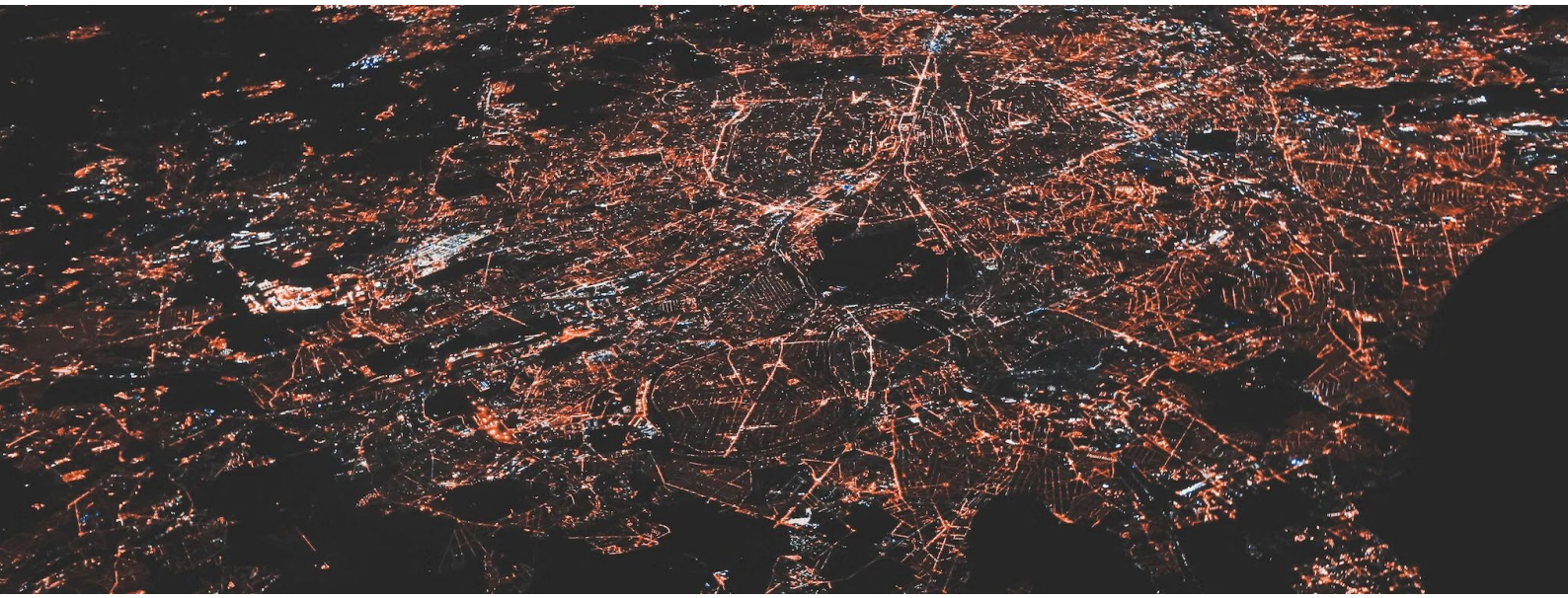
INSTITUTO UNIVERSITÁRIO DE LISBOA

Ano Letivo 2023/2024

16 de janeiro de 2024

Redes Aleatórias & Comunidades

Trabalho 2



UC Análise de Redes

Licenciatura Ciência de Dados

CDC1 e CDC2

Docentes

Maria João Lopes

Catarina Nunes

André Silvestre Nº104532

Eliane Gabriel Nº103303

Maria João Lourenço Nº104716

Margarida Pereira Nº105877

Umeima Mahomed Nº 99239

Índice

Introdução

Redes Sociais

As redes são estruturas complexas que podem ser utilizadas para modelar uma vasta gama de sistemas naturais e artificiais. No contexto das ciências sociais, as redes podem ser utilizadas para representar relações entre indivíduos, organizações ou entidades coletivas. [1]

Neste sentido, foi proposto no âmbito da UC de *Análise de Redes* inserida na licenciatura em Ciência de Dados, este trabalho que visa implementar o algoritmo de geração de redes aleatórias Passeio Aleatório (*Random Walk Model*) com diferentes configurações iniciais de modo a comparar os resultados obtidos; e aplicar os quatro métodos de deteção de comunidades à componente gigante da rede estudada no primeiro trabalho.

Questão 1

De entre os métodos de geração de redes aleatórias, iremos considerar neste trabalho o *Modelo Passeio Aleatório*. Proposto em 2003, este modelo introduziu o princípio do *fecho triádico forte* para explicar a formação das ligações nas redes sociais.

Isto é, favorece a introdução de ligações entre nodos adjacentes, útil à formação de triângulos resultante da adição de uma ligação, designada por *fecho triádico*.

A ideia subjacente ao algoritmo consiste em unir um novo nodo não só a um já existente, mas também a um ou mais nodos adjacentes deste último.

Recorrendo ao *package* `igraph` do *R*, implementamos o algoritmo que pode ser descrito, segundo [1], como:

> Considere-se uma rede qualquer de dimensão pequena. Cada iteração consiste nos passos:
>
> **1 |** Um novo nodo i é acrescentado à rede com $m > 1$ ligações;
>
> **2 |** A primeira ligação será (i, j) em que j é um nodo já existente, escolhido aleatoriamente;
>
> **3 |** Cada uma das restantes ligações:
>
> - Une i a um dos adjacentes de j com probabilidade p ou;
> - Une i a um nodo escolhido aleatoriamente com probabilidade $1 - p$.

O parâmetro p é a probabilidade de fecho triádico, porque ao estabelecer uma ligação entre i e um vizinho de j, digamos l, fechamos o triângulo (i, j, l).

- Se $p = 0$, não há fecho triádico e os novos nodos escolhem os seus vizinhos de forma totalmente aleatória.
- Quando $p = 1$, todas as ligações, exceto a primeira, são ligadas aos vizinhos do nodo antigo inicialmente selecionado, fechando assim os triângulos.

Assim, dado que o número de triângulos formados depende da probabilidade p considerada, se esta não for demasiado pequena, este modelo vai gerar *hubs*.

O funcionamento deste algoritmo é ilustrado na Figura 1.

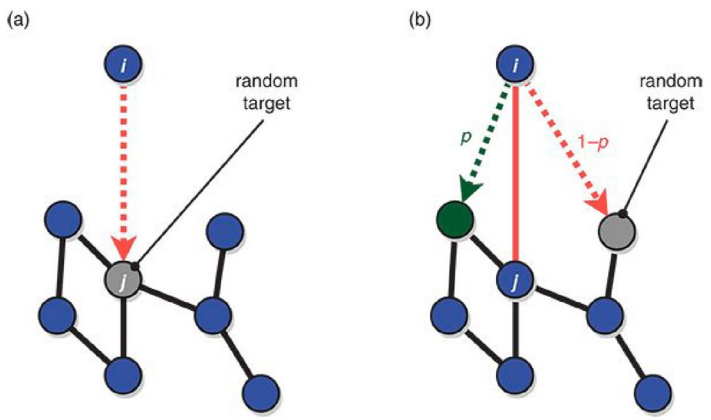


Figura 1 | Ilustração do funcionamento do *Random Walk Model*.
Fonte: [1]

Como configuração inicial do modelo, iremos considerar, primeiramente, uma clique com n nodos (sendo $n = 10$ na alínea **a)** e $n = 20$ na alínea **b)**) e, a partir dela, gere 10 redes aleatórias, cada uma com 200 nodos.

Assim, implementámos o algoritmo presente no apêndice deste relatório que contempla os parâmetros: $p = 0.8$; e o número de ligações introduzidas em cada iteração igual a 3.

Deste modo tivemos em atenção ao implementar o código, que a ligação introduzida entre cada par de nodos fosse apenas 1, ou seja, no *3º Passo*, em que são adicionadas 2 ligações, para perfazer o parâmetro estipulado, os pares de nodos escolhidos não correspondam aos mesmos nas duas iterações deste passo codificadas.

Iremos caracterizar as redes geradas quanto à distância média, ao coeficiente de *clustering* da rede e à existência de *hubs*. Para tal, consideramos as métricas:

- **Distância Média** ($< \boldsymbol{l} >$): É a média dos comprimentos dos caminhos mais curtos (caminho entre um par de nodos com menor comprimento) de uma rede não orientada. Pode ser calculada como:

$$< l > = \frac{2 \sum_{i,j} l_{ij}}{N(N-1)}$$

em que l_{ij} é o **comprimento do caminho mais curto** entre i e j, e N representa o número de nodos da rede.

  - É um valor superior a 0 e se for pequeno, as distâncias são curtas.
  - Para avaliar se é pequeno, consideramos $< l > \sim \log_{10} N$ aceite como significativo de que as distâncias são curtas.

