1)$.

Então, a matriz permutada, $M_{\text{perm}} = M_{\text{fill}}(v_{\text{pos}})$, é dada por

$$M_{\text{perm}} = \begin{array}{c} \\ a \\ b \\ c \\ d \\ e \end{array} \begin{array}{c} \begin{array}{ccccc} a & b & c & d & e \end{array} \\ \begin{pmatrix} 0 & 3 & 4 & 1 & 4 \\ & 0 & 4 & 3 & 4 \\ & & 0 & 4 & 2 \\ & & & 0 & 4 \\ & & & & 0 \end{pmatrix} \end{array} \tag{3.7}$$

cujo respectivo dendrograma pode ser visto na figura 3.7.

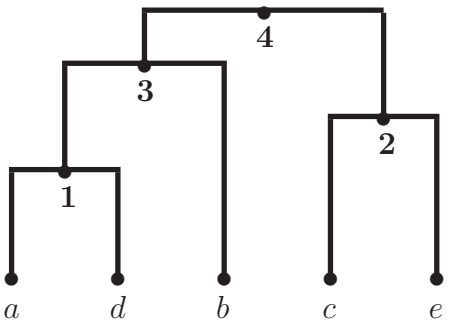


Figura 3.7: Método Permutação Dupla – Dendrograma gerado.

Note-se que o procedimento que foi realizado é equivalente a manter a matriz preenchida, M_{fill}, permutando apenas os objectos segundo o vector $v_{\text{pos}} = (4, 3, 2, 5, 1)$ obtido. Então, o objecto a ocuparia a quarta posição, o objecto b a terceira, c a segunda, d a quinta e e a primeira. A matriz M_{perm} é equivalente à

[6]É novamente usada a função *randperm* do MatLab.

matriz M_{fill} com esta nova ordenação dos objectos,

$$M'_{\text{fill}} = \begin{array}{c} \\ e \\ c \\ b \\ a \\ d \end{array} \begin{array}{c} \begin{array}{ccccc} e & c & b & a & d \end{array} \\ \begin{pmatrix} 0 & 2 & 4 & 4 & 4 \\ & 0 & 4 & 4 & 4 \\ & & 0 & 3 & 3 \\ & & & 0 & 1 \\ & & & & 0 \end{pmatrix} \end{array} \tag{3.8}$$

O método da Permutação Dupla é uniforme no sentido de Furnas, uma vez que para uma ordem n fixa gera cada um dos dendrogramas possíveis de forma equiprovável, isto é, com probabilidade $\frac{1}{d_n}$. Lapointe e Legendre [25] apresentam uma prova sucinta deste resultado, em Tendeiro [38] pode ser vista uma demonstração mais completa.

3.3.2 Método de Geração Uniforme

Este método foi proposto por Sousa em 2000 [37].

O método actua partindo da raiz para os nós terminais. Adopta-se, portanto, um procedimento *top-down*. Numa árvore binária com n nós terminais tem-se $n-1$ níveis de fusão. Inicialmente tem-se um conjunto de objectos de cardinal n e outro conjunto de níveis de fusão (ou índices de nível) de cardinal $n-1$. Sobre cada nó da árvore, tendo início na raiz, será feita uma divisão aleatória do conjunto dos objectos e dos níveis de fusão.

Neste método, a informação de cada nó é decomposta em dois ramos. Assim, é gerado aleatória e uniformemente, em cada nó, o número de objectos que serão atribuídos ao ramo da esquerda e, consequentemente, ao ramo da direita. O subconjunto dos objectos e o subconjunto dos níveis atribuídos ao ramo da esquerda (e, por conseguinte, ao da direita) são obtidos aleatoriamente. Na metodologia proposta por Sousa [37] esta decomposição actua sucessivamente nos ramos da esquerda, passando depois para os da direita e procedendo de forma análoga. O processo termina quando o ramo mais à direita está decomposto.

No algoritmo proposto a decomposição efectuada em cada nó é igual ao método exposto, apenas não actua sucessivamente nos ramos da esquerda. Se o algoritmo está a actuar num nó cujo nível de agregação é m, então no passo seguinte irá actuar no nó cujo nível de agregação é $m-1$.

A Figura 3.8 é um esquema do modo de actuação do método em cada nó.

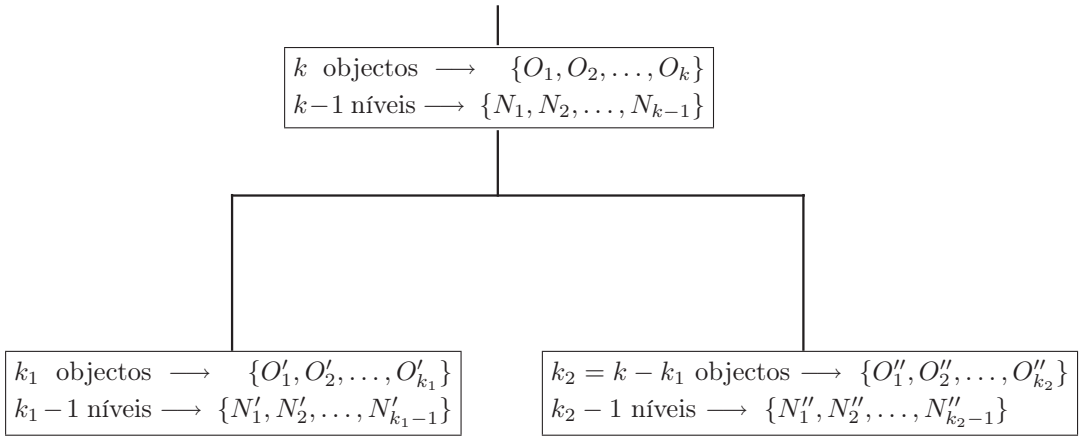


Figura 3.8: Filosofia-base para o método da geração Uniforme

Num determinado nó, seja $\{O_1, O_2, \ldots, O_k\}$ o conjunto dos objectos que o constituem, de cardinal k ($k \geq 2$). É gerado um inteiro k_1 uniforme entre 1 e $k-1$, sendo feita uma decomposição aleatória dos k objectos em duas classes: a do ramo da esquerda com k_1 objectos, $\{O'_1, O'_2, \ldots, O'_{k_1}\}$; e a do ramo da direita com $k_2 = k - k_1$ objectos, $\{O''_1, O''_2, \ldots, O''_{k_2}\}$. Dos $k-1$ níveis, o maior valor é atribuído ao nó onde está a ser efectuada a decomposição. Dos restantes $k-2$, serão atribuídos aleatoriamente $k_1 - 1$ níveis ao ramo da esquerda e $k_2 - 1$ ao ramo da direita, como mostra a Figura 3.8.

Analise-se novamente um pequeno exemplo com 5 nós terminais para uma melhor compreensão do modo de actuação do algoritmo.

Exemplo 3.2 Considere-se $O = \{a, b, c, d, e\}$ o conjunto dos cinco objectos e $N = \{1, 2, 3, 4\}$ o conjunto dos níveis de agregação. Como já foi referido o maior valor do nível de fusão, 4, fica associado à raiz. De seguida, é realizada a divisão aleatória e uniforme dos objectos, sendo atribuídos 3 objectos ao ramo da esquerda. As partições obtidas foram as seguintes: $O_1 = \{b, c, e\}$ para o ramo da esquerda e $O_2 = \{a, d\}$ para o ramo da direita. Dos 3 níveis restantes dois foram atribuídos ao ramo da esquerda, $N_1 = \{1, 3\}$, e um ao ramo da direita, $N_2 = \{2\}$. O que o algoritmo faz é actuar de forma análoga no nó com nível de agregação maior. Assim, no ramo da esquerda fica desde logo associado ao nó o nível 3, sendo depois a decomposição feita de forma única atribuindo-se apenas aleatoriamente as etiquetas. Os objectos b e e do conjunto $O_{11} = \{b, e\}$ agregam-se no nível 1. No nível 2 é realizada a agregação dos objectos a e d.

O dendrograma obtido, assim como as decomposições dos objectos e níveis podem ser vizulizados na Figura 3.9.

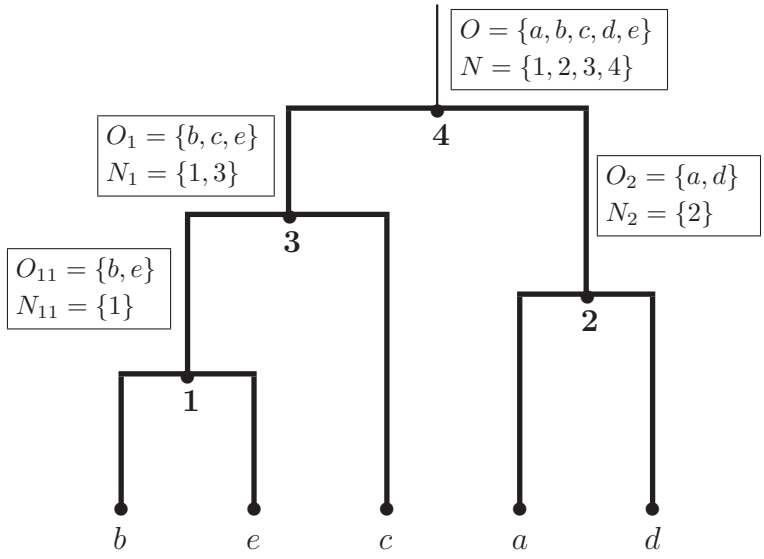


Figura 3.9: Método Geração Uniforme – Dendrograma gerado.

3.3.3 Método RA

O método RA – ***R**andom **A**gglomeration Method* – foi proposto por Podani (2000) [32].

Mais simples e de fácil compreensão, o método produz resultados idênticos aos métodos apresentados anteriormente. O algoritmo simula uma classificação hierárquica com uma única fusão aleatória em cada passo. É, portanto, um método aleatório de agregação que produz dendrogramas binários completamente ordenados. Ao contrário do método de Geração Uniforme este actua dos nós terminais até à raiz.

Para gerar um dendrograma de n objectos, no primeiro passo é feita a escolha aleatória do par de objectos entre os $\binom{n}{2}$ possíveis. No segundo passo, completamente independente do anterior, tem-se $\binom{n-1}{2}$ de pares possíveis, que resulta dos $n-2$ objectos isolados e da classe já formada no passo anterior. O processo continua de forma análoga até serem agregados todos os objectos numa só classe. Toda a informação deste procedimento é guardada numa matriz $(n-1) \times 2$, que será designada por T.

Como se está a gerar dendrogramas completamente ordenados são realizados $n-1$ passos. Cada passo é independente dos outros, pelo que a probabilidade do método RA gerar um dendrograma é dada por

$$\frac{1}{\binom{n}{2}} \times \frac{1}{\binom{n-1}{2}} \times \cdots \times \frac{1}{\binom{3}{2}} \times \frac{1}{\binom{2}{2}} = \frac{1}{\binom{n}{2} \times \binom{n-1}{2} \times \cdots \times \binom{3}{2} \times \binom{2}{2}} = \frac{1}{d_n},$$

onde o número de dendrogramas distintos d_n é dado pela Fórmula (3.3), da página 34.

O algoritmo desenvolvido por Podani pode ser visualizado em [32] na Tabela 1. No Anexo 4 encontra-se o mesmo algoritmo só que desenvolvido em linguagem Matlab.

Segue-se um exemplo com 5 nós terminais que permitirá compreender melhor o modo de actuação do algoritmo.

Exemplo 3.3 Seja $E = \{a, b, c, d, e\}$ o conjunto dos elementos a classificar. Denominam-se estes elementos pelos naturais $1, 2, \ldots, 5$ e define-se $v = (1, 2, 3, 4, 5)$ como o vector dos elementos a classificar. Existem então 4 fases de agregação que se encontram detalhadas nos passos que se seguem.

Passo 1: são escolhidos aleatoriamente dois objectos a agregar. j e k, com $j \neq k$, são gerados aleatoriamente do conjunto $\{1, 2, 3, 4, 5\}$. Estes números permitem obter as posições dos dois primeiros objectos a agregar. Assim, suponha-se $j = 4$ e $k = 2$. Os objectos a agregar são $v(4) = 4$ e $v(2) = 2$, ou seja, d e b. Na matriz T, de dimensões 4×2, são colocados os valores de $v(4)$ e $v(2)$ por ordem crescente na primeira linha. No vector v a nova classe formada é representada pelo menor valor de $v(4)$ e $v(2)$, neste caso 2, sendo o outro colocado na última posição do vector v. Este vector é agora da seguinte forma: $v = (1, 2, 3, 5, 4)$.

Passo 2: do conjunto $\{1, 2, 3, 4\}$ são gerados aleatoriamente os dois números j e k, suponha-se $j = 3$ e $k = 1$. São portanto agregados os objectos $v(3) = 3$ e $v(1) = 1$, isto é, c e a. Na matriz T vem agora $T(2, 1) = 1$ e $T(2, 2) = 3$. O vector v é actualizado, vindo $v = (1, 2, 5, 3, 4)$.

Passo 3: são gerados do conjunto $\{1, 2, 3\}$ dois números. Seja $j = 1$ e $k = 2$. São então agregadas as classes representadas por $v(1) = 1$ e $v(2) = 2$, isto é, as classes formadas nos dois passos anteriores. Na terceira linha da matriz T são colocados os valores de $v(1)$ e $v(2)$ por ordem crescente. O vector v é agora dado por $v = (1, 5, 2, 3, 4)$.

Passo 4: restam apenas duas classes a agregar: a classe composta por quatro elementos representada por 1 e a classe singular 5. É então formada a classe composta por todos os elementos.

A matriz T é dada por $T = \begin{pmatrix} 2 & 4 \\ 1 & 3 \\ 1 & 2 \\ 1 & 5 \end{pmatrix}$.

Na Figura 3.10 pode observar-se o dendrograma gerado pelo método RA.

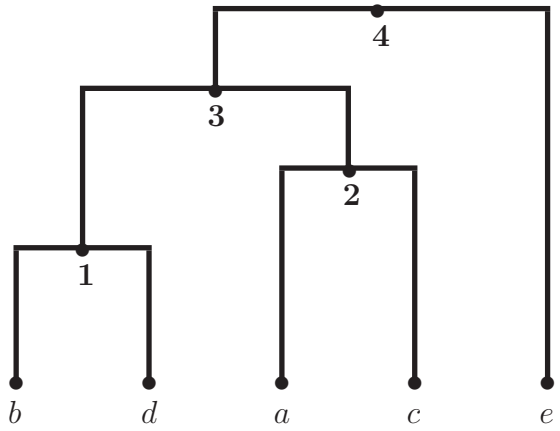


Figura 3.10: Método RA – Dendrograma gerado.

No Anexo 4 pode ser visto também um algoritmo desenvolvido que permite, a partir da matriz T, obter a respectiva matriz ultramétrica do dendrograma gerado.

3.3.4 Método de Geração com Parâmetro de Forma

A Geração de Dendrogramas com *Parâmetro de Forma* foi, como já referido, proposta por Sousa [37] e a sua metodologia de actuação é semelhante à do método de Geração Uniforme exposto anteriormente. A diferença nestes dois processos de geração de dendrogramas está na forma como é feita a divisão do número de objectos que são atribuídos ao ramo da esquerda e, por conseguinte, ao ramo da direita. Neste método é introduzido um coeficiente p, com $p \in]0; 0,5]$, denominado parâmetro de forma, que permite a geração de árvores de um determinado tipo desejado. Assim, em cada nó, cada elemento tem probabilidade p de ir para o ramo da esquerda e $1 - p$ de ir para o ramo da direita.

Este processo de geração tem bastante interesse já que permite a geração de dendrogramas de um determinado tipo prefixado. A tendência para gerar dendrogramas do tipo cadeia aumenta à medida que o parâmetro p diminui. Para valores de p próximos de 0.5 os dendrogramas gerados são predominantemente "arredondados" ou "equilibrados", isto é, o número de objectos que em cada nó é atribuído ao ramo da esquerda é sensivelmente o mesmo que é atribuído ao ramo da direita. Assim, num dado nó com k objectos, o número de objectos que irá para o ramo da esquerda seguirá uma distribuição próxima de uma variável aleatória binomial de parâmetros k e p. Diz-se próxima uma vez que as realizações 0 e k não são possíveis, por imposição do algoritmo que não permite a atribuição de nenhum ou k objectos a um dos ramos.

3.4 Considerações Finais

3.4.1 Comparação do Desempenho dos Métodos

Os três métodos de geração uniforme no sentido de Furnas, isto é, o da Permutação Dupla, o da Geração Uniforme e o RA têm comportamentos semelhantes e produzem resultados idênticos.

Sousa [37] fez um estudo analítico de dois destes métodos de geração aleatória de dendrogramas, o da Permutação Dupla e o da Geração Uniforme. Numa primeira fase, calculou as probabilidades teóricas que cada método tem para gerar um dendrograma de cada um dos tipos topológicos possíveis de ordem n. Nos quatro casos analisados, $n = 4, 5, 6, 7$, comprovou-se que o comportamento probabilístico dos métodos da Permutação Dupla e Uniforme é exactamente o mesmo. Para o método RA este comportamento é bastante semelhante. Quer isto dizer que ambos os métodos geram um dendrograma com um determinado tipo topológico com igual probabilidade, o que seria de esperar pois como já se tinha visto estes três métodos são uniformes no sentido de Furnas.

Sousa recorreu à simulação para fazer uma análise comparativa dos métodos da Geração Uniforme e da Permutação Dupla e concluiu que têm desempenhos bastante análogos. Se por um lado a simplicidade e naturalidade associadas ao método de Geração Uniforme o tornam mais atraente, por outro lado a complexidade algorítmica do método da Permutação Dupla é mais reduzida.

Tendeiro [38] fez um estudo da complexidade computacional dos três métodos de geração uniforme. A complexidade computacional pretende aferir a dificuldade

de execução dum algoritmo. Mostrou que a complexidade algorítmica associada ao método da Permutação Dupla é $\mathcal{O}(n^2)$. Concluiu que o método da Geração Uniforme tem um tempo de execução de complexidade $\mathcal{O}(n^3)$. A complexidade associada ao método RA é de $\mathcal{O}(n^2)$. Este método produz uma matriz T, de dimensões $(n-1)\times 2$, que contém toda a informação necessária para representar o dendrograma correspondente. Torna-se útil converter a informação contida em T para a matriz ultramétrica. Para esta passagem Tendeiro apresentou dois processos possíveis, o primeiro com tempo de complexidade $\mathcal{O}(n^4)$ e o segundo $\mathcal{O}(n^3)$.

Este estudo permitiu concluir que, dos três métodos, o mais eficiente sob o ponto de vista computacional é o método da Permutação Dupla.

O método de geração com parâmetro de forma tem a vantagem, em relação aos restantes métodos, de permitir controlar a forma final do dendrograma gerado. Neste sentido este método não é completamente aleatório, sendo no entanto de grande utilidade prática em variados contextos.

3.4.2 Mistura de Dendrogramas

Um aspecto que poderá ter bastante interesse de estudo é a mistura de dendrogramas gerados aleatoriamente por métodos diferentes ou então gerados pelo mesmo método, mas que produza dendrogramas topologicamente distintos. Isto é, gerar, por exemplo, dendrogramas pelo método de Geração com Parâmetro de Forma atribuindo coeficientes de p distintos aos dendrogramas gerados independentemente. Este método permitirá obter dendrogramas com topologias prefixadas.

O estudo deste assunto não foi aprofundado ficando apenas a sugestão do que poderá ser desenvolvido e a visualização do tipo de dendrogramas que é possível obter. Foi elaborado um algoritmo (Anexo 6) que permite gerar dois dendrogramas pelo método de Geração com Parâmetro de Forma com valores de p distintos e número de objectos em cada dendrograma também diferenciado. Para visualizar os dendrogramas que é possível gerar observe-se a Figura 3.11 que foi obtida atribuindo oito objectos na geração de cada dendrograma com coeficientes de p diferentes: $0,05$ no ramo da esquerda e $0,5$ no ramo da direita.

Da análise da Figura 3.11 verifica-se facilmente que este processo de geração de dendrogramas permite obter dendrogramas com ramificações bastante diferenciadas. Assim, é possível obter com frequência dendrogramas em que um dos ramos é do tipo cadeia e outro do tipo "arredondado".

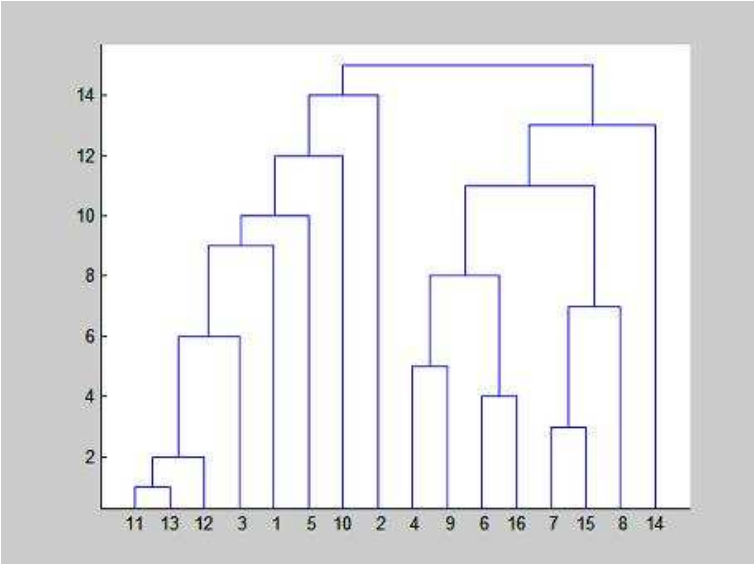


Figura 3.11: Mistura de dendrogramas.

3.4.3 Implementação dos Algoritmos

Os diferentes métodos apresentados foram desenvolvidos em linguagem Matlab, bem como outros algoritmos que permitem o estudo de algumas propriedades dos dendrogramas. Assim,

- no Anexo 1 são apresentados algoritmos que permitem a identificação topológica de dendrogramas com 4, 5 e 6 nós terminais. Esta identificação é feita partindo da matriz ultramétrica que é transformada num vector denominado vector ultramétrico;
- no Anexo 2 tem-se o algoritmo do método da Permutação Dupla;
- no Anexo 3 é apresentado o algoritmo do método da Geração Uniforme;
- no Anexo 4 pode visualizar-se o algoritmo do método RA;
- no Anexo 5 é apresentado o algoritmo de Geração com Parâmetro de Forma. Este algoritmo é descrito parcialmente porque é muito semelhante ao método de Geração Uniforme. Difere apenas na função que gera os objectos e níveis a atribuir a cada ramo;
- no Anexo 6 é apresentado o algoritmo que permite a mistura de dois dendrogramas gerados de forma independente pelo método de Geração com Parâmetro de Forma.

3.4.4 Resumo

Neste capítulo foram apresentados algoritmos de geração aleatória de dendrogramas com o objectivo de alargar estes métodos para a geração aleatória de pirâmides.

Introduziram-se algumas noções associadas às árvores de classificação, em particular aos dendrogramas. Seguiu-se um estudo topológico de dendrogramas com 4, 5 e 6 nós terminais. Foram descritos três algoritmos de geração aleatória e uniforme no sentido de Furnas: o da Permutação Dupla, o da Geração Uniforme e o RA. Estes métodos foram acompanhados de pequenos exemplos que auxiliaram a compreensão dos mesmos. Os resultados produzidos pelos diferentes métodos de geração são bastante semelhantes. Foi também referenciado o método de geração com Parâmetro de Forma que permite a geração de dendrogramas de um determinado tipo prefixado.

Capítulo 4

Algoritmos de Classificação Piramidal Ascendente

4.1 Introdução

Este capítulo é dedicado ao estudo e evolução de alguns algoritmos de Classificação Piramidal Ascendente, sem dúvida os mais usados, encontrados na literatura. Serão descritos os algoritmos propostos por Diday (1984) [12] e [13], Bertrand (1986) [4] e Mfoumoune (1998) [27]. No final, será feita ainda referência aos algoritmos propostos por Brito (1991) [9] e Durand (1989) [16].

Inicialmente serão apresentadas algumas noções e propriedades necessárias à implementação e compreensão dos algoritmos. Estas irão complementar as definições apresentadas no Capítulo 2 sobre a Classificação Piramidal. Os três algoritmos descritos apresentam melhoramentos e são uma base, juntamente com os algoritmos de geração aleatória de dendrogramas, para a construção do algoritmo de geração aleatória de uma pirâmide proposto no Capítulo 5.

4.2 Principais Noções

Nesta subsecção segue-se de perto a nomenclatura usada em Bertrand [4] e Mfoumoune [27]. Para ilustrar estas noções, considere-se a Figura 4.1.