- **Coeficiente de Clustering ($\boldsymbol{C}$):** Mede a quantidade de triângulos da rede.

$$C = \frac{Nº\ de\ ternos\ conexos\ fechados}{Nº\ total\ de\ ternos\ conexos} = \frac{3 \times Nº\ de\ Triângulos}{Nº\ total\ de\ ternos\ conexos} = \frac{traço(A^3)}{\sum_i k_i(k_i - 1)}$$

  - C varia entre $[0,1]$, sendo que quando próximo de 0, a rede tem poucos triângulos, e quando próximo de 1, a rede tem muitos triângulos.
  - Seja $C > 0.5$, significa que existem muitos triângulos, sendo $C > 0.3$ aceite como significativo de que existem muitos triângulos.
  - É de notar que coeficientes de *clustering* significativos ocorrem geralmente em redes sociais, devendo-se de existirem muitos triângulos, sendo este o mecanismo designado por fecho triádico que é característico do modelo de geração de redes aleatórias *Modelo Passeio Aleatório*.

- **Coeficiente de Heterogeneidade (de grau) (K):** Avalia a existência de *hubs*.

$$K = \frac{\langle k^2 \rangle}{\langle k \rangle^2}$$

  - É um valor superior a 1, e quanto maior, mais a rede é heterogénea em grau.
  - Seja $K > 1.5$, é aceite como significativo de que os graus dos nodos são heterogéneos, e por isso, indicia a existência de *hubs* na rede.

Alínea a)

Observa-se de seguida os resultados da aplicação do algoritmo, usando uma clique com **10 nodos** como configuração inicial (**Tabela 1**).

Rede	Distância Média (<l>)	Coeficiente de Clustering (C)	Heterogenidade (K)
1	3.2217	0.2663	1.7410
2	3.2431	0.2669	1.6186
3	3.3264	0.2670	1.6509
4	3.3539	0.2864	1.5972
5	3.3677	0.2933	1.5443
6	3.1559	0.2504	1.8095
7	3.2735	0.2738	1.6199
8	3.1357	0.2526	1.7828
9	3.2399	0.2714	1.6881
10	3.2759	0.2791	1.6633

Tabela 1 | Redes geradas pelo Modelo Passeio Aleatório na *alínea a)*.

Avaliando os resultados das 10 redes geradas na **alínea a)**, podemos concluir que a distância média não é curta (pois $< l >$ não é um valor próximo de $\log_{10} 200 = 2.3$), sendo que no caso apresentado a distância média varia entre aproximadamente 3.14 e 3.37; a rede tem poucos triângulos ($C < 0.3$), dado que os valores variam de cerca de 0.25 a 0.29; e existem *hubs,* cujo valores de K variam entre cerca de 1.54 e 1.81 ($K > 1.5$, e por isso os graus dos nodos da rede são heterogéneos).

Alínea b)

Já quando observamos os resultados da aplicação do algoritmo, usando uma clique com 20 nodos como configuração inicial, os resultados são apresentados na Tabela 2.

Rede	Distância Média (<l>)	Coeficiente de Clustering (C)	Heterogenidade (K)
1	3.0596	0.4273	2.1499
2	3.0143	0.4089	2.2299
3	3.1070	0.4319	2.1105
4	3.0222	0.4230	2.1479
5	3.0139	0.4136	2.2004
6	2.9723	0.4094	2.2492
7	3.0223	0.4197	2.1664
8	3.0242	0.4275	2.1664
9	3.0660	0.4196	2.1790
10	3.0197	0.4208	2.1451

Tabela 2 | Redes geradas pelo Modelo Passeio Aleatório na *alínea b)* .

Avaliando os resultados das 10 redes geradas na alínea b), podemos concluir que a distância média não é curta (pois $< l >$ não é um valor próximo de $\log_{10} 200 = 2.3$), sendo que no caso apresentado a distância média varia entre aproximadamente 2.97 e 3.11; a rede tem muitos triângulos ($C > 0.3$), dado que os valores variam de cerca de 0.41 a 0.43; e existem *hubs,* cujo valores de K variam entre cerca de 2.11 e 2.25 ($K > 1.5$, e por isso os graus dos nodos da rede são heterogéneos).

Alínea c)

Ao comparar os resultados obtidos anteriormente (Tabela 3), estes revelam diferenças a considerar.

	Clique com 10 nodos			Clique com 20 nodos		
Rede	<l>	C	K	<l>	C	K
1	3.2217	0.2663	1.7410	3.0596	0.4273	2.1499
2	3.2431	0.2669	1.6186	3.0143	0.4089	2.2299
3	3.3264	0.2670	1.6509	3.1070	0.4319	2.1105
4	3.3539	0.2864	1.5972	3.0222	0.4230	2.1479
5	3.3677	0.2933	1.5443	3.0139	0.4136	2.2004
6	3.1559	0.2504	1.8095	2.9723	0.4094	2.2492
7	3.2735	0.2738	1.6199	3.0223	0.4197	2.1664
8	3.1357	0.2526	1.7828	3.0242	0.4275	2.1664
9	3.2399	0.2714	1.6881	3.0660	0.4196	2.1790
10	3.2759	0.2791	1.6633	3.0197	0.4208	2.1451

Tabela 3 | Tabela comparativa dos resultados obtidos na *alínea a)* e *b)* .

As redes geradas a partir de cliques iniciais com 20 nodos, em comparação com as redes geradas a partir de cliques iniciais com 10 nodos, apresentam:

- distâncias ligeiramente mais curtas (valores de $< l >$ menores);
- coeficiente de *clustering* (C) significativamente mais alto, ou seja, as redes geradas com a clique inicial com menos nodos têm poucos triângulos, e com mais nodos passam a ter muitos triângulos;
- parâmetro K, que indica a heterogeneidade nos valores dos graus na rede, é mais elevado nas redes geradas a partir da clique inicial do algoritmo com 20 nodos. Isso sugere a presença de *hubs*, ou seja, nodos na rede com graus significativamente superiores aos da maioria dos outros nodos, em comparação com a clique inicial de 10 nodos.

  Estes resultados indicam teoricamente caudas mais longas ou pesadas nas redes, onde muitos nodos têm poucos vizinhos, enquanto poucos nodos têm muitos vizinhos.

Em suma, a escolha do tamanho inicial da clique no algoritmo *Passeio Aleatório* influencia as características das redes geradas.

Questão 2

Considerando a componente gigante da rede estudada no primeiro trabalho de grupo, cujos nodos representam os habitantes da zona residencial e cada ligação indica a existência de contacto social direto entre dois habitantes, iremos aplicar os métodos de deteção de comunidades lecionados.

De forma a avaliar as partições obtidas, iremos utilizar a **modularidade**, que é uma medida baseada na ideia de que avaliar as partições consiste em comparar os resultados com redes em que não existem comunidades. Nas redes aleatórias, redes em que as ligações são escolhidas aleatoriamente, não é expectável encontrar comunidades.

Assim, pode-se comparar o número de ligações internas de cada subrede da partição com o número esperado de ligações existentes numa rede aleatória, sendo assim, esta é utilizada como *baseline*.

A modularidade, Q, de uma partição numa rede não orientada e não ponderada é definida como

$$Q = \frac{1}{L}\sum_{C}\left(L_C - \frac{k_C^2}{4L}\right)$$

em que:

- a soma abrange todos os *clusters* da partição;
- L_c é o número de ligações internas no cluster C;
- k_c é o grau total dos nodos em C;
- L é o número de ligações na rede.

No caso de o número de ligações internas de cada subrede ser bastante superior ao número esperado numa rede aleatória, o valor da modularidade será **mais elevado**.

Caso contrário, o número de ligações internas de cada subrede for aproximado do número esperado numa rede aleatória então a modularidade tomará um valor **mais baixo**.

Apesar de útil na comparação da qualidade das partições de diferentes métodos, a modularidade apresenta algumas limitações, entre as quais: o valor máximo tende a ser mais elevado em redes de maior dimensão, o que não permite a comparação entre redes de dimensões diferentes; e apesar de ser uma medida que procura as diferenças entre a rede em estudo e a estrutura de uma rede aleatória, pode assumir valores elevados para algumas redes aleatórias.

No caso em estudo, a primeira limitação não se aplica na medida em que a rede utilizada em todos os métodos será a mesma e por isso a sua dimensão mantem-se, sendo assim comparável.

Com o intuito de detetar as comunidades presentes na rede em análise, utilizaram-se os métodos: Algoritmo de *Girvan e Newmann/ Edge Betweenness* (Remoção de Pontes), *Fast Greedy* (Otimização de Modularidade), Algoritmo de *Louvain* e *Label Propagation* (Propagação de Etiquetas).

De modo a estudar a aleatoriedade e encontrar soluções alternativas, iremos replicar os métodos 5 *vezes* para que a análise seja mais adequada e completa. Os resultados produzidos encontram-se na Tabela 4.