Seja E o conjunto de elementos a classificar, neste caso de cardinal 13, com $E = \{a, b, c, d, e, f, g, h, i, j, k, l, m\}$. Designa-se por P a pirâmide incompleta, uma vez que $E \notin P$. Considere-se também θ uma ordem compatível com P, que neste exemplo é $\theta = (e, g, f, l, m, i, k, a, h, b, c, j, d)$.

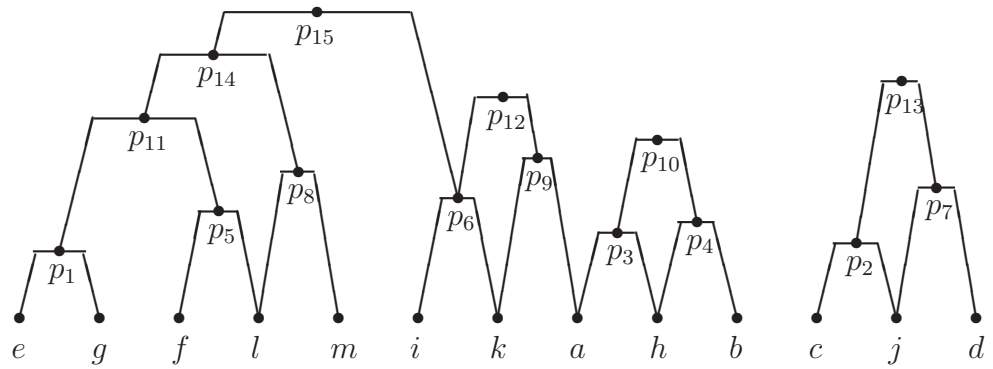


Figura 4.1: Pirâmide incompleta para ilustração de noções.

Seja $p \in P$. Designa-se por $min\,(p)$ o menor elemento da classe p e $max\,(p)$ o maior elemento da classe p, segundo a ordem compatível θ. Na Figura 4.1, para a classe p_{12}, tem-se que $min\,(p_{12}) = i$ e $max\,(p_{12}) = a$.

4.2.1 Sucessores e Predecessores

As noções de sucessor e predecessor já foram apresentadas no Capítulo 2.

Uma classe que não possua nenhum predecessor será chamada de ***classe maximal*** e uma classe que não tenha nenhum sucessor é designada por ***classe singular*** ou ***terminal***. Uma classe não singular $p \in P$ possui sempre dois sucessores e, no máximo, dois predecessores, como vimos na proposição 2.3, da página 17.

Na Figura 4.1:

- As classes p_6 e p_9 são sucessores de p_{12} e, por conseguinte, p_{12} é predecessor de p_6 e p_9;
- A classe p_6 tem dois predecessores: p_{15} e p_{12};
- As classes p_{15}, p_{12}, p_{10} e p_{13} são classes maximais;
- As classes $\{a\}, \ldots, \{m\}$ são classes singulares.

Predecessor à esquerda

Diz-se que uma classe p é um *predecessor à esquerda* duma classe q se e só se p é um predecessor de q e $max\,(p) = max\,(q)$.

Na Figura 4.1, p_{15} é um predecessor à esquerda de p_6.
Diz-se igualmente que p_6 é um *sucessor à direita* de p_{15}.

Predecessor à direita

Diz-se que uma classe p é um *predecessor à direita* duma classe q se e só se p é um predecessor de q e $min\,(p) = min\,(q)$.

Na Figura 4.1, p_{12} é um predecessor à direita de p_6.
Diz-se igualmente que p_6 é um *sucessor à esquerda* de p_{12}.

Descendentes e Ascendentes

Considere-se uma pirâmide (incompleta) P, diz-se que uma classe p é um *descendente* da classe q se e só se $p \subset q$ (p está incluído em q). Na Figura 4.1, o conjunto dos descendentes de p_{12} é $D\,(p_{12}) = \left\{p_6, p_9, \{i\}, \{k\}, \{a\}\right\}$.

Diz-se que uma classe p é um *ascendente* da classe q se e só se $p \supset q$ (p contém q). Na Figura 4.1, o conjunto dos ascendentes da classe p_5 é $A\,(p_5) = \{p_{11}, p_{14}, p_{15}\}$.

Em Mfoumoune [27] são apresentados algoritmos que permitem determinar o conjunto dos descendentes e ascendentes de uma dada classe, assim como as suas complexidades algorítmicas.

4.2.2 Ordem Piramidal

Recorde-se a **noção de ordem**. Uma *relação binária* R definida sobre um conjunto não vazio E é uma *ordem* se e só se ela é

(i) *reflexiva*: $\forall x \in E, \quad xRx$;

(ii) *transitiva*: $\forall x, y, z \in E, \quad (xRy \wedge yRz) \Longrightarrow xRz$;

(iii) *antisimétrica*: $\forall x, y \in E, \quad (xRy \wedge yRx) \Longrightarrow x = y$.

Se esta ordem verifica ainda a propriedade seguinte:

(iv) $\forall x, y \in E,\, x \neq y, \quad (xRy \vee yRx)$,

então a relação R é uma *ordem total* sobre E.

Relembre-se igualmente a noção de **ordem compatível**. Dado um conjunto finito de elementos E e d um índice de dissemelhança sobre E, diz-se que uma ordem θ sobre E é compatível com o índice de dissemelhança d, dito indíce piramidal (2.18), se:

$$\forall\, x, y, z \in E, \quad x <_\theta y <_\theta z \quad \Longrightarrow \quad d(x,z) \geq max\big\{d(x,y), d(y,z)\big\}.$$

Seja P uma pirâmide sobre E e θ a ordem total sobre E induzida por P. Diz-se que uma parte X de E é um intervalo de θ se:

$$\forall\, x, y \in X, \quad \forall\, z \in E, \quad x <_\theta z <_\theta y \quad \Longrightarrow \quad z \in X.$$

Toda a classe pertencente a P é um intervalo de θ por definição de pirâmide.

Definem-se a seguir algumas relações de ordem sobre P.

4.2.3 Relações de Ordem

Sejam p e q duas classes de uma pirâmide (incompleta) P.

Relações "*antes de*" e "*depois de*"

Diz-se que p está *antes de* q se

$$\big(min(p) <_\theta min(q) \text{ e } max(p) <_\theta max(q)\big) \text{ ou } p = q.$$

Se p está *antes de* q, diz-se também que q está *depois de* p (Bertrand [4]).

Relações "*à esquerda de*" e "*à direita de*"

p está *à esquerda* de q se

$$min(p) \leq_\theta min(q) \text{ e } max(p) \leq_\theta max(q).$$

p está *à direita* de q se q está *à esquerda* de p.

p está *estritamente à esquerda* de q se

$$min(p) <_\theta min(q) \text{ e } max(p) = max(q).$$

p está *estritamente à direita* de q se

$$min(p) = min(q) \text{ e } max(p) <_\theta max(q).$$

Relação "*no interior de*"

Diz-se que p é *interior* a q se

$$min\,(q) <_\theta min\,(p) \;\text{ e }\; max\,(p) <_\theta max\,(q)\,.$$

Por exemplo, a classe p_5 é interior a p_{14} (Figura 4.1).

Ordem de uma classe singular

Seja (P, θ) uma pirâmide sobre E e θ uma ordem sobre as classes singulares de P. A ordem duma classe singular p, designada por $Ord\,(p)$, é o número de ocorrências (incluindo p) sobre a ordem θ, partindo da esquerda para a direita.

Na Figura 4.1, $Ord(\{i\}) = 6$.

4.2.4 Componentes Conexas

Antes de definir componente conexa é importante perceber a noção de caminho entre duas classes.

Diz-se que uma sequência $\{p_i : i = 1, \ldots, k\}$ de classes é um *caminho* (Bertrand [4]) se:

- $\forall\, i < k,\ p_i$ está antes de p_{i+1}, no sentido da ordem compatível;
- $\forall\, i < k,\ max\,(p_i)$ é superior ou igual a $min\,(p_{i+1})$, no sentido da ordem compatível.

Por exemplo, na Figura 4.1, $\left\{ \{e\}, p_1, p_{11}, p_{14}, p_8, \{m\} \right\}$ é um caminho que liga $\{e\}$ a $\{m\}$.

Seja C uma parte de uma pirâmide incompleta P. Diz-se que C é uma *componente conexa* de P se:

(i) quaisquer que sejam p e q pertencentes a C, existe um caminho que une p a q;

(ii) C é a maior parte de P, no sentido de inclusão, a verificar a propriedade (i).

Por exemplo, na Figura 4.1, $C_{\{e,g,f,l,m,i,k,a,h,b\}}$ e $C_{\{c,j,d\}}$ são as duas componentes conexas da pirâmide incompleta P.

Designa-se por $C(p)$ a componente conexa da classe p. Na Figura 4.1, $C(p_{12})$ é a primeira componente conexa, isto é, $C_{\{e,g,f,l,m,i,k,a,h,b\}}$ e $C(p_7) = C_{\{c,j,d\}}$ a segunda.

Viu-se que uma classe $p \in P$ é maximal se não tem predecessores. Nota-se por $M\big(C(p)\big)$ o conjunto das classes maximais da componente conexa da classe p. Isto é,

$$M\big(C(p)\big) = \{S_j : j = 1, \ldots, k\},$$

onde S_j, com $j = 1, \ldots, k$, são classes maximais e S_j está antes de S_{j+1}.

Por exemplo, $M\big(C(p_6)\big) = \{p_{15}, p_{12}, p_{10}\}$ (Figura 4.1).

4.2.5 Classes Extremas

Seja $C(p)$ a componente conexa da classe p e θ a ordem sobre os elementos. A classe p é *extrema* se e só se

$$min(p) = min\big(C(p)\big) \qquad ou \qquad max(p) = max\big(C(p)\big)$$

Por exemplo, $\{e\}, p_1, p_{11}, p_{14}, p_{15}, p_{10}, p_4, \{b\}, \{c\}, p_2, p_{13}, p_7, \{d\}$ são as classes extremas na pirâmide incompleta P da Figura 4.1.

4.2.6 Classes Hierárquicas e Classes Maximais Hierárquicas

Em Bertrand [4] investiga-se as classes que se comportam, do ponto de vista duma ordem compatível com P, como uma classe duma hierarquia. Para isso, define-se:

Classes Hierárquicas

Diz-se que $p \in P$ é uma *classe hierárquica* se:

$$\forall\, x \in P, \quad x \neq p, \quad p \cap x \in \{\emptyset, p, x\}$$

Designa-se por $\mathbb{H}$ o conjunto das classes hierárquicas de P.

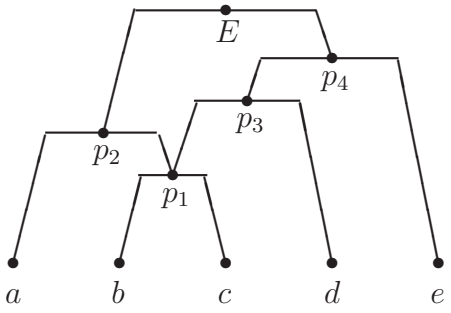
A seguir são apresentados alguns resultados importantes.

- Se P é uma hierarquia então $\mathbb{H}$ (conjunto das classes hierárquicas) é igual a P;

- A hierarquia trivial T, constituída pelas classes singulares e por E, está sempre incluída no conjunto das classes hierárquicas de uma pirâmide.

- Se uma classe tem dois predecessores, nenhum desses predecessores é hierárquico.

- O conjunto $\mathbb{H}$ é uma hierarquia sobre E, mas não é, em geral, a hierarquia de maior cardinal incluída em P, como mostra a Figura 4.2.



$$\begin{aligned} P &= T \cup \{p_1, p_2, p_3, p_4\} \\ \mathbb{H} &= T \cup \{p_1\} \\ \mathbb{H}_1 &= T \cup \{p_1, p_3, p_4\} \end{aligned}$$

Figura 4.2: Classes Hierárquicas.

$\mathbb{H}_1$ é uma hierarquia incluída em P que tem cardinal superior ao de $\mathbb{H}$.

O número de ordens compatíveis com uma pirâmide está relacionado com o número de classes hierárquicas. Assim,

Proposição 4.1 *Seja P uma pirâmide binária sobre E, que não é incompleta ($E \in P$). O número de ordens sobre E compatíveis com P são:*

$$2^{\textit{número de classes hierárquicas de cardinal superior a 1}} \tag{4.1}$$

A demonstração pode ser vista em Bertrand [4] (cap 3, prop 3.2).

De referir ainda que, numa pirâmide P, se $\mathbb{H} = T$ então exitem 2 ordens compatíveis com P. Na Figua 2.2 do Capítulo 2, tem-se que $\mathbb{H} = T \cup \{p_2\}$. Viu-se que existiam $4 = 2^2$ ordens compatíveis, que resultam de existirem duas classes hierárquicas de cardinal superior a 1: p_2 e $p_6 = E$.

Classes Maximais Hierárquicas

Diz-se que $p \in P$ é uma *classe maximal hierárquica*[1] se:

- p é uma classe hierárquica;
- p não admite predecessor hierárquico.

Designa-se por $\mathbb{S}$ o conjunto das classes maximais hierárquicas de P.

Por exemplo, na Figura 4.1, p_1 e p_{14} são classes hierárquicas, sendo p_{14} uma classe maximal hierárquica.

4.2.7 Fronteira de uma Pirâmide Incompleta e Classes Livres

A *fronteira* de uma pirâmide incompleta é o conjunto das classes periféricas da pirâmide, no sentido literal do termo. Esta noção foi introduzida por Bertrand [4].

Seja P uma pirâmide incompleta. A *fronteira* $F(P)$ é igual a:

$$F(P) = \{p \in P : \ p \text{ é maximal para a relação "no interior de" e } p \text{ admite, no máximo, um predecessor}\}$$

Por exemplo, na Figura 4.1, a fronteira da pirâmide P é:

$$F(P) = \left\{ \{e\}, p_1, p_{11}, p_{14}, p_{15}, p_{12}, p_9, p_3, p_{10}, p_4, \{b\}, \{c\}, p_2, p_{13}, p_7, \{d\} \right\}$$

Para ***classe livre***, Bertrand [4] propôs a definição seguinte:

"*h é uma classe livre se: h é hierárquico e $s(h) \in F(P)$, onde $s(h)$ é a classe maximal hierárquica de h (no sentido de inclusão).*"

Bertrand [4] enunciou também o seguinte resultado:

Proposição 4.2 *Seja h uma classe de P, tem-se que*

(i) $s(h) \in F(P) \iff s(h)$ *é uma classe extrema;*

(ii) h *é uma classe extrema* $\implies h \in F(P)$.

[1] *sommet hierarchique*, em francês.

Na Figura 4.1, as classes $\{g\}$, $\{m\}$ e p_8 são classes livres. Deve-se referir que a classe $\{g\}$ é hierárquica e $s\big(\{g\}\big) = p_{14} \in F(P)$, bem como $\{m\}$ é classe hierárquica e $s\big(\{m\}\big) = p_{14} \in F(P)$. No entanto, p_8 **não** é uma classe hierárquica uma vez que $p_8 \cap p_5 = l \notin \{\emptyset, p_8, p_5\}$. Pela definição proposta por Bertrand [4], p_8 não seria classe livre. No entanto, se $\{m\}$ é livre, ao inverter a ordem dos elementos da classe p_{14}, a classe p_8 será também extrema. Deve-se portanto considerar uma definição de classe livre que contemple estas classes.

Vai-se então considerar como ***classe livre*** a definição dada por Mfoumoune [27]:

"Diz-se que uma classe é livre se e só se:

(i) *p tem um só predecessor;*

(ii) *todas as classes ascendentes de p têm, no máximo, um só predecessor;*

(iii) *todas as classes descendentes cujo menor **ou** maior elemento coincide com o da classe p têm um só predecessor;*

(iv) *seja s_p o mais pequeno ascendente extremo de p (no sentido de inclusão). Todos os descendentes de s_p com o mesmo elemento mínimo ou máximo têm um só predecessor."*

Verifica-se que p_8 é realmente uma classe livre pois:

(1) tem um só predecessor p_{14};

(2) todos os ascendentes de p_8, $A(p_8) = \{p_{14}, p_{15}\}$, têm cada um, no máximo, um só predecessor (p_{15} nem tem);

(3) o único descendente cujo elemento máximo coincide com o de p_8 é $\{m\}$, que tem um só predecessor. O mesmo não se verifica para o único descendente cujo elemento mínimo coincide com o de p_8, $\{l\}$, que tem dois predecessores. Mas, para satisfazer a condição (iii) é suficiente que seja satisfeito apenas um;

(4) $s_{p_8} = p_{14}$, isto é, p_{14} é o menor ascendente extremo de p_8. Todos os descendentes de p_{14} com o mesmo elemento mínimo ou máximo, seja $\big\{\{e\}, p_1, p_{11}, p_8, \{m\}\big\}$, têm um só predecessor.

Por exemplo, na Figura 4.1, as classes $\{l\}$, $\{i\}$ e $\{f\}$ não são classes livres. Com efeito:

- a classe $\{l\}$ não é livre pois tem dois predecessores;
- a classe $\{i\}$ não é livre pois a classe ascendente p_6 tem dois predecessores;
- a classe $\{f\}$ não é livre pois $s_{\{f\}} = p_{11}$ e $\{l\}$, que é um descendente de p_{11}, tem dois predecessores.

A existência de classes livres leva a considerar estas últimas como fazendo parte da fronteira, pelo que se passa a considerar a fronteira $F'(P)$ da pirâmide incompleta P da seguinte forma:

$$F'(P) = F(P) \cup \{\text{classes livres}\}$$

Assim, na Figura 4.1, uma vez que as classes $\{g\}$, $\{m\}$ e p_8 são livres tem-se a fronteira:

$$F'(P) = \{\{e\}, p_1, p_{11}, p_{14}, p_{15}, p_{12}, p_9, p_3, p_{10}, p_4, \{b\}, \{c\}, p_2, p_{13}, p_7, \{d\}, \{g\}, \{m\}, p_8\}$$

4.2.8 Vale

Seja $B(p)$ o conjunto das classes com o mesmo elemento mínimo ou (exclusivamente) máximo que p e que não tenham ascendente possuindo dois predecessores.

Designa-se *vale* de duas classes p e q o conjunto $B(p) \cup B(q)$ tal que $\forall\, x \in B(p)$ e $\forall\, y \in B(q)$ se tem que $x \cap y = p \cap q$. Nenhum ascendente de $p \cap q$ possui mais do que um predecessor. $V(p, q)$ representa o vale das classes p e q.

Na identificação do vale de duas classes p e q dois casos distintos podem ocorrer:

- se as classes p e q não pertencem à mesma componente conexa tem-se que, por exemplo (Figura 4.1), $V(p_4, \{c\}) = \{\{b\}, \{c\}, p_2, p_4, p_{10}, p_{13}\}$;
- se as classes p e q pertencem à mesma componente conexa tem-se, no mesmo exemplo (Figura 4.1), $V(p_3, p_9) = \{p_3, p_9, p_{10}, p_{12}\}$.

Como se vê esta noção de vale entre duas classes pode ser estendida a classes extremas que pertençam a componentes conexas distintas. No entanto, será normalmente referida a classes de uma mesma componente conexa situadas entre duas classes maximais consecutivas, no sentido da ordem θ.

4.2.9 Fecho de uma Classe

Diz-se que uma classe p é um *fecho* de uma classe q se é a maior classe, no sentido de inclusão, onde o menor elemento ou o maior elemento de p coincide com o da classe q.

É o mesmo que dizer:

$$\forall\, p' \quad \Big(\big(min(p') = min(q)\big) \quad \text{ou} \quad \big(max(p') = max(q)\big) \Big) \quad \Longrightarrow \quad p' \subseteq p$$

Um fecho p de uma classe q é dito *fecho à direita* ou *fecho pelo elemento inferior* se $min\,(p) = min\,(q)$.

Um fecho p de uma classe q é dito *fecho à esquerda* ou *fecho pelo elemento superior* se $max\,(p) = max\,(q)$.

Por exemplo, na Figura 4.1, o conjunto dos fechos de $\{l\}$ é $\{p_{11}, p_8\}$, sendo p_{11} o fecho à esquerda e p_8 o fecho à direita.

Em Mfoumoune [27] são enunciadas e demonstradas duas propriedades importantes:

- Toda a classe tem, no máximo, dois fechos;
- Todos os fechos têm, no máximo, um predecessor.

4.3 Algoritmos de C.P.A.

4.3.1 Diday (1984)

Diday [12] e [13] introduziu as pirâmides e propôs um princípio fundamental para a sua construção. Analogamente ao algoritmo de C.H.A., Diday propôs um algoritmo de C.P.A. com as etapas seguintes:

(a) Cada elemento de E é chamado de *grupo*;

(b) São agregados os dois *grupos* mais próximos entre os *grupos* que ainda não foram agregados duas vezes;

(c) Começa-se novamente em (b) até o *grupo* que contém E ser formado;

Para evitar cruzamentos é necessário acrescentar as seguintes condições:

(d) Cada vez que um *grupo* é formado, deve-se associar uma ordem sobre esses *grupos* que são reunidos;

(e) Dois *grupos* não podem ser agregados se a sua união não é conexa;

(f) Sejam i e j elementos extremos de uma parte conexa de E associada a uma classe h. Nenhum *grupo* pode ser ligado a um *grupo* incluído em h que não contenha nem i nem j.

Este algoritmo contrói uma pirâmide (Diday [13]). Como se vê, as etapas (e) e (f) do algoritmo mais a restrição segundo a qual as classes não podem ser agregadas duas vezes constituem as condições de agregação das classes. Estas, definidas aqui sumariamente, são formalizadas por Bertrand [4].

Algoritmo dos *"vizinhos recíprocos"*

Também proposto por Bertrand [4], substitui no algoritmo anterior a etapa (b) pela etapa (b') seguinte:

(b') São agregados os *grupos* que são *vizinhos recíprocos* entre os *grupos* que não foram agregados duas vezes.

Dois *grupos* são *vizinhos recíprocos* quando são reciprocamente mais próximos um do outro no sentido da dissemelhança. Formalmente, duas classes p e q são *vizinhos recíprocos* se (Bertrand [4])

$$\begin{aligned} \Gamma(p,q) &= min\{\Gamma(p,x) : x \text{ é agregável com } p\} \\ &= min\{\Gamma(q,x) : x \text{ é agregável com } q\} \end{aligned}$$

Comentário

O algoritmo de Diday serve de ponto de partida a todos os outros algoritmos do tipo ascendente aplicados à Classificação Piramidal. No entanto, nota-se que o conjunto das classes agregáveis não é claramente definido, confundindo-se com o conjunto das classes já formadas. O procedimento de procura de pares de classes a agregar explora todas as classes da pirâmide e testa aquelas que ainda não foram agregadas duas vezes verificando as condições (e) e (f). A complexidade algorítmica é pesada pelo número de classes criadas e investigadas. Para atenuar este peso, Bertrand [4] propôs o algoritmo dos *vizinhos recíprocos*, que permite acelerar a contrução da pirâmide uma vez que tem em conta a posição física das classes.

A seguir é apresentado o algoritmo proposto por Bertrand para a C.P.A..

4.3.2 Bertrand (1986)

Bertrand [4] propôs um método de classificação piramidal que trata as matrizes de dissemelhança. Propôs também a extensão da fórmula de Lance-William-Jambu ao caso das pirâmides para os índices de agregação usualmente utilizados. Esta fórmula permite o cálculo da dissemelhança entre uma classe e uma outra, formada numa etapa anterior pela reunião de duas classes.

Bertrand formaliza as condições de agregação das classes. Estas condições apoiam-se nas posições fisícas dos elementos das classes. O conjunto de classes agregáveis é a fronteira da pirâmide em formação, notado $F(P)$. O algoritmo segue o princípio fundamental da C.P.A..

Algoritmo

Passo 1 (Inicialização)

classes: classes singulares de E.

classes agregáveis: todas as classes são agregáveis.

componentes conexas: cada componente conexa reduz-se a uma classe singular.

ordem compatível: fixa-se uma ordem compatível θ, que é, por exemplo, a ordem implícita sobre E.

Passo 2 (Condições de Agregação)

Agrega-se um par de classes (p, q)[2] se a dissemelhança que separa p e q é mínima entre o conjunto de pares de classes que verificam:

- Se p e q pertencem à mesma componente conexa, notando S_j a classe maximal que contem p, tem-se:
  - p está à direita de S_j e estritamente à esquerda de $S_j \cap S_{j+1}$.
  - q está à esquerda de S_{j+1} e estritamente à direita de $S_j \cap S_{j+1}$.
- Se p e q **não** pertencem à mesma componente conexa, então:
  - p está à direita de S_k ou à esquerda de S_1.
  - q está à direita de S'_l ou à esquerda de S'_1.[3]
  - S_1 está antes de S'_1.