Método	Número de Comunidades	Dimensão das Comunidade	Modularidade
Remoção de Pontes [1]	27	18, 17, 18, 18, 18, 33, 16, 62, 6, 20, 7, 15, 11, 12, 18, 30, 5, 18, 11, 45, 25, 31, 9, 15, 3, 9, 6	0.840
Remoção de Pontes [2]	27	18, 17, 18, 18, 18, 33, 16, 62, 6, 20, 7, 15, 11, 12, 18, 30, 5, 18, 11, 45, 25, 31, 9, 15, 3, 9, 6	0.840
Remoção de Pontes [3]	27	18, 17, 18, 18, 18, 33, 16, 62, 6, 20, 7, 15, 11, 12, 18, 30, 5, 18, 11, 45, 25, 31, 9, 15, 3, 9, 6	0.840
Remoção de Pontes [4]	27	18, 17, 18, 18, 18, 33, 16, 62, 6, 20, 7, 15, 11, 12, 18, 30, 5, 18, 11, 45, 25, 31, 9, 15, 3, 9, 6	0.840
Remoção de Pontes [5]	27	18, 17, 18, 18, 18, 33, 16, 62, 6, 20, 7, 15, 11, 12, 18, 30, 5, 18, 11, 45, 25, 31, 9, 15, 3, 9, 6	0.840
Fast Greedy [1]	21	60, 35, 30, 63, 37, 19, 20, 26, 18, 57, 16, 18, 15, 11, 11, 16, 14, 6, 6, 13, 5	0.838
Fast Greedy [2]	21	60, 35, 30, 63, 37, 19, 20, 26, 18, 57, 16, 18, 15, 11, 11, 16, 14, 6, 6, 13, 5	0.838
Fast Greedy [3]	21	60, 35, 30, 63, 37, 19, 20, 26, 18, 57, 16, 18, 15, 11, 11, 16, 14, 6, 6, 13, 5	0.838
Fast Greedy [4]	21	60, 35, 30, 63, 37, 19, 20, 26, 18, 57, 16, 18, 15, 11, 11, 16, 14, 6, 6, 13, 5	0.838
Fast Greedy [5]	21	60, 35, 30, 63, 37, 19, 20, 26, 18, 57, 16, 18, 15, 11, 11, 16, 14, 6, 6, 13, 5	0.838
Algoritmo de Louvain [1]	21	27, 12, 31, 18, 20, 48, 16, 18, 46, 6, 46, 20, 18, 11, 16, 24, 25, 26, 40, 14, 14	0.846
Algoritmo de Louvain [2]	20	27, 12, 35, 18, 32, 66, 16, 18, 46, 6, 47, 20, 18, 11, 16, 16, 25, 26, 27, 14	0.845
Algoritmo de Louvain [3]	21	32, 12, 35, 18, 21, 61, 16, 18, 46, 6, 18, 11, 16, 16, 17, 44, 25, 26, 39, 5, 14	0.846
Algoritmo de Louvain [4]	19	34, 12, 35, 18, 23, 55, 16, 18, 46, 6, 47, 27, 18, 11, 16, 20, 25, 40, 29	0.844
Algoritmo de Louvain [5]	21	32, 35, 18, 36, 54, 16, 46, 19, 6, 16, 11, 12, 16, 16, 45, 25, 26, 27, 14, 12, 14	0.843
Propagação de Etiquetas [1]	58	9, 22, 12, 12, 9, 4, 5, 12, 3, 47, 6, 13, 7, 5, 6, 23, 7, 5, 3, 11, 8, 16, 10, 8, 4, 25, 13, 12, 5, 22, 5, 9, 4, 11, 14, 9, 8, 3, 3, 4, 6, 3, 4, 4, 3, 5, 4, 3, 9, 6, 6, 7, 4, 5, 4, 3, 4, 2	0.790
Propagação de Etiquetas [2]	56	13, 9, 8, 5, 8, 5, 3, 7, 3, 47, 6, 12, 7, 8, 6, 21, 9, 11, 3, 11, 11, 12, 5, 18, 11, 3, 6, 21, 10, 12, 9, 19, 10, 25, 5, 15, 12, 4, 4, 4, 3, 4, 10, 4, 3, 11, 3, 9, 6, 4, 5, 4, 2, 5, 3, 2	0.804
Propagação de Etiquetas [3]	53	13, 12, 11, 13, 6, 7, 5, 12, 19, 4, 57, 6, 7, 6, 16, 13, 7, 6, 3, 9, 11, 16, 14, 7, 6, 11, 25, 10, 8, 18, 11, 9, 4, 20, 5, 8, 10, 11, 3, 3, 4, 4, 5, 4, 8, 3, 6, 6, 3, 2, 4, 2, 3	0.794
Propagação de Etiquetas [4]	57	14, 10, 9, 11, 5, 13, 5, 6, 53, 6, 13, 8, 6, 13, 9, 6, 3, 11, 11, 12, 18, 13, 5, 2, 6, 24, 12, 8, 22, 5, 7, 20, 4, 11, 9, 6, 4, 5, 10, 5, 3, 4, 4, 4, 5, 4, 3, 3, 9, 8, 6, 6, 3, 5, 4, 3, 2	0.795
Propagação de Etiquetas [5]	57	27, 10, 7, 9, 5, 7, 5, 16, 46, 6, 13, 6, 8, 6, 7, 3, 8, 11, 12, 2, 16, 14, 5, 8, 4, 24, 13, 14, 5, 18, 10, 21, 5, 4, 13, 5, 7, 4, 4, 4, 3, 9, 9, 4, 6, 4, 5, 4, 3, 9, 6, 4, 4, 3, 4, 3, 4	0.804

Tabela 4 | Resumo dos resultados da aplicação dos métodos de deteção de comunidades.

1 | Algoritmo de *Girvan-Newmann* (Remoção de Pontes / *Edge Betweenness*)

Uma ponte (Fig. 2) é uma ligação cuja remoção divide uma rede conexa em duas subredes. No caso de ser possível identificar todas as pontes de uma rede, a remoção destas pontes iria dividir a rede em sub-redes e a identificação de comunidades seria imediata.

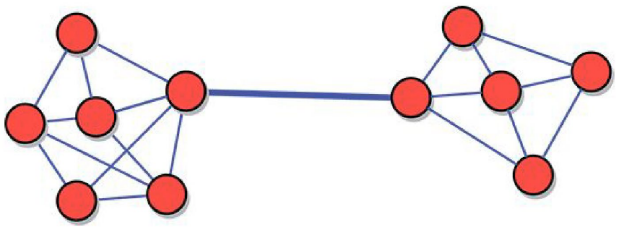


Figura 2 | Ilustração de uma ponte entre 2 redes.
Fonte: [1]

Nos métodos de remoção de pontes é necessária uma medida que permita a identificação de pontes. No algoritmo de *Girvan e Newmann*, esta medida é a intermediação de ligações (número de caminhos mais curtos que passam por uma ligação). Este algoritmo identifica a ligação com maior intermediação e remove-a. Assim, o número de componentes conexas vai aumentando durante a aplicação do algoritmo.

Começa-se por determinar a intermediação de cada ligação.

Cada iteração consiste nos passos:

1 | Remover a ligação com maior intermediação.
No caso de empate, escolher uma das ligações com intermediação mais elevada.

2 | Recalcular a intermediação para cada uma das restantes ligações.
O algoritmo termina quando todas as ligações tiverem removidas.

Em cada iteração é necessário recalcular as medidas de intermediação, o que pode tornar o algoritmo muito lento. Se a rede tiver uma estrutura de comunidades fortes, rapidamente a rede fica dividida em componentes conexas.

Este algoritmo gera N partições. A primeira inclui todos os elementos da rede e em cada iteração cada cluster é dividido em dois clusters. Trata-se, por isso, de um exemplo de um método hierárquico de *clustering* de divisão.

Por se tratar de um método lento, torna-se impraticável a sua aplicação para redes de dimensão elevada. Para tornar o método mais rápido, o cálculo da intermediação pode ser substituído por valores baseados numa amostra de nodos. Outra desvantagem é a obtenção muitas partições sem indicação sobre a sua qualidade.

Na rede em estudo estas limitações não se aplicam, pois a componente gigante da rede tem apenas $N = 496$ nodos e $L = 984$ ligações.

Aplicando o algoritmo através da função `cluster_edge_betweenness()`, obtivemos 27 comunidades (Tabela 4), que variam entre 3 e 62 habitantes, e $Q = 0.84$ nas 5 réplicas realizadas, sendo que as mesmas podem ser representadas como na Figura 3.

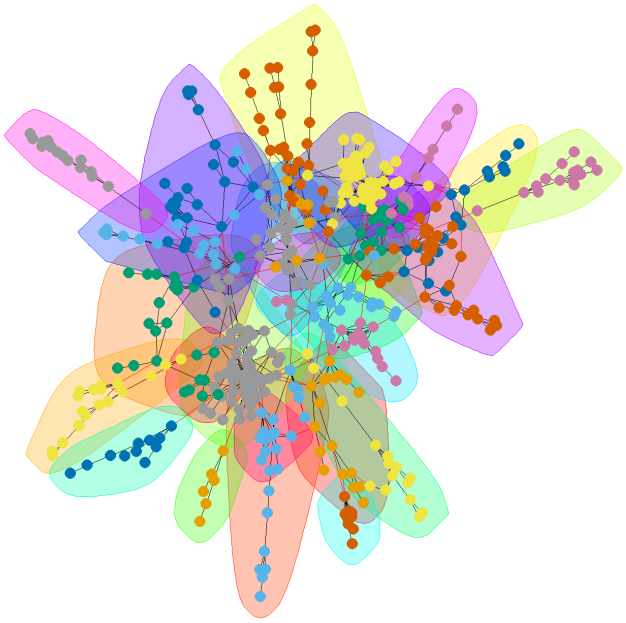


Figura 3 | Representação das comunidades da aplicação do Algoritmo de *Girvan-Newmann*.

Apesar de poder haver aleatoriedade que advém do facto de as ligações serem removidas com base na intermediação de ligação, tal não ocorreu nas réplicas efetuadas. Uma das possíveis razões para este facto pode ser a forma como o algoritmo foi implementado, que pode ter sido definido um critério de desempate, e daí produzir resultados iguais em diferentes iterações.

Isto é, se houver empate nas medidas de intermediação, e o algoritmo estiver configurado para escolher uma das ligações com intermediação mais elevada, pode haver uma consistência nas escolhas, levando a resultados semelhantes em diferentes execuções.

No contexto da rede de habitantes da zona residencial, uma ligação com elevada intermediação pode representar uma ligação social que é crucial para a comunicação entre diferentes grupos de habitantes. Geralmente estas ligações estão associadas a *hubs*. A remoção das mesmas poderá ter implicações na transmissão de informações ou na influência entre diferentes comunidades, identificando na rede subgrupos distintos.

Ainda no contexto da rede, a remoção destas ligações, poderia representar a perda de conexões mais influentes entre habitantes mais sociais.

2 | Algoritmo de Otimização de Modularidade - *Fast Greedy*

O *Fast Greedy* é um método simples, e aglomerativo, de otimização de modularidade consiste em começar com conjuntos singulares (cuja modularidade tem um valor negativo) e em cada iteração fundir 2 grupos. Esta fusão é a que gera um maior aumento de modularidade.

O método termina com apenas um grupo (com todos os nodos da rede). Durante a execução do método, o valor da modularidade começa por crescer até atingir o valor mais elevado. Depois decresce até tomar o valor nulo.

A partição escolhida será a que tem o valor mais elevado de modularidade (Fig.3). Este método é uma heurística ***greedy***.

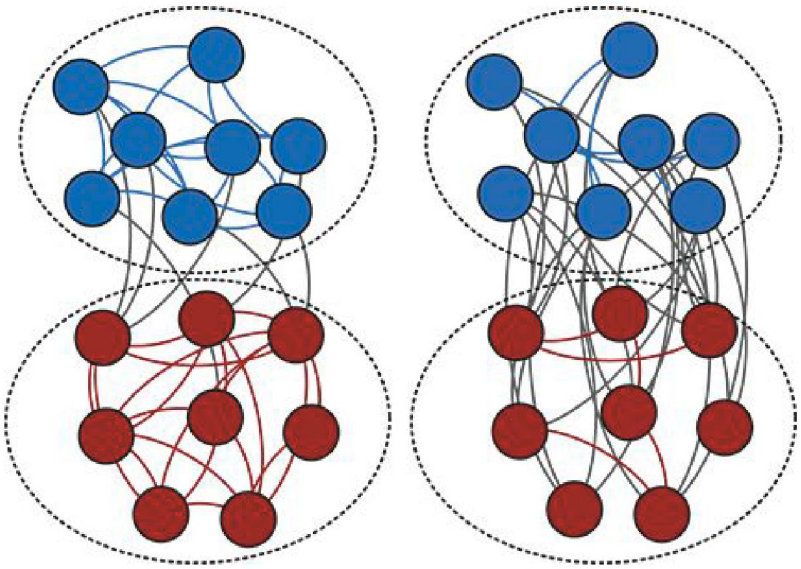


Figura 4 | Ilustração da modularidade da rede.
Fonte: [1]

Aplicando o algoritmo através da função **`cluster_fast_greedy()`**, obtivemos 27 comunidades (Tabela 4), que variam entre 5 e 63 habitantes, e $Q = 0.438$ nas 5 réplicas realizadas, sendo que as mesmas podem ser representadas como na Figura 5.

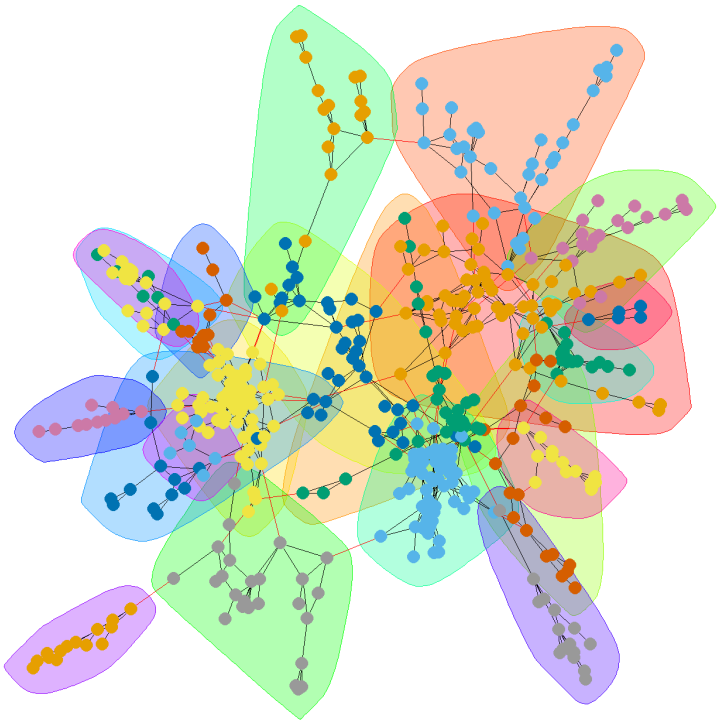


Figura 5 | Representação das comunidades produzidas pelo Algoritmo *Fast Greedy*.

Uma vez mais, apesar de poder haver soluções distintas, dada a natureza *greedy* do algoritmo que não garante que a solução apresentada é a ótima, nas 5 réplicas os resultados foram os mesmos.

Isto pode-se dever ao facto de que o *Fast Greedy* geralmente faz fusões que resultam no maior ganho de modularidade em cada etapa. Se o ganho de modularidade é claro e bem definido em cada iteração, o algoritmo pode convergir para as mesmas fusões em réplicas subsequentes. Se não houver empates significativos nas decisões de fusão, o algoritmo pode ter menos ambiguidades para resolver, levando a resultados consistentes.

Os tamanhos variados das comunidades indicam que as relações sociais na rede não são uniformemente distribuídas. Alguns grupos representam *clusters* de indivíduos com interações sociais frequentes, enquanto outros podem refletir conexões mais esporádicas.

3 | Algoritmo de Otimização de Modularidade - *Louvain*

A solução inicial é composta por conjuntos singulares. Cada iteração do algoritmo consiste em dois passos:

1 | Cada nodo é atribuído à comunidade de um dos nodos adjacentes.
O nodo adjacente é escolhido de forma a obter o maior aumento da modularidade face à solução atual. Os nodos são inspecionados diversas vezes até não ser possível incrementar o valor da modularidade. Este valor é calculado com base na rede original;

2 | Transforma-se a rede numa rede com supernodos e com pesos associados às ligações (Fig. 4).
Cada comunidade determinada no passo anterior é substituída por um supernodo.
As ligações terão peso igual à soma dos pesos das ligações entre as comunidades representadas nos supernodos.
Criam-se lacetes com peso igual à soma dos graus internos.

O algoritmo termina quando não se consegue aumentar a modularidade.

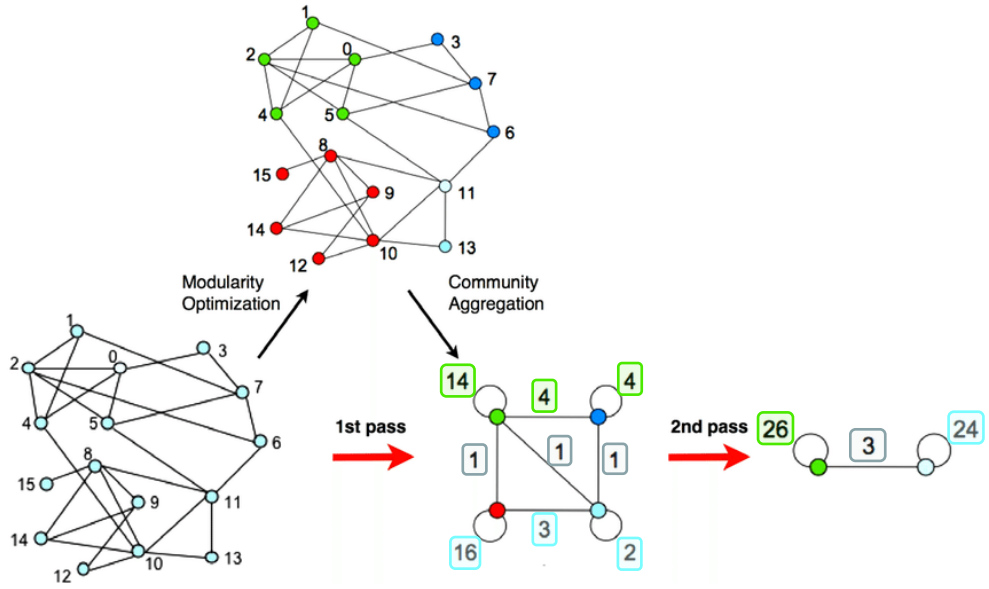


Figura 6 | Ilustração do funcionamento do algoritmo de *Louvain*.
Fonte: Adaptação da Ilustração de [1]

Trata-se de um algoritmo *greedy*, que produz soluções que, usualmente, não estão próximas da ótima. A solução gerada pelo método depende da ordem pela qual os nodos são visitados. Assim, é previsível obterem-se soluções diferentes em diferentes iterações.