Passo 3 (Actualização da Ordem)

Se duas classes p e q que venham a ser agregadas não pertencem à mesma componente conexa então colocam-se os elementos de $C(q)$ consecutivamente depois dos da $C(p)$. Mais,

- se $min(p) = min\big(C(p)\big)$, inverte-se a ordem dos elementos de $C(p)$.
- se $max(q) = max\big(C(q)\big)$, inverte-se a ordem dos elementos de $C(q)$.

[2] p à esquerda de q.

[3] S'_i com $i = 1, \ldots, l$ representa as classes maximais da componente conexa da classe q.

Passo 4 (Teste de Paragem)

Se a classe formada $p \cup q$ não é igual a E recomeça-se no **Passo 2**, caso contrário termina-se o algoritmo.

Comentário

A diferença entre o algoritmo proposto por Diday e este de Bertrand situa-se a dois níveis. Primeiro, Bertrand restringe a procura do par de classes a agregar às classes da fronteira $F(P)$. Esta restrição contribui para melhorar a complexidade do procedimento de procura do par de classes a agregar, isto é, não se explora mais todos os pares de classes da pirâmide em construção, mas todos os pares de um conjunto mais reduzido. Segundo, ele formaliza matematicamente as condições de agregação de classes, "esquecidas" por Diday, utilizando o aspecto topológico das classes, que é o mesmo que dizer a posição física das classes umas relativamente às outras.

No entanto, Bertrand não tem em conta as classes livres na construção de uma pirâmide. Assim a ordem θ é obtida de forma arbitrária. No algoritmo que se segue, Mfoumoune [27] já contempla como fazendo parte da fronteira, $F'(P)$, as classes livres.

4.3.3 Mfoumoune (1998) — Algoritmo *QuikCAP*

Mfoumoune [27] propôs uma nova implementação do algoritmo de C.P.A. que integra dados numéricos e simbólicos (Brito [9]) e utiliza uma metodologia ligeiramente diferente, fundada sobre a Selecção – Agregação – Eliminação. Mostra que desta maneira consegue-se, por um lado, diminuir a complexidade algorítmica e, por outro lado, obter resultados mais satisfatórios.

A ideia fundamental do algoritmo está assente no Postulado:

"*Em cada agregação de classes, alguns pares não são agregáveis devido às suas posições fisícas e devem ser eleminados da lista de pares de classes potencialmente agregáveis para iterações futuras.*"

Este Postulado pressupõe que sejam conhecidos todos os pares de classes agregáveis, no sentido das condições de agregação em cada etapa da construção da pirâmide. Neste sentido, Mfoumoune propõe:

- a identificação integral de todos os pares de classes agregáveis, uma vez que sendo possível identificar topologicamente os pares de classes agregáveis torna-se desnecessário verificar as condições de agregação;

- a gestão eficaz e rigorosa dos pares de classes agregáveis que visa acelerar o procedimento de procura do par de classes a agregar em cada etapa da construção ascendente da pirâmide. Há assim necessidade de eliminar todos os pares de classes que se tornam não agregáveis à medida que se formam novas classes e ainda deduzir as classes agregáveis com a nova classe formada.

Identificação dos pares de classes agregáveis

A identificação formal de todos os pares de classes agregáveis conduziu à definição da noção de *acessibilidade mútua* entre classes. Esta noção permite, por um lado, identificar os pares de classes agregáveis e, por outro, limitar a procura do par de classes de dissemelhança mínima aos pares *mutuamente acessíveis*.

Seja P uma pirâmide incompleta com duas componentes conexas e p, q e x três classes de P. O exemplo da Figura 4.3 permitirá ilustrar esta definição e ajudar na compreensão das regras de identificação das *acessibilidades* de uma classe.

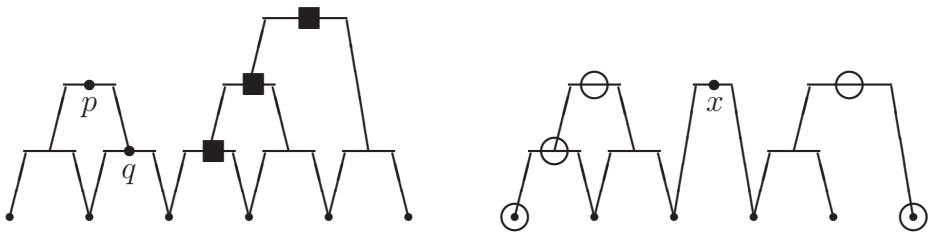


Figura 4.3: Ilustração de classes *mutuamente acessíveis*.

As acessibilidades de p são todas as classes marcadas por ■ e ◯, enquanto que as acessibilidades de q são apenas as classes marcadas por ■.

Para a identificação das *acessibilidades*, Mfoumoune considerou as regras seguintes:

(i) *a classe é interior a um vale*: as suas *acessibilidades* são todas as classes do flanco oposto, como se pode visualizar pela classe q da Figura 4.3;

(ii) *a classe é maximal não extrema*: as suas *acessibilidades* são as classes dos flancos opostos dos seus dois vales. Observe-se a classe x da Figura 4.3;

(iii) *a classe é livre ou extrema*: as suas *acessibilidades* são as classes do flanco oposto, se ele existe, mais as classes extremas ou livres de outras componentes conexas. A classe p da Figura 4.3 permite ilustrar esta situação.

Mfoumoune mostrou que existe uma bijecção entre o conjunto dos pares de classes *mutuamente acessíveis* e o conjunto das classes que verificam as condições de agregação.

Eliminação das classes não agregáveis

Depois da agregação de um par de classes, certos pares tornam-se não agregáveis para iterações futuras e devem, consequentemente, ser eliminados a fim de iniciar uma nova iteração.

Na inicialização, todos os pares de classes $\big((x,y)\,,\ x \neq y\big)$, neste caso singulares, são *mutuamente acessíveis*.

Suponha-se agora a inicializar uma iteração qualquer. Seja (p,q) o par de classes a agregar na iteração em curso. Depois da agregação de p e q, os pares que se tornam *mutuamente* **não** *acessíveis*, isto é, não agregáveis no sentido das condições de agregação, para as próximas iterações, são:

1º Caso: p e q pertencem à mesma componente conexa.

Eliminam-se todos os pares de classes (x,y) de flancos opostos de $V(p,q)$, vale das classes p e q, antes da agregação.

Na Figura 4.4 pode visualizar-se a agregação de duas classes pertencentes à mesma componente conexa, $p_{10} = p_3 \cup p_6$.

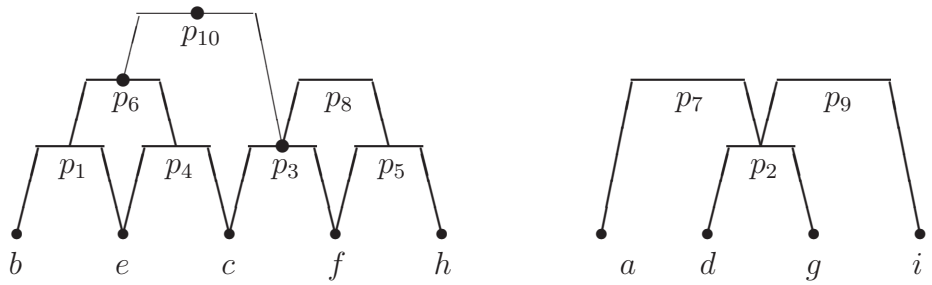


Figura 4.4: Agregação de classes na mesma componente conexa.

A formação da classe p_{10} leva à eliminação dos pares de classes (p_4, p_3), (p_4, p_8), (p_6, p_3) e (p_6, p_8).

2º Caso: p e q **não** pertencem à mesma componente conexa.

Sejam $C(p)$, $C(q)$ e $C(p \cup q)$ as componentes conexas de p, q e $p \cup q$, respectivamente. Suponha-se que p está *antes de* q na $C(p \cup q)$ e considere-se a Figura 4.5 que ilustra a agregação de duas classes de componentes conexas distintas, $p_{10} = p_3 \cup p_6$.

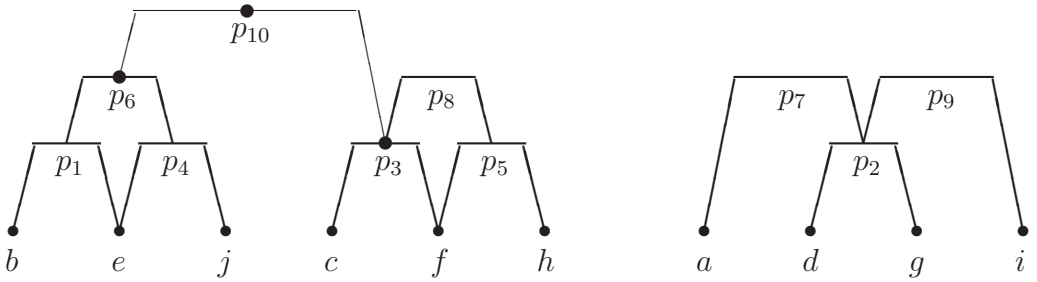


Figura 4.5: Agregação de classes de componentes conexas distintas.

Nos pares de classes a eliminar vai-se distinguir dois casos:

(i) *após a agregação, os pares de classes* (x, y) *a eliminar pertencem à mesma componente conexa.*

Eliminam-se os pares de classes (x, y) tais que x é uma classe extrema ou livre de $C(p)$, antes da agregação, e y é uma classe extrema ou livre $C(q)$, antes da agregação.

Por exemplo, na Figura 4.5, eliminam-se os pares de classes seguintes: $(\{b\}, \{c\})$, $(\{b\}, p_3)$, $(\{b\}, p_8)$, $(\{b\}, p_5)$, $(\{b\}, \{h\})$, $(p_1, \{c\})$, (p_1, p_3), (p_1, p_8), (p_1, p_5), $(p_1, \{h\})$, $(p_6, \{c\})$, (p_6, p_3), (p_6, p_8), (p_6, p_5), $(p_6, \{h\})$, $(p_4, \{c\})$, (p_4, p_3), (p_4, p_8), (p_4, p_5), $(p_4, \{h\})$, $(\{j\}, \{c\})$, $(\{j\}, p_3)$, $(\{j\}, p_8)$, $(\{j\}, p_5)$ e $(\{j\}, \{h\})$, porque estão agora na mesma componente conexa, $C(p_{10}) = C(p_3 \cup p_6)$.

(ii) *após a agregação, os pares de classes* (x, y) *a eliminar pertencem a componentes conexas distintas.*

(a) *Se* p *não é uma classe maximal ou se* $min(p) \neq min\big(C(p \cup q)\big)$ *, então seja* S_p *a classe maximal de* p *antes da agregação. Agora, se* $min(S_p) = min\big(C(p \cup q)\big)$ então faz-se $r = S_p$, senão $r = \emptyset$.

Eliminam-se todos os pares de classes (x, y) tais que x é uma classe extrema ou livre de $C(p)$ para o maior elemento antes da agregação e $x \neq r$, e y é uma classe extrema ou livre de toda a componente conexa distinta de $C(p \cup q)$.

Na Figura 4.5, $p = p_6$ é maximal e $min\,(p_6) = b = min\,(p_{10})$, pelo que não é satisfeita a condição (iia). Não há portanto lugar à eliminação de pares de classes. Ao mesmo tempo, as classes $\{j\}$ e p_4 são livres e mantêm-se agregáveis com as classes $\{a\}$, p_7, p_9 e $\{i\}$ que são classes extremas da outra componente conexa. Notar que é possível inverter os elementos da classe p_6 tornando as classes $\{j\}$ e p_4 extremas.

(b) Se q não é uma classe maximal ou se $max\,(q) \neq max\big(C\,(p \cup q)\,\big)$, então seja S_q a classe maximal de q antes da agregação. Agora, se $max\,(S_q) = max\big(C\,(p \cup q)\,\big)$ então faz-se $r = S_q$, senão $r = \emptyset$.

Eliminam-se todos os pares de classes (x, y) tais que x é uma classe extrema ou livre de $C\,(q)$ para o menor elemento antes da agregação e $x \neq r$, e y é uma classe extrema ou livre de toda a componente conexa distinta de $C\,(p \cup q)$.

No exemplo da Figura 4.5, $q = p_3$ satisfaz a condição (iib), donde se eliminam os pares de classes seguintes: $(\{c\}, \{a\})$, $(\{c\}, p_7)$, $(\{c\}, p_9)$, $(\{c\}, \{i\})$, $(p_3, \{a\})$, (p_3, p_7), (p_3, p_9) e $(p_3, \{i\})$, pois $\{c\}$ e p_3 não se tornam em classes extremas, nem livres. Aqui, $r = p_8$ é uma classe ainda extrema, pelo que não se eliminam os pares de classes que ela forma com as suas *acessibilidades* de outras componentes conexas.

Dedução das *acessibilidades* da nova classe formada

Depois da agregação, as *acessibilidades* da nova classe $p \cup q$ devem ser inseridas numa estrutura que contém todos os pares de classes potencialmente agregáveis nas iterações seguintes.

As *acessibilidades* da nova classe $p \cup q$ são:

(i) *todas as classes de flancos opostos dos seus vales, que são no máximo dois;*

(ii) *quando $p \cup q$ é extrema e $C\,(p \cup q)$ não é a única componente conexa, todas as classes extremas ou livres de outras componentes conexas.*

Por exemplo, na Figura 4.5, a classe p_{10} tem como *acessibilidades* o conjunto $\big\{p_8, \{a\}, p_7, p_9, \{i\}\big\}$.

O número de *acessibilidades* de uma classe varia segundo as diferentes fases de construção de uma pirâmide. Na fase de construção da ordem onde há mais componentes conexas, fase que Mfoumoune designou de *seriação*, as classes extremas têm um número considerável de *acessibilidades*. Quando se tem apenas uma componente conexa, na fase dita de *classificação*, esta quantidade é reduzida para as classes de flancos opostos de um mesmo vale.

Mfoumoune criou um modelo bilinear para estruturar a lista de pares de classes agregáveis. O interesse desta estrutura é, por um lado, acelerar as operações de eliminação dos pares de classes; e, por outro, acelerar a procura do par de classes de dissemelhança mínima. O par de classes com menor dissemelhança está no topo desta lista.

Algoritmo *QuikCAP*

O algoritmo segue o princípio fundamental da C.P.A..

- Cada elemento de E forma uma classe, à qual é associado um número inteiro de 1 a n.
- Cada elemento de E forma uma componente conexa.
- M_D é a matriz de dissemelhanças sobre $E \times E$.
- M_C é uma lista contendo todos os $\frac{n(n-1)}{2}$ pares agregáveis de M_D.

  Se $M_C \neq \emptyset$ **fazer**

  - Seleccionar o par de classes que está no topo de M_C, seja (p_1, p_2).
  - Agregar as classes, seja $p = p_1 \cup p_2$.
  - Eliminar de M_C o par (p_1, p_2) e todos os pares que não são mais agregáveis.
  - Se as classes agregadas pertencem a componentes conexas diferentes, então reunir as suas componentes conexas, ordenando-as de maneira a respeitar a estrutura piramidal.
  - Deduzir as novas classes agregáveis com p.

  Fim

Comentário

O algoritmo proposto por Mfoumoune contempla já as classes livres como fazendo parte da fronteira e, por conseguinte, estas não são excluídas na construção de uma pirâmide. Desta forma, o algoritmo tem em conta não só a inversão total, mas também a inversão parcial de uma componente conexa. Este assunto será abordado na Secção 4.4.

Para além disso, o algoritmo permite uma gestão mais eficaz dos pares de classes agregáveis, acelerando o processo de procura do par de classes a agregar em cada iteração. A complexidade algorítmica do *QuikCAP* é mais reduzida.

4.4 Inversão de Componentes Conexas

A construção de uma pirâmide implica a resolução de numerosos problemas algorítmicos, em particular, os algoritmos de inversão parcial ou total de componentes conexas. Estas inversões estão associadas à fase, dita de *seriação* (Mfoumoune), onde as agregações entre classes pertencentes a componentes conexas distintas permitem estabelecer a ordem compatível sobre os elementos.

Com efeito, a agregação de duas classes ainda não agregadas atribui uma ordem arbitrária entre os elementos. Esta ordem pode ser revista então numa segunda agregação de uma das classes ou de um "*parente*", ascendente ou descendente, desta. A revisão desta ordem consiste em inverter parcial ou globalmente uma ou outra ou as duas componentes conexas de uma pirâmide em construção.

Na Figura 4.6 pode visualizar-se uma pirâmide incompleta formada por duas componentes conexas, onde estão assinaladas com ◯ as classes a agregar, $\{a\}$ e p_4.

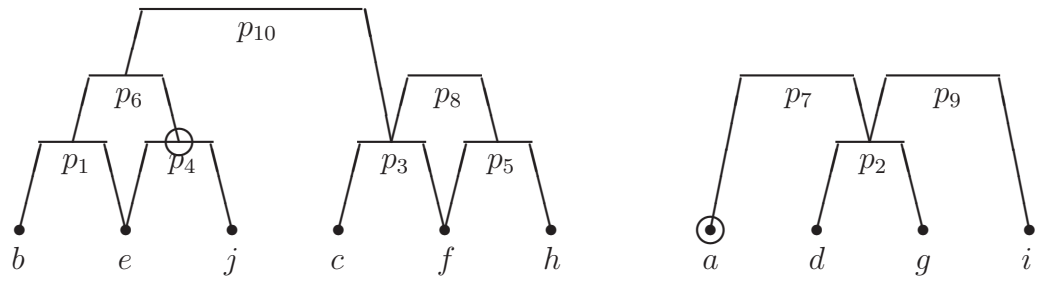


Figura 4.6: Pirâmide incompleta formada por duas componentes conexas.

A agregação das classes $\{a\}$ e p_4 dá origem à formação da classe p_{11}. Para tal é necessário inverter a ordem dos elementos de $C\big(\{a\}\big) = C_{\{a,d,g,i\}}$ e colocá-los antes de $C\left(p_4\right) = C_{\{b,e,j,c,f,h\}}$. É ainda preciso inverter a ordem dos elementos da classe hierárquica p_6 para permitir que a classe p_4 seja extrema. As componentes conexas são unificadas e a sua representação encontra-se na Figura 4.7.

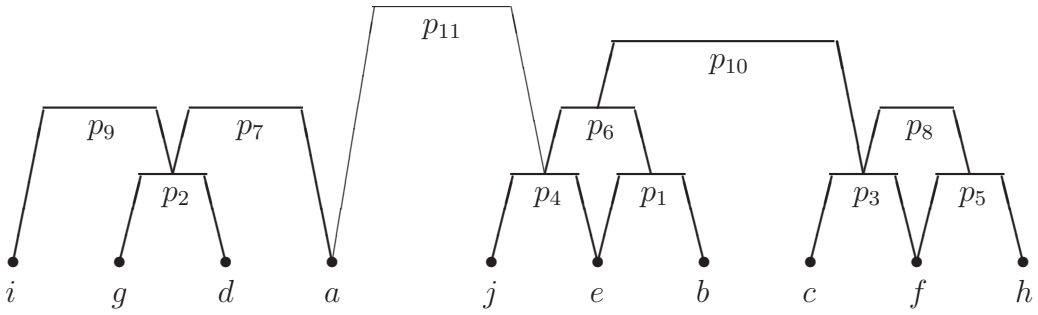


Figura 4.7: Agregação das classes $\{a\}$ e p_4 e fusão das componentes conexas da Figura 4.6.

Mfoumoune propôs o seguinte algoritmo informal para a inversão parcial de uma componente conexa:

"*De uma maneira geral, a inversão de uma componente conexa relativamente a uma classe livre consiste em inverter a classe associada ao mais pequeno ascendente (no sentido de inclusão) extremo dessa classe, de forma a que o elemento mínimo ou máximo dessa classe livre coincida com o da componente conexa.*"

4.5 Considerações Finais

4.5.1 Algoritmos alternativos

Brito (1991)

O algoritmo proposto por Brito [9] representa uma aproximação simbólica da C.P.A.. As pirâmides simbólicas são uma estrutura herdada do conjunto de objectos simbólicos, introduzidos por Diday. De uma forma geral, define-se um objecto simbólico como uma *descrição* que se exprime sob a forma de uma conjunção de acontecimentos sobre os valores do domínio das variáveis. No fundo, a aproximação simbólica extende a problemática da análise de dados a objectos com propriedades

mais complexas, difíceis de exprimir nas tabelas de dados clássicas. Essa extensão situa-se a três níveis: ao nível dos dados, cada objecto é representado por uma *descrição*; ao nível da caracterização das classes, cada classe é munida de uma *intensão* (a sua função característica) e de uma *extensão* (os objectos que a compõem); e ao nível do critério de agregação ou de divisão.

Brito generaliza as condições de agregação "esquecidas" por Diday e posteriormente formalizadas por Bertrand. Numa primeira fase, considera que o objectivo é a construção de uma pirâmide binária, para depois admitir a formação de classes tendo mais de dois sucessores, donde resultam as pirâmides não-binárias. Em cada iteração não se procuram n-pletos de classes, antes reagrupam-se numa mesma classe as classes que pertençam à *extensão* de uma mesma *descrição*, de maneira a ser possível conservar a esrutura piramidal. O algoritmo não é aqui apresentado uma vez que exigia a definição de algumas noções e, para além disso, este trabalho centra-se no estudo de dados numéricos.

As condições de agregação que Brito generaliza, adaptadas da aproximação numérica, não integram totalmente as restrições aplicadas a classes pertencentes à mesma componenete conexa. Com efeito, enquanto que duas classes pertencentes à mesma componente conexa para serem agregadas deveriam ser de flancos opostos de um mesmo vale, na aproximação simbólica tal já não se verifica. Considere-se a Figura 4.8 para ilustrar a agregação na aproximação simbólica.

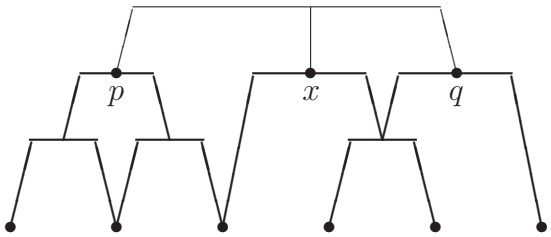


Figura 4.8: Ilusração da agregação de duas classes na aproximação simbólica.

Os pares de classes (p, x) e (q, x) são agregáveis segundo a aproximação numérica. O par (p, q) não é, contudo é potencialmente agregável na aproximação simbólica pois $p \cup q$ forma um intervalo. Neste caso, liga-se fisicamente x a $p \cup q$.

Durand (1989)

Duand [15] trabalhou sobre o melhoramento da Classificação Piramidal numérica. Interessou-se pela obtenção da maior dissemelhança de Robinson entre todas

aquelas que são menores que a dissemelhança inicial.

Existe uma bijecção entre uma pseudo-hierarquia (ou pirâmide) e a dissemelhança de Robinson. É equivalente, portanto, procurar uma pseudo-hierarquia indiciada óptima ou uma dissemelhança de Robinson óptima ou então uma ordem compatível óptima. O critério de optimização apresentado por Durand baseia-se na determinação da maior dissemelhança de Robinson inferior à dissemelhança inicial.

4.5.2 Resumo

Este capítulo centrou-se no aprofundamento do estudo da Classificação Piramidal. Foram introduzidas algumas noções fundamentais à compreensão e implementação dos algoritmos de C.P.A. e essenciais para a posterior extensão aos algoritmos de geração aleatória de pirâmides.

Foram apresentados e analisados os algoritmos de C.P.A. propostos por Diday [12], Bertrand [4] e Mfoumoune[27]. No final foi ainda referida a aproximação simbólica de Brito [9] na C.P.A.. Os algoritmos de Diday, Bertrand e Brito seguem uma metodologia baseada na Selecção – Agregação – Actualização da ordem. Esta concepção da C.P.A. trás alguns problemas de complexidade algorítmica. Outra restrição destes algoritmos é o facto de não considerarem as classes livres na fronteira da pirâmide incompleta. Mfoumoune propõe um método, *QuikCAP*, que assenta na Selecção – Agregação – Eliminação, conseguindo desta forma diminuir a complexidade algorítmica. Para além disso, o algoritmo *QuikCAP* contempla já as classes livres na fronteira da pirâmide em construção.