Aplicando o algoritmo através da função **`cluster_louvain()`**, obtivemos diferentes soluções em diferentes iterações (Tabela 4), cujo número de comunidades varia entre 19 e 20, sendo que estas variam entre 5 e 66 habitantes, e Q entre 0.843 e 0.846 nas 5 réplicas realizadas, sendo que a última réplica é representada na Figura 7.

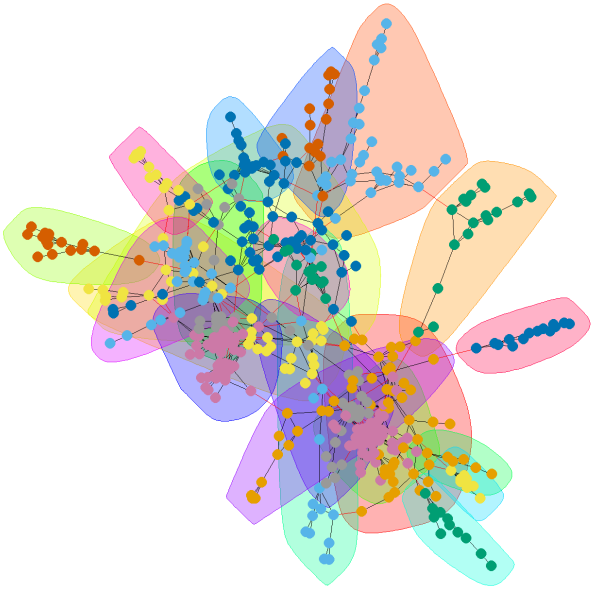


Figura 7 | Representação das comunidades de uma das réplicas da aplicação do Algoritmo de *Louvain*.

Apesar de termos obtido diferentes resultados, as diferenças não são significativas. Estas advêm, principalmente, da aleatoriedade da sensibilidade do algoritmo à escolha dos nodos que começa por agrupar.

No contexto da rede estudada (habitantes da zona residencial com ligações de contactos sociais diretos), os resultados indicam uma organização significativa em grupos sociais distintos, apresentado uma estrutura clara de comunidades na rede, sugerida pelo elevado valor de modularidade.

4 | Algoritmo de Propagação de Etiquetas

O método *Label Propagation* trata-se de um método simples e rápido de deteção de comunidades, baseado na ideia de **que nodos adjacentes pertencem à mesma comunidade**. Desta forma, espera-se que a maioria das ligações seja interna e que as comunidades sejam coesas e bem separadas. Em cada passo, o algoritmo inspeciona cada nodo e afeta-o à comunidade da maioria dos adjacentes.

Começa-se por atribuir uma etiqueta distinta a cada nodo. O método (**Fig. 8**) consiste na repetição dos seguintes passos:

1 | Inspecionam-se todos os nodos, a ordem é **aleatória**. Cada nodo recebe a etiqueta da maioria dos seus adjacentes.

Em caso de empate, escolhe-se aleatoriamente uma das etiquetas mais frequentes.

2 | Se todos os nodos têm a etiqueta da maioria dos seus adjacentes (estado de estacionaridade), o algoritmo termina.

Caso contrário, volta-se ao **Passo 1**.

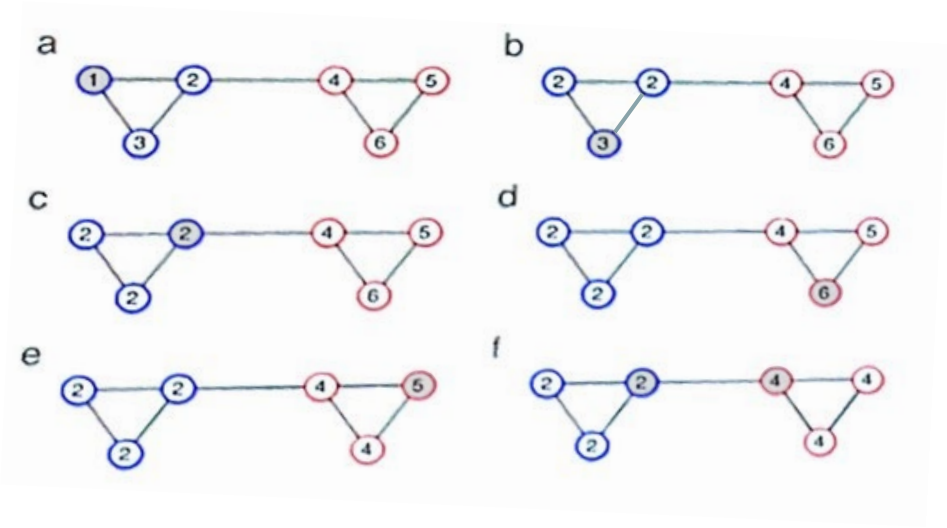


Figura 8 | Ilustração da propagação de etiquetas.
Fonte: [1]

As comunidades são os grupos com **etiquetas iguais**, no **estado estacionário**.

As etiquetas propagam-se durante a execução do algoritmo. Algumas desaparecem, enquanto outras tornam-se mais frequentes.

Trata-se de um método rápido e que pode ser aplicado a rede com dimensão elevada. Contudo, a solução obtida **depende** da ordem de inspeção, daí a relevância de estudar soluções alternativas.

Aplicando o algoritmo através da função **`cluster_label_prop()`**, obtivemos diferentes soluções em diferentes iterações (Tabela 4), cujo número de comunidades varia entre 53 e 58, sendo que estas variam entre 2 e 57 habitantes, e Q entre 0.79 e 0.804 nas 5 réplicas realizadas, sendo que a última réplica é representada na Figura 9.

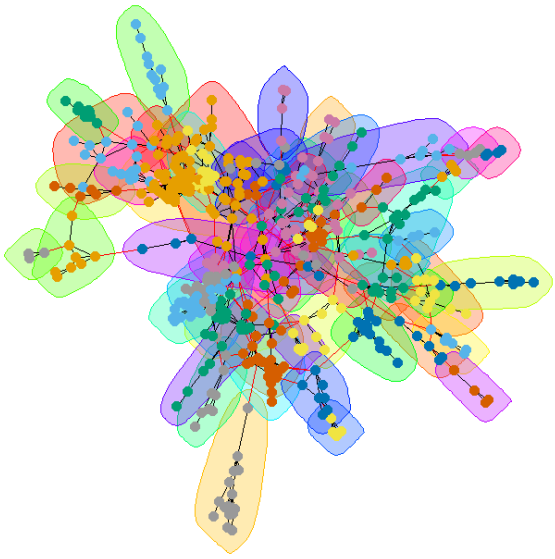


Figura 9 | Representação das comunidades de uma das réplicas da aplicação da Propagação de Etiquetas.

A aleatoriedade deste algoritmo poderá ser resultado de vários fatores, entre eles o ponto de início do algoritmo ser diferente em cada réplica, levando a variações no número de comunidades encontradas, isto porque a dinâmica do algoritmo depende da ordem em que os nodos são visitados e como os rótulos se propagam, sendo este um método local.

Contextualizando para a rede em questão, o número e dimensão das comunidades variável, sugere que diferentes grupos de habitantes estão a ser identificados em cada execução do algoritmo.

Comparação dos Resultados

Os resultados obtidos com os quatro métodos são relativamente semelhantes, com a modularidade a variar entre 0.79 e 0.85. No entanto, existem algumas diferenças importantes entre os resultados.

O número de comunidades obtido com o algoritmo de *Propagação de Etiquetas* é superior ao obtido com os outros métodos. Isso sugere que o algoritmo de *Propagação de Etiquetas* é mais propenso a identificar comunidades pequenas e subdividir comunidades grandes. Esta deteção pode ser pouco valiosa, dependendo do estudo.

Já a dimensão das comunidades obtida com o algoritmo de *Louvain* é mais uniforme do que a obtida com os outros métodos. Isso sugere que o algoritmo de *Louvain* é mais propenso a identificar comunidades com tamanhos semelhantes.

As partições obtidas pelos métodos de *Remoção de Pontes*, *Fast Greedy* e *Louvain* são robustas e consistentes, com valores de modularidade elevados, comparativamente ao método de *Propagação de Etiquetas* que obteve um valor de modularidade mais baixo.

Em geral, os resultados obtidos com os quatro métodos sugerem que a rede estudada é composta por um conjunto de comunidades relativamente pequeno (entre 19 e 27 comunidades), com tamanhos semelhantes e estruturadas de forma semelhante.

Ou seja, no contexto da rede, as comunidades são representações de grupos de pessoas que possam, por exemplo, ter interesses similares ou atividades; fazer parte do mesmo bairro, rua, morar no mesmo prédio (proximidade geográfica); fazer parte da mesma família (laços familiares); ter dinâmicas culturais ou étnicas em comum; partilhar da mesma rede profissional ou educacional; ou frequentar os mesmos espaços públicos. Estas interpretações são expectáveis seguindo as conclusões retiradas do trabalho 1, onde se avaliou a existência de muitos triângulos na componente gigante da rede em estudo.

Por fim, é importante ressaltar que, em teoria, a modularidade não é uma métrica "perfeita" na avaliação das partições obtidas, podendo dever-se ao facto de dois dos algoritmos considerarem a otimização desta métrica, o que poderá enviesar os resultados obtidos. Assim, para uma avaliação mais cuidada das partições teriam de ser analisadas outras métricas.

Em suma, o estudo da deteção de comunidades é intrinsecamente ligado ao contexto e para que a análise atenda ao estudo desejado será necessário compreender o domínio para identificar o número ideal de comunidades.

Bibliografia

[1] Filippo Menczer, Fortunato, S., & Davis, C. A. (2020). *A First Course in Network Science*. Cambridge University Press.

[2] Barabási, A.-L. (2016). Network Science, 1st edition, Cambridge University Press: Cambridge.

[3] Ognyanova, K. (2016). *Introduction to R and network analysis*. https://kateto.net/wp-content/uploads/2018/03/R%20for%20Networks%20Workshop%20-%20Ognyanova%20-%202018.pdf

[4] igraph. (n.d.). *Function reference*. R.igraph.org. https://r.igraph.org/reference/index.html

Apêndice

Trabalho 2 | AR

Rede Aleatórias & Comunidades

André Silvestre Nº 104532 Eliane Gabriel Nº 103303
Maria João Lourenço Nº 104716 Margarida Pereira Nº 105877
Umeima Mahomed Nº 99239

16 de janeiro de 2024

Questão 1

Implementação do Algoritmo de Geração de Redes Aleatórias Passeio Aleatório

```
# Função para gerar redes aleatórias utilizando o Modelo Passeio Aleatório
generate_random_network <- function(initial_size) {

  # Configuração inicial de uma clique completa com n nodos
  rn <- make_full_graph(initial_size, directed = F)

  # Cada uma das redes geradas deve ter 200 nodos
  # Para isso geramos (200-valor inicial de nodos da clique)
  for (i in 1:(200-initial_size)) {

    # Adiciona um novo nodo i (1º Passo)
    rn <- add_vertices(rn, 1)

    # Escolhe um nodo existente aleatoriamente e ligá-lo ao novo nodo (2º Passo)
    selected_node <- sample(1:(vcount(rn)-1), 1)      # Escolher o nodo j
    nn <- neighbors(rn, selected_node)                # Nodos Adjacentes de j
    rn <- add_edges(rn, c(selected_node,vcount(rn)))  # Ligar j a i

    # 3º Passo:
    # Cada uma das restantes ligações une o nodo adicionado a um dos adjacentes
    # do nodo escolhido no Passo 2 com probabilidade 'p'
    # ou une o novo nodo a um nodo escolhido aleatoriamente com probabilidade '1 - p'

    # Adiciona mais 2 ligações (para perfazer as 3 que são impostas no enunciado)
    for (j in 1:2) {
      if (runif(1) < 0.8) {
        selected_node_1 <- sample(setdiff(nn, neighbors(rn, vcount(rn))), 1)
        rn <- add_edges(rn, c(vcount(rn), selected_node_1))
      }
      else {
        # Para garantir que está a escolher um nodo diferente do:
        # Novo nodo e seus vizinhos e do escolhido no Passo 2
```

```
        nn_i <- neighbors(rn, vcount(rn))
        selected_node_2 <- sample(setdiff(1:(vcount(rn)-1), c(selected_node,nn_i)), 1)
        rn <- add_edges(rn, c(selected_node_2,vcount(rn)))
      }
    }
  }

  return(rn)
}
```

```
# Parâmetros das Redes Aleatórias
initial_size_1 <- 10
initial_size_2 <- 20
```

a) Gerar e caracterizar 10 redes aleatórias a partir de clique com 10 nodos

```
# a) Gerar e caracterizar 10 redes aleatórias a partir de clique com 10 nodos
networks_1 <- lapply(1:10,function(i) generate_random_network(initial_size_1))

# Caracterizar as redes geradas quanto à distância média (<l>),
#                                    ao coeficiente de clustering da rede (C)
#                                    e à existência de hubs (K)
distances_1 <- sapply(networks_1, function(net) mean_distance(net))
coef_clustering_1 <- sapply(networks_1, function(net) transitivity(net, type = "global"))
hubs_1 <- sapply(networks_1,
                 function(net){
                   mean(degree(net, mode = "all")*degree(net, mode = "all")/
                          mean(degree(net, mode = "all"))^2)})

result_1 <- data.frame(Rede = 1:10,
                       Distancia_Media = round(distances_1, 4),
                       Coeficiente_Clustering = round(coef_clustering_1, 4),
                       K = round(hubs_1, 4))
```

```r
cat("Resultados para clique com 10 nodos:")
```

```
## Resultados para clique com 10 nodos:
```

```r
ftable <- flextable(result_1)
ftable <- border_remove(x = ftable) %>%
  hline(i= 1, part = "header", border = fp_border(color = "gray", width = 2)) %>%
  hline_bottom(part = "body", border = fp_border(color = "grey", width = 1)) %>%
  vline(j=1:3, border = fp_border(color = "white", width = 5)) %>%
  align(j= 1:4, align = "center", part = "all") %>%
  bg(j = 2:4, bg = "darkgrey", part = "header") %>%
  color(j = 2:4, color = "white", part = "header") %>%
  set_header_labels(Rede = "Rede",
                    Distancia_Media = "Distância Média (<l>)",
                    Coeficiente_Clustering = "Coeficiente de Clustering (C)",
                    K = 'Heterogenidade (K)') %>%
  bold(bold = TRUE, part = "header") %>%
  bold(j = 1, bold = TRUE, part = "body") %>%
  color(j = 1, color = "darkgrey", part = "all") %>%
  autofit()
ftable
```

Rede	Distância Média (<l>)	Coeficiente de Clustering (C)	Heterogenidade (K)
1	3.2217	0.2663	1.7410
2	3.2431	0.2669	1.6186
3	3.3264	0.2670	1.6509
4	3.3539	0.2864	1.5972
5	3.3677	0.2933	1.5443
6	3.1559	0.2504	1.8095
7	3.2735	0.2738	1.6199
8	3.1357	0.2526	1.7828
9	3.2399	0.2714	1.6881
10	3.2759	0.2791	1.6633

b) Gerar e caracterizar 10 redes aleatórias a partir de clique com 20 nodos

```r
# b) Gerar e caracterizar 10 redes aleatórias a partir de clique com 20 nodos
networks_2 <- lapply(1:10, function(i) generate_random_network(initial_size_2))
# Caracterizar as redes geradas quanto à distância média (<l>),
#                                       ao coeficiente de clustering da rede (C)
#                                       e à existência de hubs (K)
distances_2 <- sapply(networks_2, function(net) mean_distance(net))
coef_clustering_2 <- sapply(networks_2, function(net) transitivity(net, type = "global"))
hubs_2 <- sapply(networks_2,
                 function(net){
                   mean(degree(net, mode = "all")*degree(net, mode = "all")/
                          mean(degree(net, mode = "all"))^2)})

result_2 <- data.frame(Rede = 1:10,
                       Distancia_Media = round(distances_2, 4),
                       Coeficiente_Clustering = round(coef_clustering_2, 4),
                       K = round(hubs_2, 4))
```

```r
cat("Resultados para clique com 20 nodos:")
```

```
## Resultados para clique com 20 nodos:
```

```r
ftable <- flextable(result_2)
ftable <- border_remove(x = ftable) %>%
  hline(i= 1, part = "header", border = fp_border(color = "gray", width = 2)) %>%
  hline_bottom(part = "body", border = fp_border(color = "grey", width = 1)) %>%
  vline(j=1:3, border = fp_border(color = "white", width = 5)) %>%
  align(j= 1:4, align = "center", part = "all") %>%
  bg(j = 2:4, bg = "darkgrey", part = "header") %>%
  color(j = 2:4, color = "white", part = "header") %>%
  set_header_labels(Rede = "Rede",
                    Distancia_Media = "Distância Média (<l>)",
                    Coeficiente_Clustering = "Coeficiente de Clustering (C)",
                    K = 'Heterogenidade (K)') %>%
  bold(bold = TRUE, part = "header") %>%
  bold(j = 1, bold = TRUE, part = "body") %>%
  color(j = 1, color = "darkgrey", part = "all") %>%
  autofit()
ftable
```