Capítulo 5

Geração Aleatória de Pirâmides

5.1 Introdução

As pirâmides são uma estrutura classificatória ainda pouco desenvolvida na literatura. Dependendo da escolha da função de comparação entre elementos e do critério de agregação entre classes, diferentes pirâmides podem ser obtidas. Também diferentes algoritmos de *Classificação Piramidal Ascendente* podem conduzir a pirâmides distintas. A geração aleatória de pirâmides surge então como uma ferramenta importante no estudo das estruturas piramidais, podendo permitir um estudo comparativo do desempenho dos diferentes algoritmos de C.P.A..

Outro aspecto importante na qual a geração aleatória de pirâmides poderá dar um grande contributo é o facto de não existir nenhuma fórmula que permita determinar o número de pirâmides binárias distintas com n nós terminais, nem os tipos topológicos associados. Será apresentado um método de geração aleatória de pirâmides com n nós terminais, com n fixo mas sem limitação superior. Contudo, convém referir que outros métodos podem ser desenvolvidos e, no final deste capítulo, será feita uma abordagem a um deles.

O método proposto — ***R**andom Generation **A**lgorithm of **P**yramids* (***RAP***) — actua de modo muito semelhante ao algoritmo de C.P.A. proposto por Bertrand [4], em que o par de classes a agregar é gerado aleatoriamente. É, no fundo, uma extensão, para as pirâmides, do método *RA* proposto por Podani [32] e referido no Capítulo 3. O algoritmo foi implementado em linguagem Matlab e encontra-se no Anexo 7.

Neste capítulo será apresentada a filosofia base do método e algumas propriedades necessárias à implementação do mesmo. O algoritmo será depois apresentado de forma sucinta e acompanhado de uma aplicação prática para uma melhor compreensão.

5.2 Filosofia Base do Método

Seja E o conjunto dos elementos a classificar que serão identificados no algoritmo pelos inteiros de 1 até n.

As classes singulares são as *classes activas* (Proposição 5.1) para a primeira agregação, isto é, as classes que se encontram disponíveis para agregar. Inicialmente existem $\binom{n}{2}$ pares possíveis de classes a agregar. O método, que é um método iterativo, termina quando não existirem mais classes activas, ou seja, quando é formada a classe E.

No seu processo iterativo o método actua em três fases:

(1) Geração aleatória do par de classes a agregar. Para a obtenção deste par é inicialmente gerada uma classe do conjunto das *classes activas*. Segue-se a definição do conjunto das classes agregáveis com esta, satisfazendo as *condições de agregação* (Proposição 5.2). Por fim, deste conjunto de classes agregáveis é gerada aleatoriamente a segunda classe a agregar.

(2) Agregação das classes geradas. Nesta fase são actualizadas a ordem sobre os objectos, a pirâmide em construção e a matriz de Robinson.

(3) Definição do conjunto das *classes activas* para a próxima iteração.

Na secção que se segue serão detalhadas e explicadas algumas propriedades importantes para a implementação do método proposto.

5.3 Propriedades Necessárias à Implementação do Algoritmo

A explicação de algumas propriedades serão acompanhadas de um exemplo. Seja P uma pirâmide incompleta e considere-se a Figura 5.1.

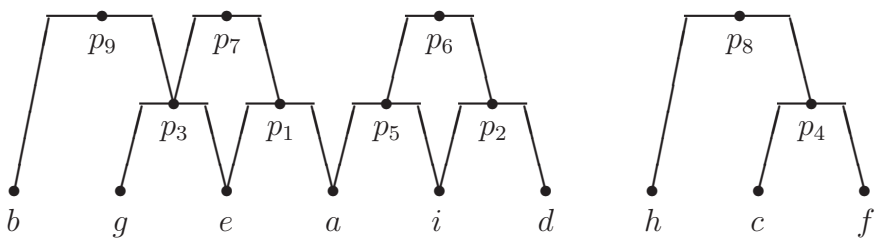


Figura 5.1: Pirâmide incompleta para ilustração de propriedades.

Na primeira iteração do algoritmo todas as classes singulares são *classes activas*. Para as iterações seguintes serão consideradas *classes activas* as classes que satisfazem as proriedades que se seguem.

Proposição 5.1 (Classes Activas) *Uma classe p de P é denominada activa se as três condições seguintes são satisfeitas:*

- *a classe p foi agregada no máximo uma vez;*
- *não existe $p' \in P$ tal que p seja classe interior de p';*
- *existe outra classe p'' de P tal que p é agregável com p''.*

Na Figura 5.1 são classes activas todas as classes extremas, $\{b\}$, p_9, p_6, p_2, $\{d\}$, $\{h\}$, p_8, p_4 e $\{f\}$, as classes maximais não extremas, p_7, e as classes interiores a um vale, p_1 e p_5. Refira-se que a classe $\{c\}$ não é activa pois o algoritmo proposto não contempla as classes livres. Esta classe seria extrema invertendo parcialmente a componente conexa $C_{\{h,c,f\}}$, ou seja, os elementos da classe p_4. Note-se ainda que a classe $\{g\}$ não é livre.

Do conjunto das classes activas é gerada aleatoriamente uma classe, seja p. É necessário definir agora o conjunto das classes agregáveis com p, satisfazendo as condições de agregação.

Proposição 5.2 (Condições de Agregação) *Sejam p e q duas classes de P, com p à esquerda de q.*

- *Se p e q* ***não pertencem*** *à mesma componente conexa, p é agregável com q se p é extrema e q é extrema.*
- *Se p e q* ***pertencem*** *à mesma componente conexa, notando S_i a classe maximal que contém p, então p e q são agregáveis se satisfazem as seguintes condições:*
  - $p \subset S_i$, $min\,(p) < min\,(S_{i+1})$ *e* $max\,(p) = max\,(S_i)$;
  - $q \subset S_{i+1}$, $min\,(q) = min\,(S_{i+1})$ *e* $max\,(q) > max\,(S_i)$.

Na prática, gerada uma classe p, esta é agregável com q se:

- p é interior a um vale, q terá de ser uma classe do flanco oposto desse vale. Na Figura 5.1 as classes agregáveis com p_1 são as classes p_5 e p_6.

- p é uma classe maximal não extrema, q terá de ser uma classe dos flancos opostos dos seus dois vales.
  Na Figura 5.1 as classes agregáveis com p_7 são as classes p_9, p_5 e p_6.
- p é extrema maximal, q terá de ser uma classe do flanco oposto do seu vale ou uma classe extrema de outra componente conexa.
  Na Figura 5.1 as classes agregáveis com p_6 são as classes p_1, p_7, $\{h\}$, p_8, p_4 e $\{f\}$.
- p é extrema não maximal, q terá de ser uma classe extrema de outra componente conexa.
  Na Figura 5.1 as classes agregáveis com p_4 são as classes $\{b\}$, p_9, p_6, p_2 e $\{d\}$.

Sejam $C(p)$ e $C(q)$ as componentes conexas das classes p e q, respectivamente. Se as duas classes p e q a agregar pertencem à mesma componente conexa a ordem sobre os elementos mantém-se.

Se as duas classes p e q a agregar **não** pertencem à mesma componente conexa é necessário proceder à actualização da ordem compatível θ. Os elementos de $C(p)$ estão antes dos de $C(q)$. Assim,

- se $min(p) = min\big(C(p)\big)$, inverte-se a ordem dos elementos de $C(p)$;
- se $max(q) = max\big(C(q)\big)$, inverte-se a ordem dos elementos de $C(q)$;
- se $min(p) = min\big(C(p)\big)$ **e** $max(q) = max\big(C(q)\big)$, colocam-se os elementos de $C(q)$ imediatamente antes dos de $C(p)$.

No exemplo considerado da Figura 5.1, seja $p = p_6$ a primeira classe gerada. Do conjunto das classes agregáveis $\big\{p_1, p_7, \{h\}, p_8, p_4, \{f\}\big\}$, seja $q = p_4$ a segunda classe gerada. Para a agregação do par (p_6, p_4) é necessário proceder à actualização da ordem θ, isto é, inverte-se a ordem dos elementos de $C(p_4)$. Na Figura 5.2 pode visualizar-se a formação da classe p_{10}, que é a reunião do par de classes gerado.

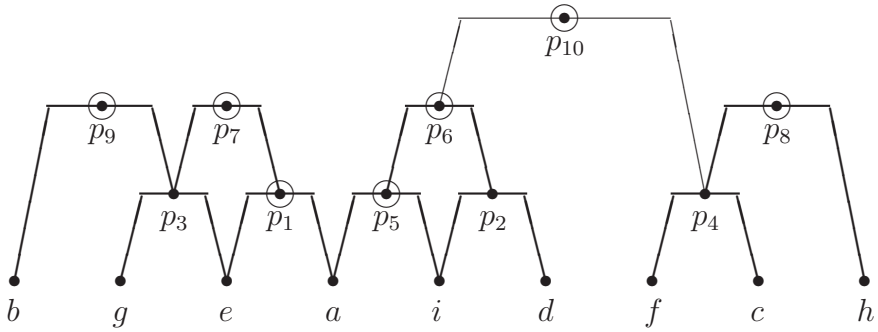


Figura 5.2: Pirâmide incompleta para ilustração de propriedades: $p_{10} = p_6 \cup p_4$.

Agregadas as duas classes procede-se à actualização da pirâmide incompleta P e ao preenchimento da matriz de Robinson. É agora necessário definir novamente quais as classes activas para a próxima iteração.

Note-se que no exemplo analisado existe agora apenas uma componente conexa pelo que o número de classes activas diminuí, assim como o número de pares de classes agregáveis. Na Figura 5.2 estão assinaladas com ◯ as classes activas para a iteração seguinte. Neste momento, as classes extremas não maximais já não são mais agregáveis. Restam então as classes maximais desta componente, p_9, p_7, p_{10} e p_8, e as classes interiores a vales, p_1, p_5 e p_6.

O algoritmo termina quando é formada a classe constituída por todos os elementos, isto é, quando o conjunto das classes activas é vazio.

5.4 Algoritmo *RAP*

O algoritmo ***R**andom Generation **A**lgorithm of **P**yramids* (***RAP***) será apresentado de forma sucinta. Começar-se-á por introduzir alguma notação utilizada na descrição do algoritmo e no final referem-se algumas limitações do método.

5.4.1 Notação Usada

- c^0_{act} – vector de classes com os n nós terminais numerados de 1 até n;
- c^i_{act} – vector de classes activas na iteração i;
- c^i_j $(j = 1, 2)$ – j-ésima classe gerada na iteração i;
- c^i_{agr} – vector de classes agregáveis com c^i_1;
- θ – n-uplo associado à ordem compatível sobre os elementos[1];
- CC – matriz associada às componentes conexas;
- P – matriz associada à pirâmide. Formada uma classe k, com $k \geq n + 1$, é acrescentada uma nova linha à matriz P. P_k representará a linha k de P, onde o primeiro elemento indica o nível de agregação e os restantes elementos, não nulos, indicam a composição da classe k;
- M – matriz $n \times n$, inicialmente com todas as entradas nulas, que vai sendo preenchida passo a passo e, no final, quando associada a ordem θ, será uma matriz de Robinson, $MR = M(\theta)$.

[1] por abuso de linguagem estamos a usar a mesma letra para a ordem e para o n-uplo associado.

5.4.2 Algoritmo — Síntese

Iteração 0: é introduzido o número de nós terminais, n, e são inicializadas as variáveis c_{act}^0, CC, θ, P e M.

Iteração i: para i de 1 até à paragem do algoritmo[2].

- geração aleatória e uniforme do par (c_1^i, c_2^i) de classes a agregar:
  - geração aleatória e uniforme de um elemento do vector c_{act}^{i-1}, classe c_1^i;
  - formação do vector de classes agregáveis, c_{agr}^i: é construído um vector com as classes que são agregáveis com c_1^i, verificando as condições de agregação da Proposição 5.2;
  - geração aleatória e uniforme de um elemento do vector c_{agr}^i, classe c_2^i.
- actualizações da ordem θ, da matriz CC, da matriz associada à pirâmide P, da matriz M e do vector das classes activas c_{act}^i.

Dados de saída: ordem θ, matriz P associada à pirâmide e matriz de Robinson $MR = M(\theta)$.

5.4.3 Comentário

O algoritmo desenvolvido gera aleatoriamente uma pirâmide com um número qualquer de nós terminais. Mas, por analogia com a geração de dendrogramas, será que o algoritmo proposto gera pirâmides uniformemente? Esta questão conduz a duas outras questões: qual o número de pirâmides não isomórficas com n nós terminais e qual o respectivo número de topologias associadas? A resposta a estas questões é complexa e não se encontra disponível na literatura. No Capítulo 6 será inicialmente determinado o número de pirâmides não isomórficas e o número de tipos topológicos das pirâmides para um número reduzido de nós terminais, $n = 3$ e $n = 4$.

Algumas limitações do algoritmo podem ser identificadas. Por um lado, as classes livres são eliminadas das classes activas por serem nós interiores. Por outro, a geração do par de classes não é obtida de forma aleatória e uniforme do conjunto de todos os pares de classes agregáveis.

Existem diferentes algoritmos de Classificação Piramidal Ascendente, pelo que o método aqui apresentado é apenas um método de geração possível. Alguns aspectos

[2] a paragem do algoritmo dá-se com a formação da classe que contém todos os elementos.

podem ser melhorados, essencialmente contemplar as classes livres na fronteira da pirâmide, isto é, mantendo-se activas para futuras agregações. Na Subsecção 5.6.1 será apresentado um método alternativo – *QuikRAP* – que, apesar de não ter sido desenvolvido e implementado, actua de forma semelhante ao algoritmo *QuikCAP* proposto por Mfoumoune [27].

5.5 Aplicação do Método *RAP*

Segue-se um pequeno exemplo, para 5 nós terminais, da aplicação do algoritmo *RAP*.

Iteração 0 Introduzido o número de nós terminais, $n = 5$, são definidas as variáveis intervenientes ao longo do algoritmo:

- o vector de classes singulares, $c^0_{act} = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \end{pmatrix}$;
- a ordem compatível θ – fixa-se, por exemplo, a ordem implícita das classes, $\theta = (1, 2, 3, 4, 5)$;
- a matriz das componentes conexas CC – cada componente conexa reduz-se a uma classe singular;
- a matriz associada à pirâmide P;
- a matriz M, que permitirá obter a matriz de Robinson final.

Na figura 5.3 é possível visualizar inicialização das matrizes CC, P e M.

$$\text{CC}=\begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 2 & 0 & 0 & 0 & 0 \\ 3 & 0 & 0 & 0 & 0 \\ 4 & 0 & 0 & 0 & 0 \\ 5 & 0 & 0 & 0 & 0 \end{pmatrix} \quad \text{P}=\left(\begin{array}{c|ccccc} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 & 0 & 0 \\ 0 & 4 & 0 & 0 & 0 & 0 \\ 0 & 5 & 0 & 0 & 0 & 0 \end{array}\right) \quad \text{M}=\begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Figura 5.3: Matrizes CC, P e M.

Iteração 1 Do vector c^0_{act} é gerada aleatória e uniformemente uma classe, $c^1_1 = 4$. As classes agregáveis com a classe 4 definem o vector $c^1_{agr} = \begin{pmatrix} 1 & 2 & 3 & 5 \end{pmatrix}$. Destas é gerada aleatória e uniformemente uma nova classe, $c^1_2 = 2$. As

classes 2 e 4 são agregadas formando uma nova classe que se denomina de 6 (ver Figura 5.4). Procede-se, de seguida, às actualizações. Na ordem sobre os elementos vem $\theta = (1, 2, 4, 3, 5)$. O número de componentes conexas reduz-se para quatro, tendo a segunda componente dois elementos, 2 e 4. Na pirâmide é acrescentada uma nova linha, $k = 6$, correspondente à nova classe gerada: $P_6 = \begin{pmatrix} 1| & 2 & 4 & 0 & 0 & 0 \end{pmatrix}$. Na matriz M vem $M(2, 4) = 1$, isto é, aos elementos das classes 2 e 4 é associado o nível de agregação 1. Procede-se à actualização das classes activas. Atendendo a que numa pirâmide cada classe admite dois predecessores, as classes 2 e 4 mantêm-se activas e a nova classe 6 é acrescentada. As classes activas são $c^1_{act} = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \end{pmatrix}$.

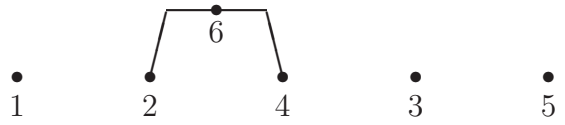


Figura 5.4: Algoritmo RAP – Iteração 1.

Iteração 2 Seja $c^2_1 = 4$ a classe gerada de c^1_{act}. As classes agregáveis com 4 são $c^2_{agr} = \begin{pmatrix} 1 & 3 & 5 \end{pmatrix}$, obtendo-se $c^2_2 = 5$. O par de classes a agregar é $(4, 5)$ formando a classe 7 (ver Figura 5.5). As actualizações procedem-se de forma análoga às da iteração 1: $\theta = (1, 2, 4, 5, 3)$; as componentes conexas ficam agora reduzidas a três; $P_7 = \begin{pmatrix} 2| & 4 & 5 & 0 & 0 & 0 \end{pmatrix}$; e na matriz M vem $M(4, 5) = 2$. A classe 4 não pode mais ser agregada, uma vez que já tem dois predecessores, pelo que o vector de classes activas é $c^2_{act} = \begin{pmatrix} 1 & 2 & 3 & 5 & 6 & 7 \end{pmatrix}$.

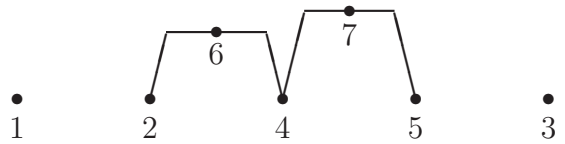


Figura 5.5: Algoritmo RAP – Iteração 2.

Iteração 3 A primeia classe gerada do vector c^2_{act} é $c^3_1 = 1$. Todas as restantes classes são agregáveis com 1, logo $c^3_{agr} = \begin{pmatrix} 2 & 3 & 5 & 6 & 7 \end{pmatrix}$. A segunda classe gerada é $c^3_2 = 6$. É formada a classe 8 que contém o elemento 1 e os elementos 2 e 4 da classe 6, como se vê na Figura 5.6. A ordem sobre os elementos

mantém-se e as componentes conexas reduzem-se a duas: a primeira contendo os elementos $1, 2, 4$ e 5 e a segunda o elemento 3. Na matriz P vem $P_8 = \begin{pmatrix} 3| & 1 & 2 & 4 & 0 & 0 \end{pmatrix}$ e $M(1,6) = 3$. A classe 2, apesar de ter sido agregada apenas uma vez, será retirada das classes activas uma vez que é um nó interior. De salientar que, neste caso, a classe 2 não é livre. Assim, $c^3_{act} = \begin{pmatrix} 1 & 3 & 5 & 6 & 7 & 8 \end{pmatrix}$.

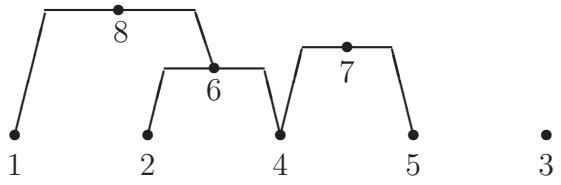


Figura 5.6: Algoritmo RAP – Iteração 3.

Iteração 4 Tem-se $c^4_1 = 3$. As classes agregáveis com 3 são as classes extremas da primeira componente conexa, isto é, $c^4_{agr} = \begin{pmatrix} 1 & 5 & 7 & 8 \end{pmatrix}$. Tem-se $c^4_2 = 7$, pelo que a classe 9 é a agregação do par $(3, 7)$. Neste momento existe apenas uma componente conexa e a ordem $\theta = (1, 2, 4, 5, 3)$ está fixa (ver Figura 5.7). $P_9 = \begin{pmatrix} 4| & 4 & 5 & 3 & 0 & 0 \end{pmatrix}$ e $M(3,7) = 4$. Das classes activas são retiradas as classes 1 e 3, pois havendo apenas uma componente conexa as classes extremas não maximais são excluídas. A classe 5 é também retirada já que é um nó interior, donde $c^4_{act} = \begin{pmatrix} 6 & 7 & 8 & 9 \end{pmatrix}$.

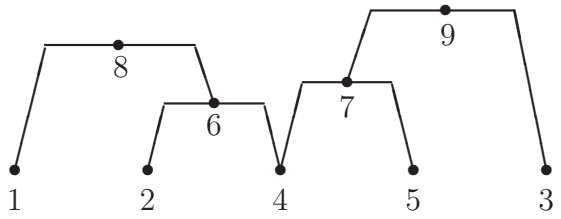


Figura 5.7: Algoritmo RAP – Iteração 4.

Iteração 5 A primeira classe obtida é $c^5_1 = 6$. As classes agregáveis com 6 são $c^5_{agr} = \begin{pmatrix} 7 & 9 \end{pmatrix}$. Destas obtém-se $c^5_2 = 9$. Na Figura 5.8 pode ver-se a formação da classe 10. $P_{10} = \begin{pmatrix} 5| & 2 & 4 & 5 & 3 & 0 \end{pmatrix}$ e $M(6,9) = 5$. Neste momento existem só duas classes activas, $c^5_{act} = \begin{pmatrix} 8 & 10 \end{pmatrix}$.

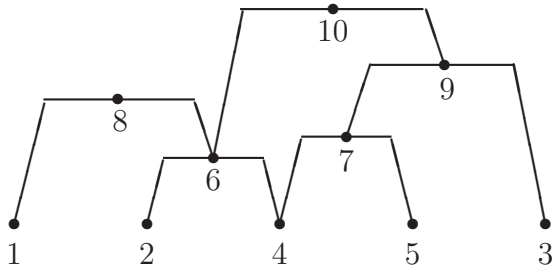


Figura 5.8: Algoritmo RAP – Iteração 5.

Iteração 6 Por fim, a geração do par de classes já não é aleatória uma vez que a solução é única. São agregadas as classes 8 e 10 dando origem à classe 11 (ver Figura 5.9). $P_{11} = \left(6|1 \quad 2 \quad 4 \quad 5 \quad 3\right)$ e $M\,(8, 10) = 6$. Sendo formada a classe que contém todos os elementos o algoritmo pára.

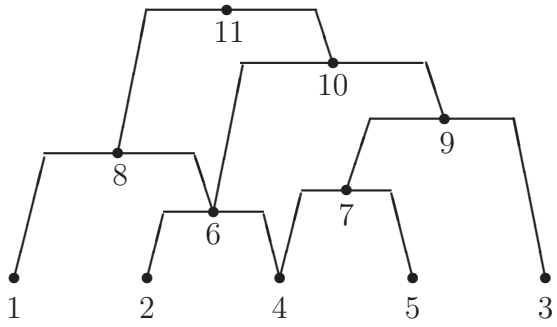


Figura 5.9: Algoritmo RAP – Iteração 6.

Dados de saída: como resultado do algoritmo RAP, que simula uma Classificação Piramidal Ascendente, obtém-se:

- a ordem sobre os elementos, $\theta = (1, 2, 4, 5, 3)$, que se encontra fixa desde o momento que se tem apenas uma componente conexa;
- a matriz P

$$P = \left(\begin{array}{l} 0|\ 1\ 0\ 0\ 0\ 0 \\ 0|\ 2\ 0\ 0\ 0\ 0 \\ 0|\ 3\ 0\ 0\ 0\ 0 \\ 0|\ 4\ 0\ 0\ 0\ 0 \\ 0|\ 5\ 0\ 0\ 0\ 0 \\ 1|\ 2\ 4\ 0\ 0\ 0 \\ 2|\ 4\ 5\ 0\ 0\ 0 \\ 3|\ 1\ 2\ 4\ 0\ 0 \\ 4|\ 4\ 5\ 3\ 0\ 0 \\ 5|\ 2\ 4\ 5\ 3\ 0 \\ 6|\ 1\ 2\ 4\ 5\ 3 \end{array}\right)$$

que origina a pirâmide

$$\{\{1\},\{2\},\{3\},\{4\},\{5\},\{2,4\},\{4,5\},\{1,2,4\},\{4,5,3\},\{2,4,5,3\},\{1,2,4,5,3\}\}$$

- e a matriz de Robinson

$$MR = M(\theta) = \begin{pmatrix} 0 & 3 & 3 & 6 & 6 \\ 3 & 0 & 1 & 5 & 5 \\ 3 & 1 & 0 & 2 & 4 \\ 6 & 5 & 2 & 0 & 4 \\ 6 & 5 & 4 & 4 & 0 \end{pmatrix}.$$

5.6 Considerações Finais

5.6.1 Método *QuikRAP*

Será apresentado de forma sumária um algoritmo de geração aleatória de uma pirâmide que surge como uma adaptação do algoritmo *RAP* aqui proposto e que segue de perto a metodologia do algoritmo *QuikCAP* de Mfoumoune [27]. Apesar de não ter sido implementado computacionalmente, parece pertinente referir alguns melhoramentos que ele pode introduzir.

Por um lado, é necessário criar uma subrotina que permita identificar as classes livres em cada iteração de maneira a considerá-las potencialmente agregáveis como classes extremas. Por outro lado, este algoritmo permitirá a geração aleatória e uniforme do par de classes a agregar de um conjunto de pares possíveis.

A metodologia do método será em tudo semelhante à do algoritmo *QuikCAP*. Inicialmente é introduzido o número de nós terminais, n. Todas as classes singulares, representadas pelos inteiros de 1 até n, são *mutuamente acessíveis*. É definida uma estrutura com todos os pares de classes agregáveis. No seu processo iterativo, o algoritmo segue as seguintes fases:

- geração aleatória e uniforme do par de classes a agregar;
- agregação do par de classes gerado e actualização das variáveis intervenientes;
- eliminação das classes não agregáveis na estrutura definida;
- dedução das acessibilidades da nova classe formada.

O algoritmo termina quando é formada a classe constituída por todos os elementos, isto é, quando não existem mais pares de classes agregáveis.

Algoritmo *QuikRAP*

Segue-se de perto a notação introduzida no algoritmo *RAP*. Define-se a matriz M_p^i, de dimensões $(m_i \times 2)$[3], com todos os pares de classes potencialmente agregáveis na iteração i. Durante o algoritmo as dimensões da matriz vão-se alterando pois, em cada iteração, é necessário proceder à eliminação de pares de classes que passam a não ser agregáveis e acrescentar os novos pares de classes agregáveis com a classe formada.

Iteração 0: é introduzido o número de nós terminais, n, e são inicializadas as variáveis intervenientes: CC, θ, P e M. É definida também a matriz M_p^1.

Iteração i: para i de 1 até à paragem do algoritmo, isto é, até $M_p^i = \emptyset$.

- geração aleatória e uniforme do par de classes a agregar de M_p^i, seja (c_1^i, c_2^i).
- Agregação do par de classes gerado, seja $c^i = c_1^i \cup c_2^i$.[4]
- Eliminação em M_p^i do par (c_1^i, c_2^i) e de todos os pares que não são mais agregáveis.
- actualizações da ordem θ, respeitando a estrutura piramidal, e das matrizes CC, P e M.
- Dedução das novas classes agregáveis com c^i e actualização da matriz M_p, seja M_p^{i+1}.

Dados de saída: ordem θ, matriz P associada à pirâmide e matriz de Robinson $MR = M(\theta)$.

[3] m_i é o número de pares de classes agregáveis na iteração i

[4] Notar que c^i é representado pelo inteiro $n+i$ uma vez que o algoritmo gera pirâmides binárias.

5.6.2 Resumo

Foi implementado um método de geração aleatória de uma pirâmide, denominado de algoritmo *RAP*.

Algumas limitações do algoritmo foram identificadas como a eliminação das classes livres da fronteira da pirâmide em construção e a forma como é gerado o par de classes a agregar. No Capítulo 6 será discutido o desempenho do método, depois de feito um estudo topológico das pirâmides para um número reduzido de nós terminais. Este estudo terá como propósito avaliar se o método gera pirâmides uniformemente, ou pelo menos com distribuição aproximadamente uniforme.

Como alternativa foi pensado e descrito, apesar de não implementado, outro algoritmo de geração aleatória de uma pirâmide que segue de perto o algoritmo de C.P.A proposto por Mfoumoune.

Capítulo 6

Discussão do Método Proposto

6.1 Introdução

Para uma análise do método proposto é importante conhecer o número de pirâmides não isomórficas, bem como a distribuição deste número pelos diferentes tipos topológicos. Para um número fixo de nós terminais pretende-se determinar o número de pirâmides binárias distintas que é possível definir, o número de topologias que lhes estão associadas e o número de pirâmides distintas para cada tipo topológico. No caso particular dos dendrogramas foi apresentado no Capítulo 3, para um valor de n fixo, o número de dendrogramas, o número de tipos topológicos e o número de dendrogramas associado a cada tipo topológico. Para as estruturas piramidais este estudo é bastante mais complexo, não há fórmulas para obter esses valores, pelo que se procurou concretizá-lo apenas para pequenos valores de n.

Assim, serão deduzidas as proporções de cada tipo topológico, fixado n, designadas aqui por frequências teóricas. Recorrendo ao algoritmo do método *RAP*, apresentado no Capítulo 5, geram-se "muitas" pirâmides de n nós terminais e calculam-se as frequências observadas por tipo topológico. A comparação entre as frequências teóricas e observadas permitirá avaliar se o método proposto gera pirâmides aleatória e uniformemente, isto é, se cada uma das pirâmides possíveis é gerada equiprovavelmente.

6.2 Estudo Topológico das Pirâmides

No Capítulo 3, para $n = 4, 5, 6$, foram apresentados esquemas de contagem que permitem obter o número de dendrogramas e tipos topológicos distintos. Nas estruturas piramidais, serão analisados esquemas análogos, mas apenas para as pirâmides com 3 e 4 nós terminais. O caso das pirâmides com 3 nós terminais, apesar de trivial, será apresentado pelo seu carácter pedagógico e porque permitirá compreender melhor a extensão, bem como o incremento de complexidade, ao caso das pirâmides com 4 nós terminais.

6.2.1 Análise para 3 nós terminais

Dendrogramas com 3 nós terminais têm uma decomposição única, variando apenas a sua etiquetagem. O caso das pirâmides, apesar de simples, é ligeiramente diferente. Apresenta-se na Figura 6.1 um esquema de contagem do número de pirâmides distintas que é possível obter com 3 nós terminais.

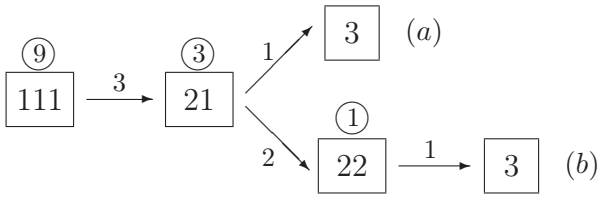


Figura 6.1: Representação esquemática das pirâmides para $n = 3$.

Inicialmente têm-se três classes singulares, representadas por $\boxed{111}$. A primeira agregação pode ser feita de 3 formas distintas, dando origem a uma classe com dois elementos e uma classe singular, representada por $\boxed{21}$. Na fusão seguinte existem quatro classes livres e duas agregações possíveis:

3 pode-se agregar a nova classe formada com a outra classe singular de forma única, formando uma classe com três elementos (Figura 6.1 caminho (a));

22 pode-se agregar uma das duas classes singulares da primeira fusão com a classe ainda não agregada, dando origem a duas classes com dois elementos. Esta agregação pode ser feita de 2 maneiras distintas. De salientar que um dos elementos pertence às duas classes. Estas duas classes são, por fim, agregadas de forma única, formando uma classe com três elementos, representada por $\boxed{3}$ (Figura 6.1 caminho (b)).

A contagem do número de pirâmides é agora feito em sentido inverso. Assim, ③ $= 1 + 2 \times$ ① e ⑨ $= 3 \times$ ③. Para $n = 3$ têm-se nove pirâmides não isomórficas e dois tipos topológicos distintos. Lembra-se que para este valor de n há apenas três dendrogramas distintos e um tipo topológico, correspondente ao caminho (a) da Figura 6.1.

Na Figura 6.2 são apresentados os dois tipos topológicos identificados. Pode ainda ser observada a decomposição utilizada em cada tipo, os níveis de agregação, a representação piramidal e o número de pirâmides associado a cada topologia.

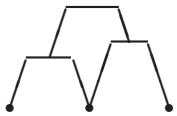
Topologia	Decomposição	Níveis	Representação Piramidal	Número de Piramides
(I)	$3 \longrightarrow 21$	2		3
(II)	$3 \longrightarrow 22$	3		6

Figura 6.2: Análise topológica das pirâmides de 3 nós terminais.

A identificação topológica obtém-se através das diferentes decomposições que é possível efectuar. Assim, para pirâmides binárias com 3 nós terminais, existem apenas duas decomposições possíveis: 21, uma classe com dois elementos e uma classe com um elemento; ou 22, duas classes com dois elementos cada.

Para a topologia I existem 3 pirâmides binárias distintas e 2 níveis de agregação. Para a topologia II existem 6 pirâmides binárias diferentes e 3 níveis de fusão.

6.2.2 Análise para 4 nós terminais

Para pirâmides com 4 nós terminais este estudo torna-se bastante mais complexo, pelo que a sua extensão para valores de n superiores implica já uma análise mais cuidada e morosa. Como tem vindo a ser feito, na Figura 6.3, apresenta-se um esquema de contagem do número de pirâmides distintas que é possível ter com 4 nós terminais.

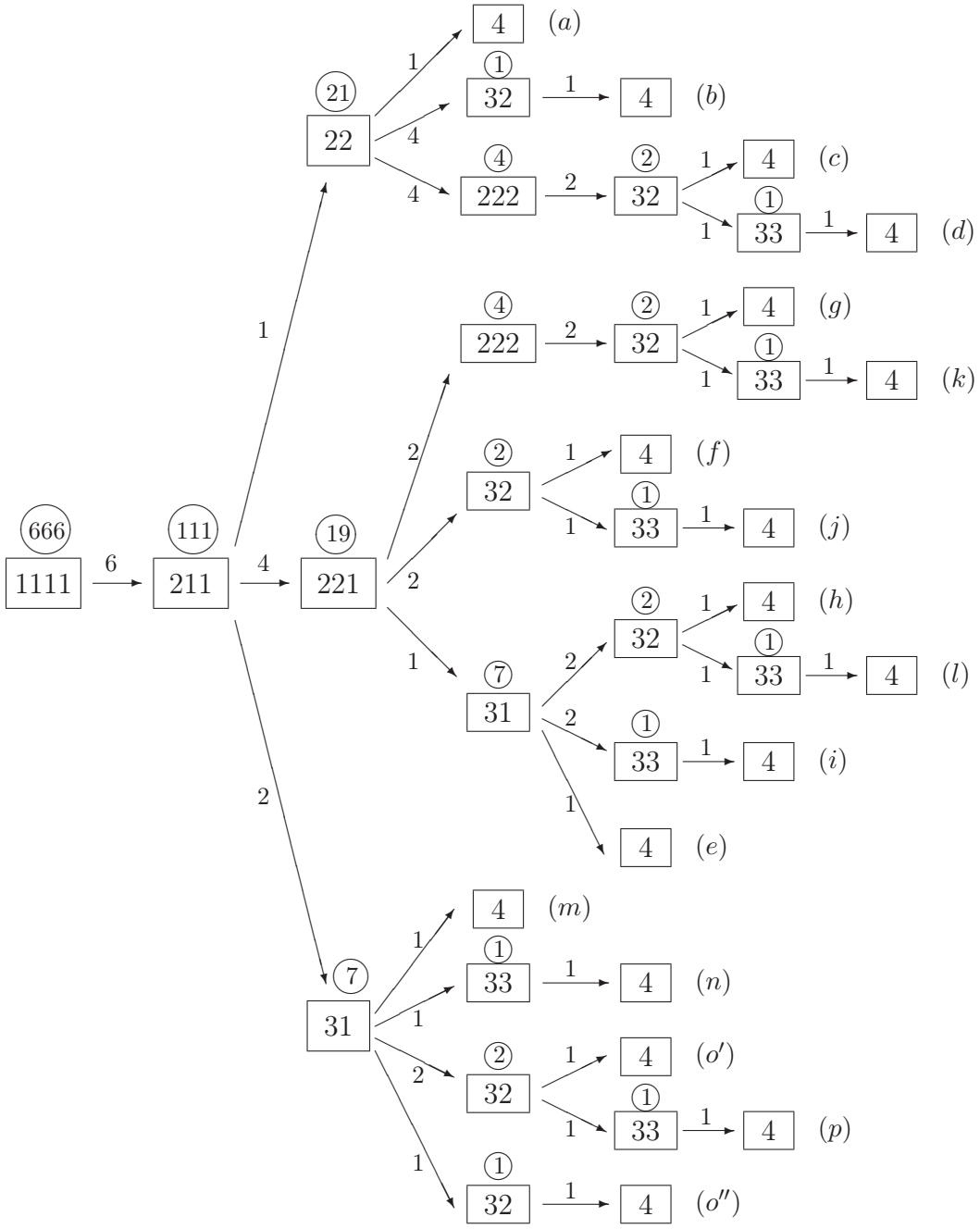
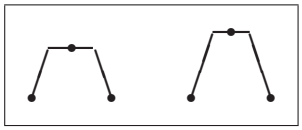


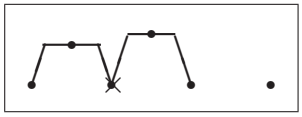
Figura 6.3: Representação esquemática das pirâmides para $n = 4$.

Inicialmente têm-se as quatro classes singulares, representadas por $\boxed{1111}$. A primeira agregação pode ser feita de $\binom{4}{2} = 6$ formas distintas, dando origem a uma nova classe com dois elementos. Usar-se-á a notação $\boxed{211}$ para indicar que existe agora uma classe com dois elementos e duas classes singulares ainda não agregadas. Nesta fase, há cinco classes livres: as quatro classes singulares e a nova classe formada. Na agregação seguinte existem três possibilidades de fusão:

$\boxed{\mathbf{22}}$ podem ser agregadas as duas classes singulares ainda não agregadas, dando origem a uma nova classe com dois elementos. Esta agregação é feita de forma única. Têm-se então seis classes livres: as quatro singulares, que foram agregadas uma única vez, e as duas classes formadas, com dois elementos cada[1].



$\boxed{\mathbf{221}}$ podem ser unidas uma das duas classes singulares ainda não agregadas com uma das duas classes singulares já agregadas no passo anterior. Esta fusão pode ser feita de 4 formas distintas. Um dos elementos já foi agregado duas vezes, assinalado na imagem com $\times$, pelo que não pode mais ser agregado. Existem agora cinco classes livres: três singulares, das quais os elementos de duas já foram agregados uma vez, e duas classes com dois elementos cada[2].

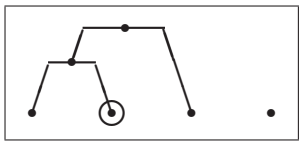


$\boxed{\mathbf{31}}$ podem ser agregadas uma das duas classes singulares ainda não agregadas com a nova classe formada, composta por dois elementos. Esta agregação pode ser efectuada de 2 maneiras diferentes, originando seis classes livres: as quatro classes singulares e as duas novas classes formadas[3]. De salientar que a classe assinalada com uma circunferência mantém-se livre, uma vez que é possível fazer a inversão da classe composta por dois elementos.

[1] a notação $\boxed{22}$ resulta de se ter estas duas classes com dois elementos.

[2] a notação $\boxed{221}$ resulta de se ter estas duas classes com dois elementos e uma classe singular ainda não agregada.

[3] a notação $\boxed{31}$ resulta de se ter uma classe com três elementos e uma classe singular ainda não agregada.



Todas as possíveis agregações que se seguem podem ser acompanhadas no esquema da Figura 6.3 da página 98. As diferentes pirâmides binárias que são possíveis obter estão identificadas neste esquema por uma letra.

A explicação de cada um destes caminhos tornava-se maçadora, pelo que será apenas exemplificado para um dos caminhos. Considere-se o caminho (f) que se encontra esquematizado na Figura 6.4.

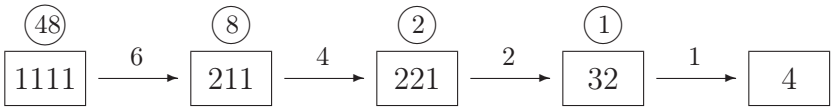


Figura 6.4: Esquema de contagem do número de pirâmides para o caminho (f) da Figura 6.3.

Como já foi visto, na primeira agregação, de $\boxed{1111}$ para $\boxed{211}$, existem 6 maneiras distintas de o fazer e na segunda fusão, de $\boxed{211}$ para $\boxed{221}$, há 4 formas diferentes. De $\boxed{221}$ para $\boxed{32}$ existem 2 agregações possíveis, uma das duas classes com dois elementos com a classe singular ainda não agregada. Por fim, de $\boxed{32}$ para $\boxed{4}$, a agregação é feita de forma única. O número de pirâmides binárias distintas para este caminho é agora dado pelo produto $1 \times 2 \times 4 \times 6 = 48$. Na figura 6.8 da página 102 pode ser observada a representação piramidal associada ao caminho (f).

Esta análise foi feita para cada um dos caminhos e apresenta-se sintetizada nas Figuras 6.8 e 6.9. Assim, podem ser observados, para cada caminho, os níveis de agregação, a representação piramidal e o número de pirâmides não isomórficas associadas.

No esquema da Figura 6.3 existem dois caminhos com a letra (o), (o') e (o''). Tal facto deve-se à diferenciação que deve ser feita, no patamar $\boxed{31}$, às respectivas classes singulares dos elementos que constituem a classe com três objectos. Isto é, deve distinguir-se a situação das duas classes singulares activas que deram origem à primeira classe formada, constituída por dois elementos, da situação da classe singular que deu origem à formação da classe com três elementos.

Na Figura 6.5 pode observar-se o esquema de contagem associado à letra (o), que contempla os caminhos (o') e (o'') da Figura 6.3.

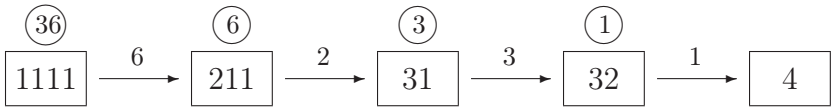


Figura 6.5: Esquema de contagem do número de pirâmides para os caminhos (o') e (o'') da Figura 6.3.

A bifurcação dos caminhos, na Figura 6.3 da página 98, surge na passagem de $\boxed{31}$ para $\boxed{32}$, isto é, quando se agrega uma das três classes singulares já agregadas uma vez com a classe singular ainda não agregada. Neste passo, podem ocorrer uma das duas situações seguintes:

- ser agregada uma das duas classes singulares, que deram origem à primeira classe formada (de dois elementos) com a classe singular ainda não agregada. Na Figura 6.6 pode visualizar-se a representação piramidal correspondente a este caminho. De referir que este caminho origina 24 pirâmides distintas.

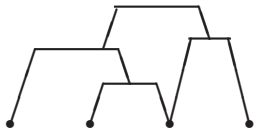


Figura 6.6: Representação piramidal para o caminho (o') da Figura 6.3.

- ser agregada a classe singular, que deu origem à formação da classe com três elementos, com a classe singular ainda não agregada. Na Figura 6.7, que é a mesma utilizada na Figura 6.9, caminho (o''), pode observar-se esta representação piramidal. De referir que este caminho origina 12 pirâmides diferentes.

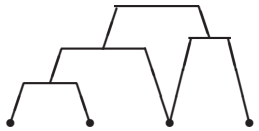


Figura 6.7: Representação piramidal para o caminho (o'') da Figura 6.3.

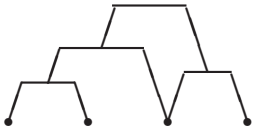
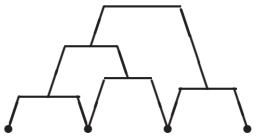
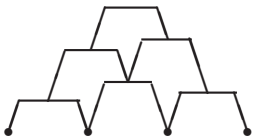
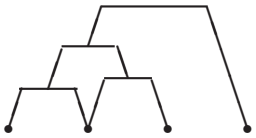
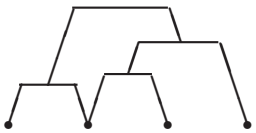
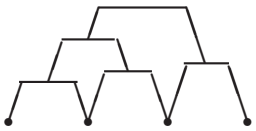
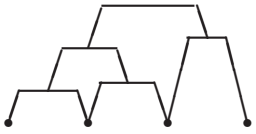
	Caminho	Níveis	Representação Piramidal	Número de Piramides
(a)	$4-22-211$	3		6
(b)	$4-32-22-211$	4		24
(c)	$4-32-222-22-211$	5		48
(d)	$4-33-32-222-22-211$	6		48
(e)	$4-31-221-211$	4		24
(f)	$4-32-221-211$	4		48
(g)	$4-32-222-221-211$	5		96
(h)	$4-32-31-221-211$	5		48

Figura 6.8: Contagem do número de pirâmides para $n = 4$.

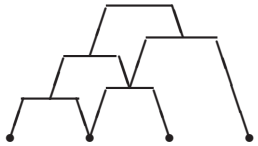
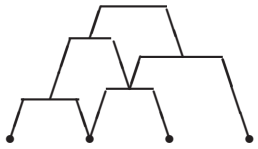
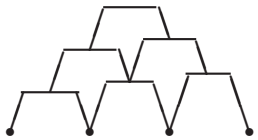
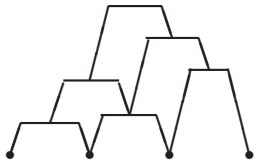
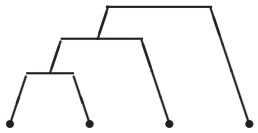
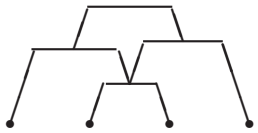
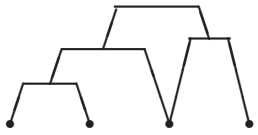
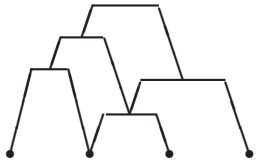
	Caminho	Níveis	Representação Piramidal	Número de Piramides
(i)	$4-33-31-221-211$	5		48
(j)	$4-33-32-221-211$	5		48
(k)	$4-33-32-222-221-211$	6		96
(l)	$4-33-32-31-221-211$	6		48
(m)	$4-31-211$	3		12
(n)	$4-33-31-211$	4		12
(o)	$4-32-31-211$	4		36
(p)	$4-33-32-31-211$	5		24

Figura 6.9: Contagem do número de pirâmides para $n = 4$ (continuação).

Para a obtenção do número de pirâmides binárias distintas com 4 nós terminais procede-se de forma análoga ao que tem sido feito. Assim,

$$\begin{aligned}
\textcircled{21} &= 1+4\times\textcircled{1}+4\times\textcircled{4}\ ,\\
\textcircled{19} &= 2\times\textcircled{4}+2\times\textcircled{2}+1\times\textcircled{7}\ ,\\
\textcircled{7} &= 1+1\times\textcircled{1}+2\times\textcircled{2}+1\times\textcircled{1}\ .
\end{aligned}$$

Agora,

$$\textcircled{111} = 1\times\textcircled{21}+4\times\textcircled{19}+2\times\textcircled{7}\ .$$

Por fim,

$$\textcircled{666} = 6\times\textcircled{111}\ .$$

Portanto, o número de pirâmides binárias distintas com 4 nós terminais é **666**. Este número pode também ser facilmente obtido somando todos os valores da coluna *número de pirâmides* das Figuras 6.8 e 6.9, ou seja, adicionando todas as pirâmides distintas obtidas para cada caminho da Figura 6.3.

Lembra-se que para este valor de n existem apenas 18 dendrogramas distintos e dois tipos topológicos, correspondentes aos caminhos (a) e (m) da Figura 6.3. É possível verificar que a complexidade, no caso das pirâmides, aumenta "exponencialmente" relativamente aos dendrogramas. Far-se-á a seguir um estudo topológico para pirâmides com 4 nós terminais.

Para a identificação dos tipos topológicos utiliza-se uma metodologia descendente, isto é, parte-se da raiz composta por 4 elementos e efectuam-se as decomposições possíveis. Assim, a raiz pode ser decomposta de quatro formas distintas:

- 22 – são atribuídos 2 elementos a cada uma das classes;
- 31 – são atribuídos 3 elementos a uma das classes e 1 elemento à outra;
- 32 – são atribuídos 3 elementos a uma das classes e 2 elementos à outra;
- 33 – são atribuídos 3 elementos a cada uma das classes.

As classes com 3 elementos podem ser decompostas de duas formas distintas:

- 21 – são atribuídos 2 elementos a uma das classes e 1 elemento à outra;
- 22 – são atribuídos 2 elementos a cada uma das classes.

Uma classe com 2 elementos tem decomposição única, isto é, origina uma partição de duas classes singulares.

Usando estas decomposições identificaram-se **8** tipos topológicos distintos para as pirâmides binárias com 4 nós terminais. Na Figura 6.10 estão representadas estas oito topologias. Para cada uma delas pode ser observada a decomposição efectuada, os níveis de agregação, a representação piramidal e o número de pirâmides que lhe está associado.