Rede	Distância Média (<l>)	Coeficiente de Clustering (C)	Heterogenidade (K)
1	3.0596	0.4273	2.1499
2	3.0143	0.4089	2.2299
3	3.1070	0.4319	2.1105
4	3.0222	0.4230	2.1479
5	3.0139	0.4136	2.2004
6	2.9723	0.4094	2.2492
7	3.0223	0.4197	2.1664
8	3.0242	0.4275	2.1664
9	3.0660	0.4196	2.1790
10	3.0197	0.4208	2.1451

c) Comparar os resultados obtidos em a) e b)

```
# c) Comparar os resultados obtidos em a) e b)
result_a_b <- data.frame(result_1, result_2[2:4])

ftable <- flextable(result_a_b)
ftable <- border_remove(x = ftable) %>%
  add_header_row(values = c("",
                            "Clique com 10 nodos",
                            "Clique com 20 nodos"), colwidths = c(1,3,3)) %>%
  hline(i= 2, part = "header", border = fp_border(color = "gray", width = 2)) %>%
  hline_bottom(part = "body", border = fp_border(color = "grey", width = 1)) %>%
  vline(j=1:6, border = fp_border(color = "white", width = 5)) %>%
  align(j= 1:7, align = "center", part = "all") %>%
  bg(i=2, j = 2:4, bg = "#DF6613", part = "header") %>%
  bg(i=2, j = 5:7, bg = "navy", part = "header") %>%
  color(i=1, j = 2:4, color = "#DF6613", part = "header") %>%
  color(i=1, j = 5:7, color = "navy", part = "header") %>%
  color(i=2, j = 2:7, color = "white", part = "header") %>%
  set_header_labels(Rede = "Rede", Distancia_Media = "<l>", Distancia_Media.1 = "<l>",
                    Coeficiente_Clustering = "C", Coeficiente_Clustering.1="C",
                    K = 'K', K.1 = 'K') %>%
  bold(bold = TRUE, part = "header") %>%
  bold(j = 1, bold = TRUE, part = "body") %>%
  color(j = 1, color = "darkgrey", part = "all") %>%
  autofit()
ftable
```

	Clique com 10 nodos			Clique com 20 nodos		
Rede	<l>	C	K	<l>	C	K
1	3.2217	0.2663	1.7410	3.0596	0.4273	2.1499
2	3.2431	0.2669	1.6186	3.0143	0.4089	2.2299
3	3.3264	0.2670	1.6509	3.1070	0.4319	2.1105
4	3.3539	0.2864	1.5972	3.0222	0.4230	2.1479
5	3.3677	0.2933	1.5443	3.0139	0.4136	2.2004
6	3.1559	0.2504	1.8095	2.9723	0.4094	2.2492
7	3.2735	0.2738	1.6199	3.0223	0.4197	2.1664
8	3.1357	0.2526	1.7828	3.0242	0.4275	2.1664
9	3.2399	0.2714	1.6881	3.0660	0.4196	2.1790
10	3.2759	0.2791	1.6633	3.0197	0.4208	2.1451

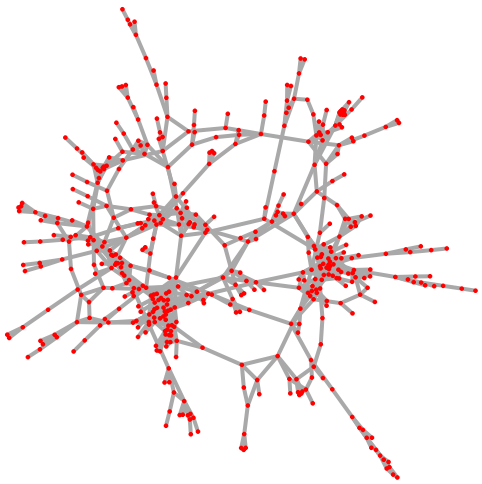
Questão 2

Componente Gigante da Rede de Contactos Sociais Diretos

```
# Importar a rede de um ficheiro .txt
rede <- read_graph("trab_links.txt", format = "edgelist", directed=F)

# Encontrar a componente gigante da rede
components <- clusters(rede)
giant_component <- induced.subgraph(rede,
                                    which(components$membership == which.max(components$csize)))

# Representar graficamente a componente gigante da rede
plot(giant_component,
     vertex.color= "red",vertex.label=NA, vertex.size=2,
     vertex.frame.color=NA, edge.width= 2, )
```



```
cat("Número de nodos na componente gigante:", vcount(giant_component), "\n")
```

```
## Número de nodos na componente gigante: 496
```

```
cat("Número de ligações na componente gigante:", ecount(giant_component), "\n")
```

```
## Número de ligações na componente gigante: 984
```

Métodos de detecção de comunidades

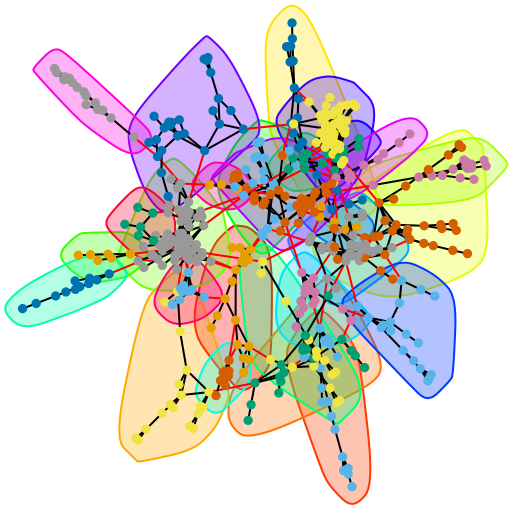
1 | Remoção de Pontes

```
# Iterar 5 vezes o algoritmo para verificar os resultados
results_list_cluster_edge_betweenness <- list()
for (i in 1:5) {
  set.seed(i*123) # Para reprodutibilidade
  cebd <- cluster_edge_betweenness(giant_component)
  results_list_cluster_edge_betweenness[[i]] <- data.frame(
    Método = paste("Remoção de Pontes [", i, "]", sep=""),
    Número_de_Comunidades = length(cebd),
    Dimensão_de_Cada_Comunidade = toString(table(membership(cebd)), collapse = ','),
    Modularidade = round(modularity(cebd), 3)
  )
}

# Combine os resultados em um único dataframe
all_results_cluster_edge_betweenness <- do.call(rbind,
                                                results_list_cluster_edge_betweenness)
t(all_results_cluster_edge_betweenness)
```

```
##                             [,1]
## Método                      "Remoção de Pontes [1]"
## Número_de_Comunidades       "27"
## Dimensão_de_Cada_Comunidade "18, 17, 18, 18, 18, 33, 16, 62, 6, 20, 7, 15, 11, 12, 18, 30, 5, 18, 11
## Modularidade                "0.84"
##                             [,2]
## Método                      "Remoção de Pontes [2]"
## Número_de_Comunidades       "27"
## Dimensão_de_Cada_Comunidade "18, 17, 18, 18, 18, 33, 16, 62, 6, 20, 7, 15, 11, 12, 18, 30, 5, 18, 11
## Modularidade                "0.84"
##                             [,3]
## Método                      "Remoção de Pontes [3]"
## Número_de_Comunidades       "27"
## Dimensão_de_Cada_Comunidade "18, 17, 18, 18, 18, 33, 16, 62, 6, 20, 7, 15, 11, 12, 18, 30, 5, 18, 11
## Modularidade                "0.84"
##                             [,4]
## Método                      "Remoção de Pontes [4]"
## Número_de_Comunidades       "27"
## Dimensão_de_Cada_Comunidade "18, 17, 18, 18, 18, 33, 16, 62, 6, 20, 7, 15, 11, 12, 18, 30, 5, 18, 11
## Modularidade                "0.84"
##                             [,5]
## Método                      "Remoção de Pontes [5]"
## Número_de_Comunidades       "27"
## Dimensão_de_Cada_Comunidade "18, 17, 18, 18, 18, 33, 16, 62, 6, 20, 7, 15, 11, 12, 18, 30, 5, 18, 11
## Modularidade                "0.84"
```

```
plot(cebd, giant_component, vertex.label=NA, vertex.size=4,
     vertex.frame.color=NA, edge.width= 1)
```