Existem topologias que englobam mais do que um caminho da Figura 6.3. Neste caso, a representação piramidal utilizada é aquela cuja letra se encontra a negrito.

De salientar ainda que na última fase da decomposição existem números centrais com um ponto em cima. Isto porque a classe representada por este número, com um ponto, resulta da decomposição do ramo direito de uma classe e do ramo da esquerda da outra. Por exemplo, na topologia V tem-se a decomposição $4 \longrightarrow 33 \longrightarrow 1\dot{2}1$. Quer isto dizer que inicialmente a raiz é decomposta em duas classes com 3 elementos cada. Estas classes são, por sua vez, decompostas na forma 21, ou seja, 2 elementos para uma classe e 1 elemento para outra. A classe $\dot{2}$ é a classe com 2 elementos comum à decomposição de ambas as classes com 3 elementos.

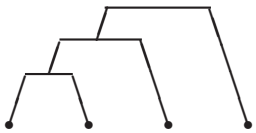
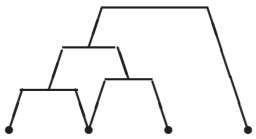
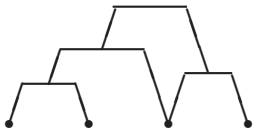
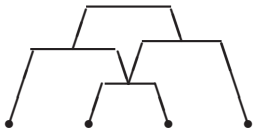
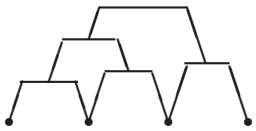
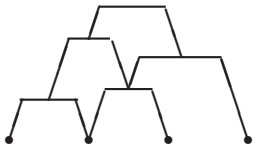
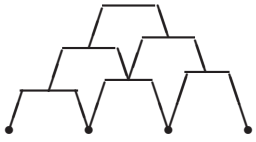
Topologia	Decomposição	Níveis	Representação Piramidal	Número de Piramides
(I)	$4 \longrightarrow 22$ **(a)**	3		6
(II)	$4 \longrightarrow 31 \longrightarrow 211$ **(m)**	3		12
(III)	$4 \longrightarrow 31 \longrightarrow 221$ **(e)**	4		24
(IV)	$4 \longrightarrow 32 \longrightarrow 212$ **(b)**, (f), (o)	4		108
(V)	$4 \longrightarrow 33 \longrightarrow 1\dot{2}1$ **(n)**	4		12
(VI)	$4 \longrightarrow 32 \longrightarrow 222$ (c), **(g)**, (h)	5		192
(VII)	$4 \longrightarrow 33 \longrightarrow 2\dot{2}1$ (i), **(j)**, (p)	5		120
(VIII)	$4 \longrightarrow 33 \longrightarrow 2\dot{2}2$ (d), **(k)**, (l)	6		192

Figura 6.10: Análise topológica das pirâmides com 4 nós terminais.

6.3 Discussão do Método Proposto

Recorrendo a ferramentas de cálculo combinatório contabilizou-se o número de pirâmides com 3 e 4 nós terminais para cada tipo topológico. Um objectivo deste estudo foi o de averiguar se o algoritmo *RAP* pode ser entendido como um método de geração aleatória e uniforme no sentido de Furnas, isto é, se cada uma das pirâmides possíveis é gerada com igual probabilidade.

Para as pirâmides com 3 nós terminais verificou-se que o método *RAP* gera estas pirâmides de forma uniforme, ou seja, gera pirâmides do tipo topológico I com probabilidade $\frac{1}{3}$ e do tipo topológico II com probabilidade $\frac{2}{3}$.

Para as pirâmides com 4 nós terminais contabilizaram-se 666 pirâmides binárias distintas e o respectivo número por cada um dos oito tipos topológicos identificados. Estes valores, assim como a distribuição do conjunto das pirâmides não distinguíveis pelos diferentes tipos topológicos (Freq. Teóricas), podem ser observados na Tabela 6.1.

Tipos Topológicos	I	II	III	IV	V	VI	VII	VIII
Número de Pirâmides	6	12	24	108	12	192	120	192
Freq. Teóricas	0.009	0.018	0.036	0.162	0.018	0.288	0.181	0.288

Tabela 6.1: Número de pirâmides e frequências teóricas, por tipo topológico, para 4 nós terminais.

Com base no método proposto, *RAP*, foram geradas 100000 pirâmides, identificados os respectivos tipos topológicos e calculadas as frequências observadas (Freq. Observadas). Os valores obtidos encontram-se na Tabela 6.2.

Tipos Topológicos	I	II	III	IV	V	VI	VII	VIII
Freq. Observadas	0.011	0.076	0.024	0.245	0.077	0.195	0.175	0.197

Tabela 6.2: Frequências observadas, por tipo topológico, para pirâmides com 4 nós terminais obtidas pelo algoritmo *RAP*.

A comparação dos valores das frequências, teóricas e observadas, leva-nos a pensar que o método proposto não gera pirâmides uniformemente. Numa breve análise dos valores das frequências observadas, quando comparados com as teóricas, referem-se as diferenças nos tipos II e IV, por excesso, e nos tipos VI e VIII, por defeito. Contudo, pelos valores obtidos pode afirmar-se que a distribuição obtida tende a não se afastar muito da uniforme.

Podem ainda ser comparados os valores das frequências, teóricas e observadas, por nível de agregação, isto é, avaliar se o método gera pirâmides de 4 elementos para cada nível de agregação (de 3 a 6) de forma uniforme. Os respectivos valores podem ser consultados na Tabela 6.3.

Nível de agregação	Freq. Teórica	Freq. Observad
3	0.027	0.087
4	0.216	0.346
5	0.469	0.37
6	0.288	0.197

Tabela 6.3: Frequências teóricas e observadas, por nível de agregação, para pirâmides com 4 nós terminais.

Observa-se que o método proposto tem tendência para gerar pirâmides com nível de agregação inferior. Aumentando o número de nós terminais é provável que não sejam geradas pirâmides binárias *saturadas*, isto é, com todos os $\frac{n(n-1)}{2}$ níveis de agregação possíveis. Este facto deve-se sobretudo ao aspecto já referido da exclusão das classes livres da fronteira da pirâmide em construção.

6.4 Considerações Finais

A análise topológica das pirâmides com mais de 4 nós terminais é inquestionavelmente necessária para a avaliação do comportamento do algoritmo *RAP*. Tentar encontrar relações de combinatória que permitam obter o número de pirâmides não isomórficas fixado n e o número de diferentes tipos topológicos associados, é um tema pertinente de estudo, se bem que, em nosso entender, de dificíl concretização.

Na sequência desta dificuldade procurou-se fazer o estudo enumerativo, análogo ao que havia sido feito para $n = 4$, para pirâmides com 5 nós terminais. Contudo a complexidade desta análise levou a que esse trabalho não fosse concluído. O problema encontrado na identificação dos caminhos assinalados com a letra (o) na Figura 6.3, acentua-se bastante quando se consideram as pirâmides com 5 nós terminais.

Outro trabalho a realizar num futuro próximo é a implementação do algoritmo associado ao método *QuikRAP*, permitindo eliminar algumas limitações referidas que existem no algoritmo *RAP*. A concretização deste dois últimos objectivos referidos: a determinação das frequências teóricas por tipo topológico para $n = 5$ e a

implementação do algoritmo *QuikRAP*, poderá permitir um desenvolvimento considerável nesta área do conhecimento. Se, como esperámos, a geração aleatória de pirâmides, no algoritmo *QuikRAP*, tender a aproximar-se da distribuição uniforme, então daí pode resultar um grande contributo para a contabilização do número de pirâmides não isomórficas e identificação dos tipos topológicos, para valores de n superiores.

6.4.1 Resumo

Neste capítulo procurou-se avaliar se o método *RAP* gera pirâmides aleatória e uniformemente.

Para a análise do método proposto foi importante conhecer o número de pirâmides não isomórficas, bem como a distribuição deste número pelos diferentes tipos topológicos. Dada a complexidade deste estudo, este objectivo foi apenas conseguido para pirâmides com 3 e 4 nós terminais. Assim, para $n = 3$ têm-se 9 pirâmides não isomórficas e dois tipos topológicos diferentes. No caso de $n = 4$ foram contabilizadas 666 pirâmides não isomórficas e identificados oito tipos topológicos distintos.

Do estudo de simulação realizado verificou-se que o método *RAP* não gera pirâmides uniformemente, como era esperado pelas limitações identificadas no algoritmo. Contudo, pode afirmar-se que esta geração não se afasta muito da uniforme.

Capítulo 7

Considerações Finais e Desenvolvimentos Futuros

O objectivo deste trabalho foi o de propor um método de geração aleatória de estruturas piramidais. Fixado o número de nós terminais pretendido, este método gera uma pirâmide tendo em conta os três aspectos que a caracterizam completamente: a sua topologia, os níveis de agregação e a identificação dos nós terminais.

Começou-se por analisar e comparar duas metodologias de Classificação: a Classificação Hierárquica e a Classificação Piramidal. A Classificação Piramidal é uma generalização da Classificação Hierárquica, pelo que origina estruturas mais complexas: num mesmo patamar, um elemento pode pertencer a mais do que uma classe e é produzido um número reduzido de ordens compatíveis sobre o conjunto de elementos a classificar.

O método *RAP* aqui apresentado surge como uma extensão de trabalhos anteriores de geração de dendrogramas. Alguns destes métodos foram analisados, com especial ênfase para os algoritmos de geração aleatória e uniforme no sentido de Furnas [19]: Permutação Dupla, Geração Uniforme e RA. Foi também feito um estudo topológico dos dendrogramas para uma melhor compreensão do estudo posteriormente realizado para as pirâmides. Verificou-se que a complexidade aumenta exponencialmente quando passamos de um dendrograma para uma estrutura piramidal.

A metodologia Classificação Piramidal foi apresentada com detalhe, introduzindo-se várias noções e propriedades necessárias para a compreensão e implementação de algoritmos de Classificação Piramidal Ascendente. Foram apresentados os algoritmos de Diday [12], Bertrand [4] e Mfoumoune[27] e analisada a evolução dos mesmos. Este estudo foi fundamental para a posterior proposta dos algoritmos de

geração aleatória de pirâmides.

No algoritmo *RAP* foram identificadas algumas limitações, como a eliminação das classes livres da fronteira da pirâmide em construção e a forma como é gerado o par de classes a agregar, que não é obtido de forma uniforme de um conjunto que contenha todos os pares de classes agregáveis numa determinada iteração. Tendo por objectivo conhecer o desempenho deste algoritmo, para um número reduzido de nós terminais, procedeu-se à identificação dos diferentes tipos topológicos, bem como à determinação do respectivo número de pirâmides não isomórficas. Este estudo teve como propósito avaliar se o método gera pirâmides uniformemente.

Existem 9 pirâmides não isomórficas com 3 nós terminais e dois tipos topológicos distintos. Para as pirâmides de 4 nós terminais contabilizaram-se 666 pirâmides não isomórficas distribuídas por oito tipos topológicos diferentes. Dada a complexidade deste estudo, este objectivo foi concretizado apenas para pirâmides com 3 e 4 nós terminais. Como já era esperado, em virtude das limitações que o algoritmo *RAP* apresenta, o estudo de simulação realizado permitiu verificar que o método não gera pirâmides uniformemente, contudo esta geração não se afasta muito da uniforme.

Algumas das perspectivas de desenvolvimento futuro foram já referidas ou sugeridas ao longo do trabalho. Há ainda muito trabalho a fazer neste domínio, quer a nível teórico quer a nível experimental ou de simulação.

A análise topológica das pirâmides com mais de 4 nós terminais parece ser um estudo de combinatória bastante interessante. Permitirá avaliar se o método proposto tem tendência para se afastar da geração uniforme.

Procurou-se fazer este estudo para pirâmides com 5 nós terminais. Contudo, a complexidade desta análise levou a que este trabalho não fosse concretizado. O problema encontrado na identificação dos caminhos assinalados com a letra (o) na Figura 6.3 multiplica-se agora nas pirâmides com 5 nós terminais. Será um aspecto importante encontrar relações de combinatória que permitam obter o número de pirâmides não isomórficas fixado n e o número de diferentes tipos topológicos associados.

Alternativamente aos métodos de geração aleatória de estruturas de classificação anteriormente apresentados, Flajolet et al. [18] e Van Cutsem [39] desenvolveram técnicas de cálculo para estruturas combinatórias e um método para a sua geração aleatória. Van Cutsem [39] mostrou que as estruturas de classificação são casos par-

ticulares de estruturas combinatórias, o que permite usar o método desenvolvido para gerar estruturas classificatórias. Um estudo mais aprofundado destes trabalhos e a sua adaptação às estruturas piramidais poderá ajudar na resolução de alguns problemas encontrados.

A eliminação das limitações apresentadas pelo algoritmo *RAP* poderá ser conseguida com a implementação do método *QuikRAP*. Este poderá permitir avaliar se são geradas aleatória e uniformemente pirâmides de 4 nós terminais. Desta análise, pode resultar um grande contributo deste método para a contagem do número de pirâmides não isomórficas e identificação dos diferentes tipos topológicos, para valores de n superiores. Pode também ser importante depois na comparação do desempenho dos diferentes métodos de Classificação Piramidal Ascendente propostos na literatura.

Como trabalho futuro refere-se então o estudo mais geral sobre a identificação topológica das estruturas piramidais e o desenvolvimento de outros algoritmos de geração aleatória de pirâmides. Procurou-se desenvolver outro método de geração apesar de este não ser referido ao longo do trabalho. Parece natural a extensão do método de Permutação Dupla, de geração aleatória de dendrogramas, para um método semelhante que permita a geração aleatória de pirâmides. Isto é, assim como no método de Permutação Dupla o problema da geração aleatória de um dendrograma é equivalente à geração de uma matriz ultramétrica, o problema da geração aleatória de uma pirâmide pode reduzir-se à geração da respectiva matriz de Robinson. Apesar de alguns esforços tentados este objectivo não foi conseguido.

A identificação de alguns desenvolvimentos futuros ilustra a importância do tema abordado neste trabalho. É uma área com problemas de difícil solução, com novas questões a serem levantadas e para as quais se impõem contribuições futuras.

Anexos

Anexo 1— Identificação Topológica de Dendrogramas

Construção do vector ultramétrico e do vector ultramétrico ordenado

```
function [vector_ultrmetrico,vector_ultrmetrico_ord] = vector_ult(matriz_ultrametrica);
    [n,d] = size(matriz_ultrametrica);
    vector_ultrmetrico = zeros(1,(n^2-n)/2);
    k = 1;
    for i = 1:n-1;
        for j = i+1:n;
            vector_ultrmetrico(1,k) = matriz_ultrametrica(i,j);
            k = k+1;
        end;
    end;
    vector_ultrmetrico_ord = sort(vector_ultrmetrico);
return;
```

Identificação da proporção das topologias para 4 nós terminais

```
function [prob_top_1,prob_top_2] = topologias_4_...(m,display);
    % m - numero de dendrogramas a gerar
    top_1 = 0;
    top_2 = 0;
    for i = 1:m;
        [Matriz_ultrametrica,T] = ;
        [vector_ultrmetrico,vector_ultrmetrico_ord] = vector_ult(Matriz_ultrametrica);
        if vector_ultrmetrico_ord(3) = = 2;
            top_1 = top_1+1;
        else;
            top_2 = top_2+1;
        end;
    end;
    prob_top_1 = top_1/m;
    prob_top_2 = top_2/m;
    disp(sprintf('A probabilidde do dendrogama ter topologia I =
    = %g e topologia II = %g',prob_top_1,prob_top_2));
return;
```

Identificação da proporção das topologias para 5 nós terminais

```
function [prob_top_1,prob_top_2,prob_top_3] = topologias_5_...(m,display);
    % m - numero de dendrogramas a gerar
    top_1 = 0;
    top_2 = 0;
    top_3 = 0;
    for I = 1:m;
        [Matriz_ultrametrica,T] =  (5,display);
        [vector_ultrmetrico,vector_ultrmetrico_ord] = vector_ult(Matriz_ultrametrica);
        if vector_ultrmetrico_ord(5) = = 4;
            top_2 = top_2+1;
        else;
```

```
            if vector_ultrmetrico_ord(3) = = 2;
                top_1 = top_1+1;
            else;
                top_3 = top_3+1;
            end;
        end;
    end;
    prob_top_1 = top_1/m;
    prob_top_2 = top_2/m;
    prob_top_3 = top_3/m;
    disp(sprintf('A probabilidde do dendrogama ter topologia I = %g, topologia II =
    = %g e topologia III = %g',prob_top_1,prob_top_2,prob_top_3));
return;
```

Identificação da proporção das topologias para 6 nós terminais

```
function [prob_top_1,prob_top_2,prob_top_3,prob_top_4,prob_top_5,prob_top_6] =
= topologias_6_...(m,display);
    % m - numero de dendrogramas a gerar
    top_1 = 0;
    top_2 = 0;
    top_3 = 0;
    top_4 = 0;
    top_5 = 0;
    top_6 = 0;
    for i = 1:m;
        [Matriz_ultrametrica,T] =  (6,display);
        [vector_ultrmetrico,vector_ultrmetrico_ord] = vector_ult(Matriz_ultrametrica);
        if vector_ultrmetrico_ord(7) = = 5;
            top_6 = top_6+1;
        elseif vector_ultrmetrico_ord(8) = = 5;
            if (vector_ultrmetrico_ord(4) = = 4) |
            | (vector_ultrmetrico_ord(3) = = vector_ultrmetrico_ord(6));
                top_5 = top_5+1;
            else;
                top_4 = top_4+1;
            end;
        else;
            if vector_ultrmetrico_ord(5) = = 4;
                top_3 = top_3+1;
            elseif vector_ultrmetrico_ord(3) = = 3;
                top_2 = top_2+1;
            else;
                top_1 = top_1+1;
            end;
        end;
    end;
    prob_top_1 = top_1/m;
    prob_top_2 = top_2/m;
    prob_top_3 = top_3/m;
    prob_top_4 = top_4/m;
    prob_top_5 = top_5/m;
    prob_top_6 = top_6/m;
    disp(sprintf('A probabilidde do dendrogama ter topologia:'));
    disp(sprintf('I = %g',prob_top_1));
```

```
    disp(sprintf('II = %g',prob_top_2));
    disp(sprintf('III = %g',prob_top_3));
    disp(sprintf('IV = %g',prob_top_4));
    disp(sprintf('V = %g',prob_top_5));
    disp(sprintf('VI = %g',prob_top_6));
return;
```

Anexo 2— Algoritmo do Método de Permutação Dupla

```
function [matriz_fill,vector_perm,matriz_perm] = permutacao_dupla(n);
    %vector aleatorio
    vector_alea = randperm(n-1);
    %matriz ultrametrica
    matriz_fill = zeros(n);
    for i = 1:n-1;
        matriz_fill(i,i+1) = vector_alea(i);
        matriz_fill(i+1,i) = vector_alea(i);
    end;
    for i = 1:n-2;
        for j = i+2:n;
            if matriz_fill(i,j-1) > matriz_fill(j-1,j);
                matriz_fill(i,j) = matriz_fill(i,j-1);
            else;
                matriz_fill(i,j) = matriz_fill(j-1,j);
            end;
            matriz_fill(j,i) = matriz_fill(i,j);
        end;
    end;
    %permutaçao dos objectos
    vector_perm = randperm(n);
    matriz_perm = matriz_fill(vector_perm,vector_perm);
return;
```

Anexo 3— Algoritmo de Geração Uniforme

```
function [Matriz_ultrametrica,T] = geracao_uniforme(n_objectos,display);
    %Gera aleatoria e uniformemente uma arvore binaria de n objectos;
    for i = 1:n_objectos-1;
        vector_objectos(i) = i;
        vector_niveis(i) = i;
    end;
    vector_objectos(n_objectos) = n_objectos;
    T = zeros(n_objectos-1,3);
    Matriz_ultrametrica = zeros(n_objectos);

    matriz_objectos = zeros(n_objectos);
    matriz_niveis = zeros(n_objectos-1);
    nivel_aux = zeros(1,n_objectos);
    nivel_maximo = max(vector_niveis);
    nivel_auxiliar = 0;
    k = 1;
    while nivel_maximo > 0;
        n_niveis = length(vector_niveis);
        if n_niveis = = 1;
            if nivel_maximo ~= vector_niveis(1);
                disp('Problema!');
            end;
            Matriz_ultrametrica(vector_objectos(1),vector_objectos(2)) = nivel_maximo;
            Matriz_ultrametrica(vector_objectos(2),vector_objectos(1)) = nivel_maximo;
            if vector_objectos(1) < vector_objectos(2);
                T(nivel_maximo,1) = vector_objectos(1);
                T(nivel_maximo,2) = vector_objectos(2);
                T(nivel_maximo,3) = nivel_maximo;
            else;
                T(nivel_maximo,1) = vector_objectos(2);
                T(nivel_maximo,2) = vector_objectos(1);
                T(nivel_maximo,3) = nivel_maximo;
            end;
            if vector_niveis(1) ~= 1;
                nivel_auxiliar = max(nivel_aux);
                nivel_maximo = nivel_auxiliar;
                [vector_objectos,vector_niveis,nivel_aux] =
                = find_vectores(nivel_maximo,matriz_objectos,matriz_niveis,nivel_aux,k);
            else
                nivel_maximo = nivel_maximo-1;
            end
        else
            vector_niveis_aux = zeros(1,n_niveis-1);
            for i = 1:n_niveis-1
                vector_niveis_aux(i) = vector_niveis(i);
            end;
            [n_objectos_esquerda,n_objectos_direita,objectos_esquerda,
            objectos_direita,niveis_esquerda,niveis_direita] =
            = parte_objectos_niveis(vector_objectos,vector_niveis_aux);
            if niveis_esquerda = = 0;
                objectos_direita = sort(objectos_direita);
                Matriz_ultrametrica(objectos_esquerda,objectos_direita) = nivel_maximo;
                Matriz_ultrametrica(objectos_direita,objectos_esquerda) = nivel_maximo;
                if objectos_esquerda(1) < objectos_direita(1)
                    T(nivel_maximo,1) = objectos_esquerda(1);
```

```
        T(nivel_maximo,2) = objectos_direita(1);
        T(nivel_maximo,3) = nivel_maximo;
    else;
        T(nivel_maximo,1) = objectos_direita(1);
        T(nivel_maximo,2) = objectos_esquerda(1);
        T(nivel_maximo,3) = nivel_maximo;
    end;
elseif niveis_direita = = 0;
    objectos_esquerda = sort(objectos_esquerda);
    Matriz_ultrametrica(objectos_esquerda,objectos_direita) = nivel_maximo;
    Matriz_ultrametrica(objectos_direita,objectos_esquerda) = nivel_maximo;
    if objectos_esquerda(1) < objectos_direita(1);
        T(nivel_maximo,1) = objectos_esquerda(1);
        T(nivel_maximo,2) = objectos_direita(1);
        T(nivel_maximo,3) = nivel_maximo;
    else;
        T(nivel_maximo,1) = objectos_direita(1);
        T(nivel_maximo,2) = objectos_esquerda(1);
        T(nivel_maximo,3) = nivel_maximo;
    end;
else;
    objectos_esquerda = sort(objectos_esquerda);
    objectos_direita = sort(objectos_direita);
    Matriz_ultrametrica(objectos_esquerda,objectos_direita) = nivel_maximo;
    Matriz_ultrametrica(objectos_direita,objectos_esquerda) = nivel_maximo;
    if objectos_esquerda(1) < objectos_direita(1);
        T(nivel_maximo,1) = objectos_esquerda(1);
        T(nivel_maximo,2) = objectos_direita(1);
        T(nivel_maximo,3) = nivel_maximo;
    else;
        T(nivel_maximo,1) = objectos_direita(1);
        T(nivel_maximo,2) = objectos_esquerda(1);
        T(nivel_maximo,3) = nivel_maximo;
    end;
end;
nivel_maximo_esquerda = max(niveis_esquerda);
nivel_maximo_direita = max(niveis_direita);
if nivel_auxiliar > max(nivel_maximo_esquerda,nivel_maximo_direita);
    [vector_objectos,vector_niveis,nivel_aux] =
    = find_vectores(nivel_auxiliar,matriz_objectos,matriz_niveis,nivel_aux,k);
    nivel_maximo = max(vector_niveis);
    if niveis_esquerda ~= 0;
        nivel_aux(k) = nivel_maximo_esquerda;
        matriz_objectos(k,1:n_objectos_esquerda) = objectos_esquerda;
        matriz_niveis(k,1:n_objectos_esquerda-1) = sort(niveis_esquerda);
        k = k+1;
    end
    if niveis_direita ~= 0;
        nivel_aux(k) = nivel_maximo_direita;
        matriz_objectos(k,1:n_objectos_direita) = objectos_direita;
        matriz_niveis(k,1:n_objectos_direita-1) = sort(niveis_direita);
        k = k+1;
    end
else;
    if nivel_maximo_esquerda > nivel_maximo_direita;
        vector_objectos = objectos_esquerda;
        vector_niveis = sort(niveis_esquerda);
```

```
                    nivel_maximo = nivel_maximo_esquerda;
                    if niveis_direita ~= 0;
                        nivel_aux(k) = nivel_maximo_direita;
                        matriz_objectos(k,1:n_objectos_direita) = objectos_direita;
                        matriz_niveis(k,1:n_objectos_direita-1) = sort(niveis_direita);
                        k = k+1;
                    end;
                else;
                    vector_objectos = objectos_direita;
                    vector_niveis = sort(niveis_direita);
                    nivel_maximo = nivel_maximo_direita;
                    if niveis_esquerda ~= 0;
                        nivel_aux(k) = nivel_maximo_esquerda;
                        matriz_objectos(k,1:n_objectos_esquerda) = objectos_esquerda;
                        matriz_niveis(k,1:n_objectos_esquerda-1) = sort(niveis_esquerda);
                        k = k+1;
                    end;
                end;
            end;
        end;
        nivel_auxiliar = max(nivel_aux);
    end;
    if display
        figure,[h,t,PERM] = dendrogram(T);
        hold on
    end
return;

function [n_objectos_esquerda,n_objectos_direita,objectos_esquerda,objectos_direita,
niveis_esquerda, niveis_direita] = parte_objectos_niveis(vector_objectos,vector_niveis);
    n_objectos = length(vector_objectos);
    n_niveis = length(vector_niveis);
    if n_objectos ~= n_niveis+2;
        disp('Problema!');
    end;
    n_objectos_esquerda = 0;
    while (n_objectos_esquerda = = 0) | (n_objectos_esquerda = = n_objectos);
        n_objectos_esquerda = fix(rand(1)*n_objectos+1);
    end;
    ordem_objectos = randperm(n_objectos);
    for i = 1:n_objectos_esquerda
        objectos_esquerda(i) = vector_objectos(ordem_objectos(i));
    end;
    n_objectos_direita = n_objectos-n_objectos_esquerda;
    for i = 1:n_objectos_direita
        objectos_direita(i) = vector_objectos(ordem_objectos(n_objectos_esquerda+i));
    end;
    ordem_niveis = randperm(n_objectos-2);
    if n_objectos_esquerda = = 1;
        for i = 1:n_niveis;
            niveis_direita(i) = vector_niveis(ordem_niveis(i));
        end;
        niveis_esquerda = 0;
    elseif n_objectos_esquerda = = n_objectos-1;
        for i = 1:n_niveis;
            niveis_esquerda(i) = vector_niveis(ordem_niveis(i));
        end;
```

```
        niveis_direita = 0;
    else;
        for i = 1:n_objectos_esquerda-1;
            niveis_esquerda(i) = vector_niveis(ordem_niveis(i));
        end;
        for i = 1:n_objectos_direita-1;
            niveis_direita(i) = vector_niveis(ordem_niveis(n_objectos_esquerda-1+i));
        end;
    end;
return;

function [vector_objectos,vector_niveis,nivel_aux] =
= find_vectores(nivel_maximo,matriz_objectos,matriz_niveis,nivel_aux,k);
    n = length(matriz_objectos(1,:));
    for i = 1:k-1;
        if nivel_maximo = = nivel_aux(i);
            for j = 1:n-1;
                if matriz_niveis(i,j) ~= 0;
                    vector_niveis(j) = matriz_niveis(i,j);
                end;
            end;
            for j = 1:n;
                if matriz_objectos(i,j) ~= 0;
                    vector_objectos(j) = matriz_objectos(i,j);
                end;
            end;
            nivel_aux(i) = 0;
        end;
    end;
return;
```

Anexo 4— Algoritmo do Método RA

```
function [T] = gera_ra(n,display);
    for i = 1:n;
        vector_aglom(i) = i;
    end;
    T = zeros(n-1,3);
        for i = 1:n-1;
        m = n-i+1;
        [j,k] = inteiros_aleatorios(m);
        temp = vector_aglom(m);
        if vector_aglom(j) < vector_aglom(k);
            T(i,1) = vector_aglom(j);
            T(i,2) = vector_aglom(k);
            T(i,3) = i;
            matriz_auxiliar(i,1:2) = T(i,1:2);
            matriz_ultrametrica(T(i,1),T(i,2)) = i;
            matriz_ultrametrica(T(i,2),T(i,1)) = i;
            vector_aglom(m) = vector_aglom(k);
            vector_aglom(k) = temp;
        else;
            T(i,1) = vector_aglom(k);
            T(i,2) = vector_aglom(j);
            T(i,3) = i;
            matriz_ultrametrica(T(i,1),T(i,2)) = i;
            matriz_ultrametrica(T(i,2),T(i,1)) = i;
            vector_aglom(m) = vector_aglom(j);
            vector_aglom(j) = temp;
        end;
    end;
        if display;
        figure,[h,t,PERM] = dendrogram(T);
        hold on;
    end;
return;

function [j,k] = inteiros_aleatorios(m);
    j = 0;
    k = 0;
    while j = = k;
        j = fix(rand(1)*m+1);
        k = fix(rand(1)*m+1);
    end;
return;
```

Matriz ultramétrica obtida a partir da matriz T

```
function [matriz_ultrametrica] = matriz_ultrametrica_ra(T);
    n = length(T(:,1))+1;
    matriz_ultrametrica = zeros(n);
    matriz_auxiliar = zeros(n-1,n);
    vector_n_elementos_por_classe = zeros(1,n-1);
    matriz_ultrametrica(T(1,1),T(1,2)) = 1;
    matriz_ultrametrica(T(1,2),T(1,1)) = 1;
    matriz_auxiliar(1,1:2) = T(1,1:2);
    vector_n_elementos_por_classe(1) = 2;
    for i = 2:n-1
```

```
        [classe1,classe2,matriz_auxiliar] =
        = encontar_classes(T(i,1),T(i,2),i,matriz_auxiliar,vector_n_elementos_por_classe);
        dimensao1 = length(classe1);
        dimensao2 = length(classe2);
        if classe1 ~= 0;
            if classe2 ~= 0;
                matriz_ultrametrica(classe1,classe2) = i;
                matriz_ultrametrica(classe2,classe1) = i;
                matriz_auxiliar(i,1:dimensao1) = classe1;
                matriz_auxiliar(i,dimensao1+1:dimensao1+dimensao2) = classe2;
                matriz_auxiliar(1,1:dimensao1+dimensao2)=sort(matriz_auxiliar(1,1:dimensao1+dimensao2));
                vector_n_elementos_por_classe(i) = dimensao1+dimensao2;
            else;
                matriz_ultrametrica(classe1,T(i,2)) = i;
                matriz_ultrametrica(T(i,2),classe1) = i;
                matriz_auxiliar(i,1:dimensao1) = classe1;
                matriz_auxiliar(i,dimensao1+1) = T(i,2);
                matriz_auxiliar(i,dimensao1+1) = sort(matriz_auxiliar(i,dimensao1+1));
                vector_n_elementos_por_classe(i) = dimensao1+1;
            end;
        else;
            if classe2 ~= 0;
                matriz_ultrametrica(T(i,1),classe2) = i;
                matriz_ultrametrica(classe2,T(i,1)) = i;
                matriz_auxiliar(i,1) = T(i,1);
                matriz_auxiliar(i,2:dimensao2+1) = classe2;
                matriz_auxiliar(i,1:dimensao2+1) = sort(matriz_auxiliar(i,1:dimensao2+1));
                vector_n_elementos_por_classe(i) = dimensao2+1;
            else
                matriz_ultrametrica(T(i,1),T(i,2)) = i;
                matriz_ultrametrica(T(i,2),T(i,1)) = i;
                matriz_auxiliar(i,1:2) = T(i,1:2);
                vector_n_elementos_por_classe(i) = 2;
            end
        end
    end
return;

function [classe1,classe2,matriz_auxiliar] =
= encontar_classes(a,b,i,matriz_auxiliar,vector_n_elementos_por_classe);
    classe1 = 0;
    classe2 = 0;
    for j = 1:i-1
        if a = = matriz_auxiliar(j,1);
            for k = 1:vector_n_elementos_por_classe(j);
                classe1(k) = matriz_auxiliar(j,k);
            end;
            matriz_auxiliar(j,1) = 0;
        end;
        if b = = matriz_auxiliar(j,1);
            for k = 1:vector_n_elementos_por_classe(j);
                classe2(k) = matriz_auxiliar(j,k);
            end;
            matriz_auxiliar(j,1) = 0;
        end;
    end;
return;
```

Anexo 5— Algoritmo de Geração de Dendrogramas com Parâmetro de Forma

```
function [Matriz_ultrametrica,T] = geracao_parametro_forma(n_objectos,p,display);

return;

function [n_objectos_esquerda,n_objectos_direita,objectos_esquerda,objectos_direita,
niveis_esquerda, niveis_direita] = parte_objectos_niveis(vector_objectos,vector_niveis,p);
    n_objectos = length(vector_objectos);
    n_niveis = length(vector_niveis);
    if n_objectos ~= n_niveis+2;
        disp('Problema!');
    end;
    n_objectos_esquerda = 0;
    while (n_objectos_esquerda = = 0);
        unif = rand(1,n_objectos);
        for I = 1:n_objectos;
            if unif(i) < p;
                 n_objectos_esquerda = n_objectos_esquerda+1;
            end;
        end;
        if n_objectos_esquerda = = n_objectos;
            n_objectos_esquerda = 0;
        end
    end;

return;

function [vector_objectos,vector_niveis,nivel_aux] =
= find_vectores(nivel_maximo,matriz_objectos,matriz_niveis,nivel_aux,k);

return;
```

Anexo 6— Algoritmo da Mistura de dois Dendrogramas

```
function [matriz_ultrametrica,T] = geracao_parametro_forma_mistura(n_objectos_esquerda,
n_objectos_direita,p_esquerda,p_direita,display);
    n_objectos = n_objectos_esquerda+n_objectos_direita;
    matriz_ultrametrica = zeros(n_objectos);
    T = zeros(n_objectos-1,3);
    objectos = randperm(n_objectos);
    níveis = randperm(n_objectos-2);
    objectos_esquerda = objectos(1:n_objectos_esquerda);
    objectos_direita = objectos(n_objectos_esquerda+1:n_objectos);
    niveis_esquerda = niveis(1:n_objectos_esquerda-1);
    niveis_direita = niveis(n_objectos_esquerda:n_objectos-2);
    matriz_ultrametrica(objectos_esquerda,objectos_direita) = n_objectos-1;
    matriz_ultrametrica(objectos_direita,objectos_esquerda) = n_objectos-1;
    minimo_objectos_esquerda = min(objectos_esquerda);
    minimo_objectos_direita = min(objectos_direita);
    T(n_objectos-1,1) = min(minimo_objectos_esquerda,minimo_objectos_direita);
    T(n_objectos-1,2) = max(minimo_objectos_esquerda,minimo_objectos_direita);
    T(n_objectos-1,3) = n_objectos-1;
    [matriz_ultrametrica_esquerda,T_esquerda] =
    = geracao_parametro_forma_dados_vectores(objectos_esquerda,niveis_esquerda,p_esquerda,0);
    [matriz_ultrametrica_direita,T_direita] =
    = geracao_parametro_forma_dados_vectores(objectos_direita,niveis_direita,p_direita,0);
    matriz_ultrametrica(objectos_esquerda,objectos_esquerda) =
    = matriz_ultrametrica_esquerda(objectos_esquerda,objectos_esquerda);
    matriz_ultrametrica(objectos_direita,objectos_direita) =
    = matriz_ultrametrica_direita(objectos_direita,objectos_direita);
    T(niveis_esquerda,1:3)=T_esquerda(niveis_esquerda,1:3);
    T(niveis_direita,1:3)=T_direita(niveis_direita,1:3);
    if display;
        figure,[h,t,PERM] = dendrogram(T);
        hold on;
    end;
return;
```

Anexo 7— Algoritmo *RAP*

```
function [ordem_objectos,matriz_robinson,mr,piramide]=gerar_piramides(n);
    % n - nº de objectos a classificar
    classes=sort(randperm(n));
    componentes_conexas=zeros(n,n);
    componentes_conexas(:,1)=classes';
    piramide=zeros(n,n+1);
    piramide(:,2)=classes';
    ordem_objectos=classes;
    matriz_robinson=zeros(n);
    k=n;
    nivel=1;
    m=n+1;
    classes_ja_agregadas=zeros(n,2);
    classes_ja_agregadas(:,1)=classes';
    while piramide(k,n+1)==0;
        [classe_1,posicao_1]=primeira_classe(classes);
        [classes_agregaveis]=classes_para_agregar(classes,classe_1,posicao_1,
        ordem_objectos,piramide,componentes_conexas,classes_ja_agregadas,n);
        [classe_2]=segunda_classe_agregar(classes_agregaveis);
        [piramide,ordem_objectos,componentes_conexas,k]=construir_piramide(piramide,k,nivel,
        classe_1,classe_2,ordem_objectos,componentes_conexas,classes_ja_agregadas,n);
        [matriz_robinson,nivel]=preencher_matriz_robinson(matriz_robinson,piramide,k,nivel,n);
        [classes,classes_ja_agregadas,m]=classes_activas(classes,classes_ja_agregadas,
        classe_1,classe_2,componentes_conexas,ordem_objectos,piramide,m,k,n);
    end;
     disp('--------------- Geraçao Aleatoria de Piramides ---------------');
     disp('Piramide');
     disp(piramide);
     disp('Ordem do Objectos');
     disp(ordem_objectos);
     disp('Matrizes de Robinson');
     mr=matriz_robinson(ordem_objectos,ordem_objectos);
     disp(matriz_robinson);
     disp('-------------');
     disp(mr);
     disp('----------------------------------------------------------------');
return;

function [classe_1,posicao_1]=primeira_classe(classes);
    tamanho=length(classes);
    posicao_1=fix(rand(1)*tamanho+1);
    classe_1=classes(posicao_1);
return;

function [classes_agregaveis_2]=classes_para_agregar(classes,classe_1,posicao_1,
ordem_objectos,piramide,componentes_conexas,classes_ja_agregadas,n);
    tamanho=length(classes);
    if posicao_1==1;
        classes_agregaveis(1:tamanho-1)=classes(2:tamanho);
    elseif posicao_1==tamanho;
        classes_agregaveis(1:tamanho-1)=classes(1:tamanho-1);
    else;
        classes_agregaveis(1:posicao_1-1)=classes(1:posicao_1-1);
        classes_agregaveis(posicao_1:tamanho-1)=classes(posicao_1+1:tamanho);
    end;
```

```
% componentes conexas a qual pertencem as classes
objectos_classe_1=nonzeros(piramide(classe_1,2:n+1))';
numero_componentes_conexas=length(componentes_conexas(:,1));
for i=1:numero_componentes_conexas;
    if find(objectos_classe_1(1)==componentes_conexas(i,:))~=0;
        componente_conexa_c1=i;
        break;
    end;
end;
for j=1:tamanho-1;
    objectos_das_classes_agregaveis(j,1:n)=piramide(classes_agregaveis(j),2:n+1);
    for i=1:numero_componentes_conexas;
        if find(objectos_das_classes_agregaveis(j,1)==componentes_conexas(i,:))~=0;
            componentes_conexas_das_classes_agregaveis(j)=i;
            break;
        end;
    end;
end;
a=1;
tamanho_c1=length(objectos_classe_1);
min_c1=objectos_classe_1(1);
max_c1=objectos_classe_1(tamanho_c1);
objectos_da_componente_conexa_c1=nonzeros(componentes_conexas(componente_conexa_c1,:))';
tamanho_componentes_conexas_c1=length(objectos_da_componente_conexa_c1);
min_componentes_conexas_c1=objectos_da_componente_conexa_c1(1);
max_componentes_conexas_c1=objectos_da_componente_conexa_c1(tamanho_componentes_conexas_c1);
%-----------------------------------------------------------------------

% determinar o no maximal da classe 1
if tamanho_componentes_conexas_c1 > 2;
    classes_cc_c1(1)=classe_1;
    pos_classes_cc_c1=find(componentes_conexas_das_classes_agregaveis==componente_conexa_c1);
    tamanho_pos_classes_cc_c1=length(pos_classes_cc_c1);
    tamanho_classes_cc_c1=tamanho_pos_classes_cc_c1+1;
    classes_cc_c1(2:tamanho_classes_cc_c1)=classes_agregaveis(pos_classes_cc_c1);
    s=1;
    for i=1:tamanho_classes_cc_c1;
        pos_classe=find(classes_cc_c1(i)==classes_ja_agregadas(:,1)');
        if classes_ja_agregadas(pos_classe,2)==0;
            classes_maximais(s)=classes_cc_c1(i);
            objectos_classe_maximal(s,1:n)=piramide(classes_maximais(s),2:n+1);
            obs_classe_maximal_s=nonzeros(objectos_classe_maximal(s,:))';
            min_obs_cm(s)=obs_classe_maximal_s(1);
            s=s+1;
        end;
    end;
    tamanho_classes_maximais=length(classes_maximais);
    t=1;
    for j=1:tamanho_classes_maximais;
        posc_min(t)=find(min_obs_cm(j)==ordem_objectos);
        t=t+1;
    end;
    [posc_min_ord,vector_posc]=sort(posc_min);
    classes_maximais_ord=classes_maximais(vector_posc);
    classe_maximal_1=0;
    classe_maximal_1_direita=0;
```

```
    classe_maximal_1_esquerda=0;
    posc_1=-2;
    for j=1:tamanho_classes_maximais;
        objs_cm1=nonzeros(objectos_classe_maximal(j,:))';
        tamanho_objs_cm1=length(objs_cm1);
        min_objs_cm1=objs_cm1(1);
        max_objs_cm1=objs_cm1(tamanho_objs_cm1);
        if min_c1==min_objs_cm1;
            if max_c1==max_objs_cm1;
                classe_maximal_1=classes_maximais(j);
                posc_1=find(classe_maximal_1==classes_maximais_ord);
                if classes_maximais(j)~=classe_1;
                    disp('erro nas classess maximais');
                end;
            else;
                classe_maximal_1_direita=classes_maximais(j);
                posc_1=find(classe_maximal_1_direita==classes_maximais_ord);
            end;
        elseif max_c1==max_objs_cm1;
            classe_maximal_1_esquerda=classes_maximais(j);
            posc_1=find(classe_maximal_1_esquerda==classes_maximais_ord);
        end;
    end;
end;
%-------------------------------------------------------------------------

for i=1:tamanho-1;
    objectos_da_cc_agregavel=
    =nonzeros(componentes_conexas(componentes_conexas_das_classes_agregaveis(i),:))';
    tamanho_da_cc_agregavel=length(objectos_da_cc_agregavel);
    min_da_cc_agregavel=objectos_da_cc_agregavel(1);
    max_da_cc_agregavel=objectos_da_cc_agregavel(tamanho_da_cc_agregavel);

    % A classe 1 pertence a mesma componente conexa das classes ...
    if componentes_conexas_das_classes_agregaveis(i)==componente_conexa_c1;
        if tamanho_componentes_conexas_c1 > 2;
            objectos_da_classe_agregavel=nonzeros(piramide(classes_agregaveis(i),2:n+1))';
            tamanho_da_classe_agregavel=length(objectos_da_classe_agregavel);
            min_classe_agregavel=objectos_da_classe_agregavel(1);
            max_classe_agregavel=objectos_da_classe_agregavel(tamanho_da_classe_agregavel);
            classe_maximal_2=-1;
            classe_maximal_2_direita=-1;
            classe_maximal_2_esquerda=-1;
            posc_2=-2;
            for j=1:tamanho_classes_maximais;
                objs_cm2=nonzeros(objectos_classe_maximal(j,:))';
                tamanho_objs_cm2=length(objs_cm2);
                min_objs_cm2=objs_cm2(1);
                max_objs_cm2=objs_cm2(tamanho_objs_cm2);
                if min_classe_agregavel==min_objs_cm2;
                    if max_classe_agregavel==max_objs_cm2;
                        classe_maximal_2=classes_maximais(j);
                        posc_2=find(classe_maximal_2==classes_maximais_ord);
                        if classes_maximais(j)~=classes_agregaveis(i);
                            disp('erro nas classess maximais');
                        end;
                    else;
```

```
                    classe_maximal_2_direita=classes_maximais(j);
                    posc_2=find(classe_maximal_2_direita==classes_maximais_ord);
                end;
            elseif max_classe_agregavel==max_objs_cm2;
                classe_maximal_2_esquerda=classes_maximais(j);
                posc_2=find(classe_maximal_2_esquerda==classes_maximais_ord);
            end;
        end;
        %-------------------------------------------------------
        if posc_1==posc_2-1;
            if classe_maximal_1~=0;
                if classe_maximal_2~=-1;
                    classes_agregaveis_2(a)=classes_agregaveis(i);
                    a=a+1;
                elseif classe_maximal_2_direita~=-1;
                    classes_agregaveis_2(a)=classes_agregaveis(i);
                    a=a+1;
                end;
            elseif classe_maximal_1_esquerda~=0;
                if classe_maximal_2~=-1;
                    classes_agregaveis_2(a)=classes_agregaveis(i);
                    a=a+1;
                elseif classe_maximal_2_direita~=-1;
                    classes_agregaveis_2(a)=classes_agregaveis(i);
                    a=a+1;
                end;
            end;
        elseif posc_1==posc_2+1;
            if classe_maximal_1~=0;
                if classe_maximal_2~=-1;
                    classes_agregaveis_2(a)=classes_agregaveis(i);
                    a=a+1;
                elseif classe_maximal_2_esquerda~=-1;
                    classes_agregaveis_2(a)=classes_agregaveis(i);
                    a=a+1;
                end;
            elseif classe_maximal_1_direita~=0;
                if classe_maximal_2~=-1;
                    classes_agregaveis_2(a)=classes_agregaveis(i);
                    a=a+1;
                elseif classe_maximal_2_esquerda~=-1;
                    classes_agregaveis_2(a)=classes_agregaveis(i);
                    a=a+1;
                end;
            end;
        end;
    end;
%-----------------------------------------------------------------
else; %componentes conexas diferentes
    objectos_da_classe_agregavel=nonzeros(piramide(classes_agregaveis(i),2:n+1))';
    tamanho_da_classe_agregavel=length(objectos_da_classe_agregavel);
    min_classe_agregavel=objectos_da_classe_agregavel(1);
    max_classe_agregavel=objectos_da_classe_agregavel(tamanho_da_classe_agregavel);
    if (min_c1==min_componentes_conexas_c1)|(max_c1==max_componentes_conexas_c1);
        if (min_classe_agregavel==min_da_cc_agregavel)|(max_classe_agregavel==max_da_cc_agregavel);
            classes_agregaveis_2(a)=classes_agregaveis(i);
            a=a+1;
```

```
                end;
            end;
        end;
    end;
return;

function [classe_2]=segunda_classe_agregar(classes_agregaveis);
    tamanho=length(classes_agregaveis);
    posicao_2=fix(rand(1)*tamanho+1);
    classe_2=classes_agregaveis(posicao_2);
return;

function [piramide,ordem_objectos,componentes_conexas_act,k]=construir_piramide(piramide,k,nivel,
classe_1,classe_2,ordem_objectos,componentes_conexas,classes_ja_agregadas,n);
    objectos_c1=nonzeros(piramide(classe_1,2:n+1))';
    objectos_c2=nonzeros(piramide(classe_2,2:n+1))';
    t1=length(objectos_c1);
    t2=length(objectos_c2);
    min_c1=objectos_c1(1);
    max_c1=objectos_c1(t1);
    min_c2=objectos_c2(1);
    max_c2=objectos_c2(t2);
    n_cc=length(componentes_conexas(:,1));
    for i=1:n_cc;
        if find(objectos_c1(1)==componentes_conexas(i,:))~=0;
            cc_c1=i;
            break;
        end;
    end;
    for i=1:n_cc;
        if find(objectos_c2(1)==componentes_conexas(i,:))~=0;
            cc_c2=i;
            break;
        end;
    end;

    % actualizaçao das componentes conexas
    if cc_c1==cc_c2; %as duas classes a agregar pertencem a mesma componente conexa
        componentes_conexas_act=componentes_conexas;
        posc_min_c1=find(min_c1==ordem_objectos);
        posc_max_c1=find(max_c1==ordem_objectos);
        posc_min_c2=find(min_c2==ordem_objectos);
        posc_max_c2=find(max_c2==ordem_objectos);
    else; %as duas classes a agregar nao pertencem a mesma componente conexa
        objectos_cc_c1=nonzeros(componentes_conexas(cc_c1,:))';
        objectos_cc_c2=nonzeros(componentes_conexas(cc_c2,:))';
        t1_cc=length(objectos_cc_c1);
        t2_cc=length(objectos_cc_c2);
        min_cc_c1=objectos_cc_c1(1);
        max_cc_c1=objectos_cc_c1(t1_cc);
        min_cc_c2=objectos_cc_c2(1);
        max_cc_c2=objectos_cc_c2(t2_cc);
        if cc_c1 < cc_c2;
            if max_c1==max_cc_c1;
                if min_c2==min_cc_c2;
                    componentes_conexas(cc_c1,t1_cc+1:t1_cc+t2_cc)=objectos_cc_c2;
                    if cc_c2==n_cc;
```

```
                componentes_conexas_act(1:n_cc-1,:)=componentes_conexas(1:n_cc-1,:);
            else;
                componentes_conexas_act(1:cc_c2-1,:)=componentes_conexas(1:cc_c2-1,:);
                componentes_conexas_act(cc_c2:n_cc-1,:)=componentes_conexas(cc_c2+1:n_cc,:);
            end;
        else; %max_c2==max_cc_c2
            %inverter a ordem dos objectos da componente conexa da classe 2
            for i=1:t2_cc;
                objectos_cc_c2_inv(i)=objectos_cc_c2(t2_cc-i+1);
            end;
            componentes_conexas(cc_c1,t1_cc+1:t1_cc+t2_cc)=objectos_cc_c2_inv;
            if cc_c2==n_cc;
                componentes_conexas_act(1:n_cc-1,:)=componentes_conexas(1:n_cc-1,:);
            else;
                componentes_conexas_act(1:cc_c2-1,:)=componentes_conexas(1:cc_c2-1,:);
                componentes_conexas_act(cc_c2:n_cc-1,:)=componentes_conexas(cc_c2+1:n_cc,:);
            end;
        end;
    else; %min_c1==min_cc_c1
        if min_c2==min_cc_c2;
            %inverter a ordem dos objectos da componente conexa da classe 1
            for i=1:t1_cc;
                objectos_cc_c1_inv(i)=objectos_cc_c1(t1_cc-i+1);
            end;
            componentes_conexas(cc_c1,1:t1_cc)=objectos_cc_c1_inv;
            componentes_conexas(cc_c1,t1_cc+1:t1_cc+t2_cc)=objectos_cc_c2;
            if cc_c2==n_cc;
                componentes_conexas_act(1:n_cc-1,:)=componentes_conexas(1:n_cc-1,:);
            else;
                componentes_conexas_act(1:cc_c2-1,:)=componentes_conexas(1:cc_c2-1,:);
                componentes_conexas_act(cc_c2:n_cc-1,:)=componentes_conexas(cc_c2+1:n_cc,:);
            end;
        else; %max_c2==max_cc_c2
            if cc_c2==n_cc;
                componentes_conexas_act(1:n_cc-1,:)=componentes_conexas(1:n_cc-1,:);
            else;
                componentes_conexas_act(1:cc_c2-1,:)=componentes_conexas(1:cc_c2-1,:);
                componentes_conexas_act(cc_c2:n_cc-1,:)=componentes_conexas(cc_c2+1:n_cc,:);
            end;
            componentes_conexas_act(cc_c1,1:t2_cc)=objectos_cc_c2;
            componentes_conexas_act(cc_c1,t2_cc+1:t2_cc+t1_cc)=objectos_cc_c1;
        end;
    end;
else; %cc_c2 < cc_c1
    if max_c2==max_cc_c2;
        if min_c1==min_cc_c1;
            componentes_conexas(cc_c2,t2_cc+1:t2_cc+t1_cc)=objectos_cc_c1;
            if cc_c1==n_cc;
                componentes_conexas_act(1:n_cc-1,:)=componentes_conexas(1:n_cc-1,:);
            else;
                componentes_conexas_act(1:cc_c1-1,:)=componentes_conexas(1:cc_c1-1,:);
                componentes_conexas_act(cc_c1:n_cc-1,:)=componentes_conexas(cc_c1+1:n_cc,:);
            end;
        else; %max_c1==max_cc_c1
            %inverter a ordem dos objectos da componente conexa da classe 1
            for i=1:t1_cc;
                objectos_cc_c1_inv(i)=objectos_cc_c1(t1_cc-i+1);
```

```
                end;
                componentes_conexas(cc_c2,t2_cc+1:t2_cc+t1_cc)=objectos_cc_c1_inv;
                if cc_c1==n_cc;
                    componentes_conexas_act(1:n_cc-1,:)=componentes_conexas(1:n_cc-1,:);
                else;
                    componentes_conexas_act(1:cc_c1-1,:)=componentes_conexas(1:cc_c1-1,:);
                    componentes_conexas_act(cc_c1:n_cc-1,:)=componentes_conexas(cc_c1+1:n_cc,:);
                end;
            end;
        else; %min_c2==min_cc_c2
            if min_c1==min_cc_c1;
                %inverter a ordem dos objectos da componente conexa da classe 2
                for i=1:t2_cc;
                    objectos_cc_c2_inv(i)=objectos_cc_c2(t2_cc-i+1);
                end;
                componentes_conexas(cc_c2,1:t2_cc)=objectos_cc_c2_inv;
                componentes_conexas(cc_c2,t2_cc+1:t2_cc+t1_cc)=objectos_cc_c1;
                if cc_c1==n_cc;
                    componentes_conexas_act(1:n_cc-1,:)=componentes_conexas(1:n_cc-1,:);
                else;
                    componentes_conexas_act(1:cc_c1-1,:)=componentes_conexas(1:cc_c1-1,:);
                    componentes_conexas_act(cc_c1:n_cc-1,:)=componentes_conexas(cc_c1+1:n_cc,:);
                end;
            else; %max_c1==max_cc_c1
                if cc_c1==n_cc;
                    componentes_conexas_act(1:n_cc-1,:)=componentes_conexas(1:n_cc-1,:);
                else;
                    componentes_conexas_act(1:cc_c1-1,:)=componentes_conexas(1:cc_c1-1,:);
                    componentes_conexas_act(cc_c1:n_cc-1,:)=componentes_conexas(cc_c1+1:n_cc,:);
                end;
                componentes_conexas_act(cc_c2,1:t1_cc)=objectos_cc_c1;
                componentes_conexas_act(cc_c2,t1_cc+1:t1_cc+t2_cc)=objectos_cc_c2;
            end;
        end;
    end;
end;

% actualizar a ordem dos objectos
if cc_c1~=cc_c2;
    n_cc_act=length(componentes_conexas_act(:,1));
    t_inicial=1;
    for i=1:n_cc_act;
        ind=nonzeros(componentes_conexas_act(i,:))';
        t=length(ind);
        ordem_objectos(t_inicial:t_inicial+t-1)=ind;
        t_inicial=t_inicial+t;
    end;
end;

% reuniao das duas classes
if cc_c1==cc_c2;
    if posc_min_c1 < posc_min_c2;
        objectos(1:posc_max_c2-posc_min_c1+1)=ordem_objectos(posc_min_c1:posc_max_c2);
    else;
        objectos(1:posc_max_c1-posc_min_c2+1)=ordem_objectos(posc_min_c2:posc_max_c1);
    end;
else;
```

```
        uniao(1:t1)=objectos_c1;
        uniao(t1+1:t1+t2)=objectos_c2;
        uniao_ord=sort(uniao);
        objectos(1)=uniao_ord(1);
        a=0;
        for i=2:t1+t2;
            if uniao_ord(i)~=uniao_ord(i-1);
                objectos(i-a)=uniao_ord(i);
            else;
                a=a+1;
            end;
        end;
    end;
    tamanho=length(objectos);
    k=k+1;
    piramide(k,1)=nivel;
    piramide(k,2:tamanho+1)=objectos;

    % actualizar a ordem da piramide segundo a ordem dos objectos
    if cc_c1~=cc_c2;
        tamanho_piramide=length(piramide(:,1));
        for i=n+1:tamanho_piramide;
            objectos_piramide=nonzeros(piramide(i,2:n+1))';
            objectos_piramide_ord=ordenar_objectos(objectos_piramide,ordem_objectos);
            top=length(objectos_piramide_ord);
            piramide(i,2:top+1)=objectos_piramide_ord;
        end;
    end;
return;

function [matriz_robinson,nivel]=preencher_matriz_robinson(matriz_robinson,piramide,k,nivel,n);
    if piramide(k,1)~=nivel;
        disp('erro no nivel');
    end;
    objectos=nonzeros(piramide(k,2:n+1))';
    t=length(objectos);
    for i=1:t-1;
        for j=i+1:t;
            if matriz_robinson(objectos(i),objectos(j))==0;
                matriz_robinson(objectos(i),objectos(j))=nivel;
                matriz_robinson(objectos(j),objectos(i))=nivel;
            end;
        end;
    end;
    nivel=nivel+1;
return;

function [classes,classes_ja_agregadas,m]=classes_activas(classes,classes_ja_agregadas,
classe_1,classe_2,componentes_conexas,ordem_objectos,piramide,m,k,n);
    classes(m)=k;
    classes_ja_agregadas(m,1)=k;
    m=m+1;
    t_classes=length(classes);

    %excluir classes que ja foram agregadas duas vezes
    pos1=find(classes_ja_agregadas(:,1)==classe_1);
    if classes_ja_agregadas(pos1,2)==0;
```

```
    classes_ja_agregadas(pos1,2)=1;
else;
    if pos1==1;
        classes_ja_agregadas_act1(1:t_classes-1,:)=classes_ja_agregadas(2:t_classes,:);
    elseif pos1==t_classes;
        classes_ja_agregadas_act1(1:t_classes-1,:)=classes_ja_agregadas(1:t_classes-1,:);
    else;
        classes_ja_agregadas_act1(1:pos1-1,:)=classes_ja_agregadas(1:pos1-1,:);
        classes_ja_agregadas_act1(pos1:t_classes-1,:)=classes_ja_agregadas(pos1+1:t_classes,:);
    end;
    classes_ja_agregadas=classes_ja_agregadas_act1;
    t_classes=t_classes-1;
    m=m-1;
end;
pos2=find(classes_ja_agregadas(:,1)==classe_2);
if classes_ja_agregadas(pos2,2)==0;
    classes_ja_agregadas(pos2,2)=1;
else;
    if pos2==1;
        classes_ja_agregadas_act2(1:t_classes-1,:)=classes_ja_agregadas(2:t_classes,:);
    elseif pos2==t_classes;
        classes_ja_agregadas_act2(1:t_classes-1,:)=classes_ja_agregadas(1:t_classes-1,:);
    else;
        classes_ja_agregadas_act2(1:pos2-1,:)=classes_ja_agregadas(1:pos2-1,:);
        classes_ja_agregadas_act2(pos2:t_classes-1,:)=classes_ja_agregadas(pos2+1:t_classes,:);
    end;
    classes_ja_agregadas=classes_ja_agregadas_act2;
    t_classes=t_classes-1;
    m=m-1;
end;
classes=classes_ja_agregadas(:,1)';
%-------------------------------------------------------------------------

%excluir as classes que sao nos interiores

% componentes conexas a qual pertencem as classes
numero_componentes_conexas=length(componentes_conexas(:,1));
tamanho_classes=length(classes);
for j=1:tamanho_classes;
    objs_das_classes=nonzeros(piramide(classes(j),2:n+1))';
    for i=1:numero_componentes_conexas;
        if find(objs_das_classes(1)==componentes_conexas(i,:))~=0;
            componentes_conexas_das_classes(j)=i;
            break;
        end;
    end;
end;
%-------------------------------------------------------------------------
a=1;
pos=0;
for k=1:numero_componentes_conexas;
    posicoes_cc_k=find(componentes_conexas_das_classes==k);
    tamanho_cc_k=length(posicoes_cc_k);
    for i=1:tamanho_cc_k-1;
        objcetos_classe1=nonzeros(piramide(classes(posicoes_cc_k(i)),2:n+1))';
        tamanho_classe1=length(objcetos_classe1);
        min_classe1=objcetos_classe1(1);
```

```
            max_classe1=objcetos_classe1(tamanho_classe1);
            for j=i+1:tamanho_cc_k;
                objcetos_classe2=nonzeros(piramide(classes(posicoes_cc_k(j)),2:n+1))';
                tamanho_classe2=length(objcetos_classe2);
                if tamanho_classe2 >= tamanho_classe1+2;
                    if find(objcetos_classe2==min_classe1)~=0;
                        if find(objcetos_classe2==max_classe1)~=0;
                            if min_classe1~=objcetos_classe2(1) && max_classe1~=
                            objcetos_classe2(tamanho_classe2);
                                pos(a)=posicoes_cc_k(i);
                                a=a+1;
                            end;
                        end;
                    end;
                end;
            end;
        end;
    end;
    if pos~=0;
        [classes_ja_agregadas,t_classes,m]=
        actualizar_classes_ja_agregadas(classes_ja_agregadas,pos,t_classes,m);
        classes=classes_ja_agregadas(:,1)';
    end;
    %-------------------------------------------------------------------------

    %excluir as classes que nao possuem nenhuma classe para serem agregadas
    if numero_componentes_conexas==1;
        a=1;
        posc_ret=0;
        min_cc=componentes_conexas(1);
        max_cc=componentes_conexas(n);
        posc=find(classes_ja_agregadas(:,2)==1);
        tamanho_posc=length(posc);
        for i=1:tamanho_posc;
            objs=nonzeros(piramide(classes(posc(i)),2:n+1))';
            tamanho_objs=length(objs);
            min_objs=objs(1);
            max_objs=objs(tamanho_objs);
            if min_objs==min_cc || max_objs==max_cc;
                posc_ret(a)=posc(i);
                a=a+1;
            end;
        end;
        if posc_ret~=0;
            [classes_ja_agregadas,t_classes,m]=
            actualizar_classes_ja_agregadas(classes_ja_agregadas,posc_ret,t_classes,m);
            classes=classes_ja_agregadas(:,1)';
        end;
    end;
return;


%-------------- funçoes auxiliares ----------------------------------------
function [vector_ultrmetrico,vector_ultrmetrico_ord]=vector_ult(matriz_ultrametrica);
[n,d]=size(matriz_ultrametrica);
vector_ultrmetrico=zeros(1,(n^2-n)/2);
k=1;
```

```
for i=1:n-1;
    for j=i+1:n;
        vector_ultrmetrico(1,k)=matriz_ultrametrica(i,j);
        k=k+1;
    end;
end;
vector_ultrmetrico_ord=sort(vector_ultrmetrico);
return;

function [classes_ja_agregadas,t_classes,m]=actualizar_classes_ja_agregadas(classes_ja_agregadas,pos,t_classes,m);
    tamanho_pos=length(pos);
    for i=1:tamanho_pos;
        if pos(i)==1;
            classes_ja_agregadas_act(1:t_classes-1,1:2)=classes_ja_agregadas(2:t_classes,:);
        elseif pos(i)==t_classes;
            classes_ja_agregadas_act(1:t_classes-1,1:2)=classes_ja_agregadas(1:t_classes-1,:);
        else;
            classes_ja_agregadas_act(1:pos(i)-1,1:2)=classes_ja_agregadas(1:pos(i)-1,:);
            classes_ja_agregadas_act(pos(i):t_classes-1,1:2)=classes_ja_agregadas(pos(i)+1:t_classes,:);
        end;
        classes_ja_agregadas=classes_ja_agregadas_act;
        classes_ja_agregadas_act=0;
        t_classes=t_classes-1;
        m=m-1;
        pos=pos-1;
    end;
return;
```

Anexo 8— Identificação Topológica de Pirâmides

Identificação da proporção das topologias para 3 nós terminais

```
function [prob_top_1,prob_top_2]=topologias_3_piramide(m);
    % m - numero de piramides a gerar com tres nos terminais
    top_1=0;
    top_2=0;
    for i=1:m;
        [ordem_objectos,matriz_robinson,piramide]=gerar_piramides(3);
        [vector_robinson,vector_robinson_ord]=vector_ult(matriz_robinson);
        if vector_robinson_ord(3)==3;
            top_1=top_1+1;
        else;
            top_2=top_2+1;
        end;
    end;
    prob_top_1=top_1/m;
    prob_top_2=top_2/m;
return;
```

Identificação da proporção das topologias para 4 nós terminais

```
function [p_t1,p_t2,p_t3,p_t4,p_t5,p_t6,p_t7,p_t8]=topologias_4_piramide(m);
    % m - numero de piramides a gerar com quatro nos terminais
    top_1=0;
    top_2=0;
    top_3=0;
    top_4=0;
    top_5=0;
    top_6=0;
    top_7=0;
    top_8=0;
    for i=1:m;
        [ordem_objectos,matriz_robinson,piramide]=gerar_piramides(4);
        [vector_robinson,vector_robinson_ord]=vector_ult(matriz_robinson);
        if vector_robinson_ord(6)==6;
            top_8=top_8+1;
        elseif vector_robinson_ord(6)==3;
            if vector_robinson_ord(3)==3;
                top_1=top_1+1;
            else;
                top_2=top_2+1;
            end;
        elseif vector_robinson_ord(6)==5;
            if vector_robinson_ord(5)==5;
                top_6=top_6+1;
            else;
                top_7=top_7+1;
            end;
        else;
            if vector_robinson_ord(4)==4;
                top_3=top_3+1;
            elseif vector_robinson_ord(5)==3;
```

```
                top_5=top_5+1;
            else;
                top_4=top_4+1;
            end;
        end;
    end;
    p_t1=top_1/m;
    p_t2=top_2/m;
    p_t3=top_3/m;
    p_t4=top_4/m;
    p_t5=top_5/m;
    p_t6=top_6/m;
    p_t7=top_7/m;
    p_t8=top_8/m;
return;
```

Bibliografia

[1] Bacelar-Nicolau, H. (1980). *Contribuições ao Estudo dos Coeficientes de Comparação em Análise Classificatória*. Tese de Douturamento, Lisboa.

[2] Barthélemy, J.P.; Guénoche, A. (1988); *Les Arbres et les Représentations des Proximités*, Masson

[3] Bandelt, H., Dress, A. (1989), *Weak hierarchies associated with similarity measures: an additive clustering technique Bull*. Math. Biol. 51, 133-166.

[4] Bertrand, P. (1986). *Etude de la représentation pyramidale*. Thèse de 3 cycle, Université Paris IX-Dauphine.

[5] Bertrand, P. (1995). *Strutural properties of pyramidal clustering*, DIMACS Series in Discrete Mathematics and Theoretical Computer Science, Volume 19.

[6] Bertrand, P. (2002). *Les 2-3 Hiérarchies: une structure de classification pyramidale parcimonieuse*. Actes du IX ème Congrès de la Société Francophone de Classification, 16-18 Septembre, Toulouse, France.

[7] Bertrand, P., Diday, E. (1990), *Une Généralisation des Arbes Hiérarchiques: les Representations Pyramidales*, Revue Statisque Appliquée, XXXVIII (3), p. 53-78.

[8] Bertrand, P., Diday, E. (1991), *An Extension of Hierarchical Clustering: the Pyramidal Presentation*, in Pattern Recognition in Practice II, Eds E.S. Gelsema and I.N. Kanal, Amsterdam: North-Holland, p. 411-424.

[9] Brito, P. (1991). *Analyse de données symboliques: Pyramides d'heritage*. Thèse de doctorat, Université Paris IX-Dauphine.

[10] Chelcea S., Bertrand, P. , Trousse, B. (2002). *Un Nouvel Algorithme de Classification Ascendante 2-3 Hiérarchique*, INRIA, Equipe AxIS, BP93, 06902 Sophia-Antipolis Cedex, France.

[11] Diatta, J., Fichet, B. (1994). From Aspresjan hierarchies and Bandelt-Dress Weak-hierarchies to Quasi-hierarchies. New approaches in classification and data analysis. Editors: Diday et al., Springer Verlag.

[12] Diday, E. (1987). Une représentation visuelle des classes empiétantes: les pyramides. Rapport de recherche I.N.R.I.A. nº. 291, Rocquencourt, France.

[13] Diday, E. (1987). Orders and overlapping clusters by pyramids. Rapport de recherche I.N.R.I.A. nº. 730, Rocquencourt, France.

[14] Diday, E., Lemaire, J., Pouget, J., Testu, F. (1982). *Eléments d'analyse des donnés*, Dunod, Paris.

[15] Durand, C. (1988). Une Appoximation de Robinson Inferieure Maximal. Université de Provence, U.E.R. de Mathématiques.

[16] Durand, C. (1989). *Ordres et Graphes Pseudo-Hiérarchiques: Théorie et Optimisation Algorithmique*. Thèse de Doctorat, Université de Provence - Saint Charles.

[17] Fichet, B. (1984). Sur une extension de la notation de hiérarchie et son équivalence avec quelques matrices de Robinson. Dans Actes des Journées de statistique de la Grand Motte.

[18] Flajolet, P., Zimmerman, P, Van Cutsem, B. (1994). A calculus for the random generation of labelled combinatorial structures. *Theorectical Computer Science*, Vol.132, p. 1-35.

[19] Furnas, G.W. (1984). The generation of random, binary unordered trees. *Journal of Classification*, Vol.1, p. 187-233.

[20] Gordon, A.D. (1987). A Review of Hierarchical Classification, *Journal of the royal Statistical Society*, A, 150, p. 119-137.

[21] Gordon, A.D. (1999). *Classification*. 2nd ed, Chapman and Hall, London.

[22] Harding, E.F. (1971). *The Probabilities of Rooted Tree-shapes Generates by Random Bifurcation*, Adv.Appl.Prob., 3, p.44-77.

[23] Jambu, M. e Lebeaux, M.O. (1978). *Classification Automatique pour l'Analyse des Données, Tome I: Méthodes et Algorithmes*. Dunod, Paris.

[24] Lance, G.N. e Williams, W.T. (1966). A Generalised Sorting Strategy for Computer Classifications, *Nature*, p. 212-218.

[25] Lapointe, F., Legendre, P. (1991). The generation of random ultrametric matrices representing dendrograms. *Journal of Classification*, Vol.8, p. 177-200.

[26] Lapointe, F.J. (2000); *How to Account for Reticulation Events in Phylogenetic Analysis: A Comparison of Distance-Based Methods*, Journal of Classification, 17, 175-184.

[27] Mfoumoune, E. (1998). *Les aspects algorithmiques de la Classification Ascendante Pyramidale et Incrémenntale*. Thèse de Docteur en Sciences, Université Paris IX-Dauphine.

[28] Murtagh, F. (1983); *A Probability Theory of Hierarchic Clustering Using Random Dendrograms*, J. Statist Comput. Simul., Vol. 18, 145-157.

[29] Murtagh, F. (1984); *Counting Dendrograms: A Survey*, Discrete Applied Mathematics, 7, 191-199.

[30] Oden, N. L.; Shao, K. T. (1984); *An Algorithm to Equiprobably Generate All Directed Trees with k Labeled Terminal Nodes and Unlabeled Interior Nodes*, Bulletin of Mathematical Biology, 46, 379-387.

[31] Pestana, D. D.; Velosa, S. F. (2002); *Introdução à Probabilidade e à Estatística – Volume I*, Fundação Calouste Gulbenkian, Lisboa.

[32] Podani, J. (2000). Simulation of random dendrograms and comparison tests: some comments. *Journal of Classification*, Vol.17, p. 123-142.

[33] Quiroz, A.J. (1989); *Fast Random Generation of Binary, t-ary, and Other Types of Trees*, Journal of Classification, 6, 223-231.

[34] Rodríguez, O., Diday, E.. Pyramidal Clustering Algorithms, in ISO-3D Project, Universite Paris IX-Dauphine.

[35] Saporta, G. (1978). *Théories et Méthodes de la Statistique*, EditionsTechnip, Paris.

[36] Sibson, R. (1972); *Order Invariant Methods for Data Analysis*, Journal of the Royal Statistical Society, B34, 311-349.

[37] Sousa, F. (2000). *Novas metodologias e validação em classificação hierárquica ascendente*. Dissertação de doutoramento, Faculdade de Ciências e Tecnologia, Universidade Nova de Lisboa.

[38] Tendeiro, J. (2005). *Comparação de dendrogramas: obtenção de distribuições empíricas de alguns coeficientes*. Dissertação de mestrado, Faculdade de Engenharia, Universidade do Porto.

[39] Van Cutsem, B. (1996). Combinatorial structures and structures for classification. *Computacional Statistics & Data Analysis*, Vol.23, p. 169-188.