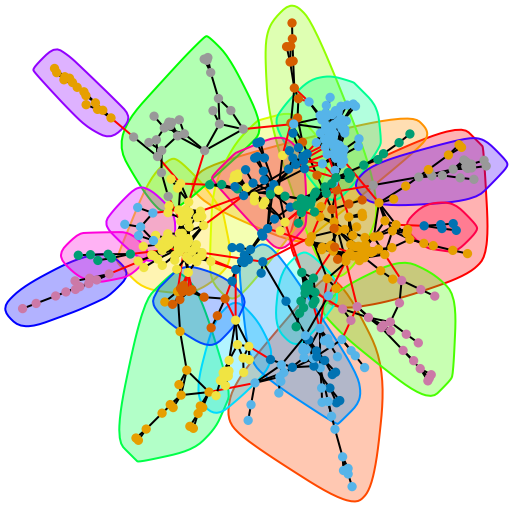
2 | Otimização de Modularidade (*Fast Greedy*)

```
# Iterar 5 vezes o algoritmo para verificar os resultados
results_list_cluster_fast_greedy <- list()
for (i in 1:5) {
  set.seed(i*123) # Para reprodutibilidade
  cfgr <- cluster_fast_greedy(giant_component)
  results_list_cluster_fast_greedy[[i]] <- data.frame(
    Método = paste("Fast Greedy [", i, "]", sep=""),
    Número_de_Comunidades = length(cfgr),
    Dimensão_de_Cada_Comunidade = toString(table(membership(cfgr)), collapse = ','),
    Modularidade = round(modularity(cfgr), 3)
  )
}

# Combine os resultados em um único dataframe
all_results_cluster_fast_greedy <- do.call(rbind,
                                           results_list_cluster_fast_greedy)
t(all_results_cluster_fast_greedy)
```

```
##                             [,1]
## Método                      "Fast Greedy [1]"
## Número_de_Comunidades       "21"
## Dimensão_de_Cada_Comunidade "60, 35, 30, 63, 37, 19, 20, 26, 18, 57, 16, 18, 15, 11, 11, 16, 14, 6, 
## Modularidade                "0.838"
##                             [,2]
## Método                      "Fast Greedy [2]"
## Número_de_Comunidades       "21"
## Dimensão_de_Cada_Comunidade "60, 35, 30, 63, 37, 19, 20, 26, 18, 57, 16, 18, 15, 11, 11, 16, 14, 6, 
## Modularidade                "0.838"
##                             [,3]
## Método                      "Fast Greedy [3]"
## Número_de_Comunidades       "21"
## Dimensão_de_Cada_Comunidade "60, 35, 30, 63, 37, 19, 20, 26, 18, 57, 16, 18, 15, 11, 11, 16, 14, 6, 
## Modularidade                "0.838"
##                             [,4]
## Método                      "Fast Greedy [4]"
## Número_de_Comunidades       "21"
## Dimensão_de_Cada_Comunidade "60, 35, 30, 63, 37, 19, 20, 26, 18, 57, 16, 18, 15, 11, 11, 16, 14, 6, 
## Modularidade                "0.838"
##                             [,5]
## Método                      "Fast Greedy [5]"
## Número_de_Comunidades       "21"
## Dimensão_de_Cada_Comunidade "60, 35, 30, 63, 37, 19, 20, 26, 18, 57, 16, 18, 15, 11, 11, 16, 14, 6, 
## Modularidade                "0.838"
```

```
plot(cfgr, giant_component, vertex.label=NA, vertex.size=4,
     vertex.frame.color=NA, edge.width= 1)
```



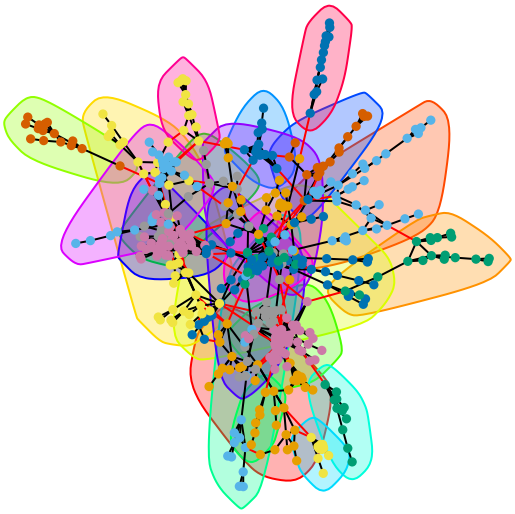
3 | Algoritmo de Louvain

```
# Iterar 5 vezes o algoritmo para verificar os resultados
results_list_cluster_louvain <- list()
for (i in 1:5) {
  set.seed(i*123) # Para reprodutibilidade
  clor <- cluster_louvain(giant_component)
  results_list_cluster_louvain[[i]] <- data.frame(
    Método = paste("Algoritmo de Louvain [", i, "]", sep=""),
    Número_de_Comunidades = length(clor),
    Dimensão_de_Cada_Comunidade = toString(table(membership(clor)), collapse = ','),
    Modularidade = round(modularity(clor), 3)
  )
}

# Combine os resultados em um único dataframe
all_results_cluster_louvain <- do.call(rbind,
                                       results_list_cluster_louvain)
t(all_results_cluster_louvain)
```

```
##                             [,1]
## Método                      "Algoritmo de Louvain [1]"
## Número_de_Comunidades       "21"
## Dimensão_de_Cada_Comunidade "27, 12, 31, 18, 20, 48, 16, 18, 46, 6, 46, 20, 18, 11, 16, 24, 25, 26, 4
## Modularidade                "0.846"
##                             [,2]
## Método                      "Algoritmo de Louvain [2]"
## Número_de_Comunidades       "20"
## Dimensão_de_Cada_Comunidade "27, 12, 35, 18, 32, 66, 16, 18, 46, 6, 47, 20, 18, 11, 16, 16, 25, 26, 1
## Modularidade                "0.845"
##                             [,3]
## Método                      "Algoritmo de Louvain [3]"
## Número_de_Comunidades       "21"
## Dimensão_de_Cada_Comunidade "32, 12, 35, 18, 21, 61, 16, 18, 46, 6, 18, 11, 16, 16, 17, 44, 25, 26, 3
## Modularidade                "0.846"
##                             [,4]
## Método                      "Algoritmo de Louvain [4]"
## Número_de_Comunidades       "19"
## Dimensão_de_Cada_Comunidade "34, 12, 35, 18, 23, 55, 16, 18, 46, 6, 47, 27, 18, 11, 16, 20, 25, 40, 1
## Modularidade                "0.844"
##                             [,5]
## Método                      "Algoritmo de Louvain [5]"
## Número_de_Comunidades       "21"
## Dimensão_de_Cada_Comunidade "32, 35, 18, 36, 54, 16, 46, 19, 6, 16, 11, 12, 16, 16, 45, 25, 26, 27, 1
## Modularidade                "0.843"
```

```
plot(clor, giant_component, vertex.label=NA, vertex.size=4,
     vertex.frame.color=NA, edge.width= 1)
```



13

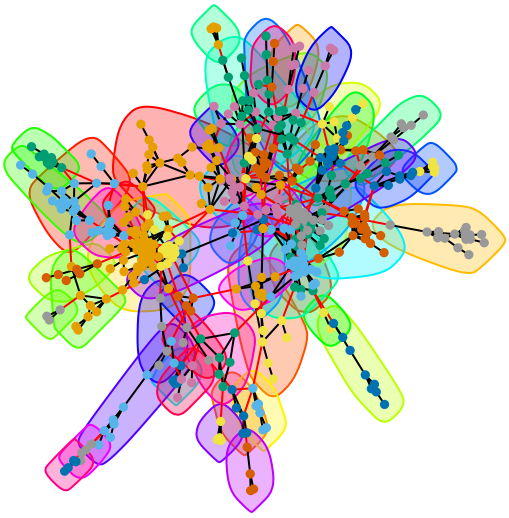
4 | Propagação de Etiquetas

```r
# Iterar 5 vezes o algoritmo para verificar os resultados
results_list_cluster_label_prop <- list()
for (i in 1:5) {
  set.seed(i*123)
  clpr <- cluster_label_prop(giant_component)
  results_list_cluster_label_prop[[i]] <- data.frame(
    Método = paste("Propagação de Etiquetas [", i, "]", sep=""),
    Número_de_Comunidades = length(clpr),
    Dimensão_de_Cada_Comunidade = toString(table(membership(clpr)), collapse = ','),
    Modularidade = round(modularity(clpr), 3)
  )
}
all_results_cluster_label_prop <- do.call(rbind, results_list_cluster_label_prop)

t(all_results_cluster_label_prop)
```

```
##                              [,1]
## Método                       "Propagação de Etiquetas [1]"
## Número_de_Comunidades        "58"
## Dimensão_de_Cada_Comunidade "9, 22, 12, 12, 9, 4, 5, 12, 3, 47, 6, 13, 7, 5, 6, 23, 7, 5, 3, 11, 8, 
## Modularidade                 "0.790"
##                              [,2]
## Método                       "Propagação de Etiquetas [2]"
## Número_de_Comunidades        "56"
## Dimensão_de_Cada_Comunidade "13, 9, 8, 5, 8, 5, 3, 7, 3, 47, 6, 12, 7, 8, 6, 21, 9, 11, 3, 11, 11, 1
## Modularidade                 "0.804"
##                              [,3]
## Método                       "Propagação de Etiquetas [3]"
## Número_de_Comunidades        "53"
## Dimensão_de_Cada_Comunidade "13, 12, 11, 13, 6, 7, 5, 12, 19, 4, 57, 6, 7, 6, 16, 13, 7, 6, 3, 9, 11
## Modularidade                 "0.794"
##                              [,4]
## Método                       "Propagação de Etiquetas [4]"
## Número_de_Comunidades        "57"
## Dimensão_de_Cada_Comunidade "14, 10, 9, 11, 5, 13, 5, 6, 53, 6, 13, 8, 6, 13, 9, 6, 3, 11, 11, 12, 1
## Modularidade                 "0.795"
##                              [,5]
## Método                       "Propagação de Etiquetas [5]"
## Número_de_Comunidades        "57"
## Dimensão_de_Cada_Comunidade "27, 10, 7, 9, 5, 7, 5, 16, 46, 6, 13, 6, 8, 6, 7, 3, 8, 11, 12, 2, 16, 
## Modularidade                 "0.804"
```

```r
plot(clpr, giant_component, vertex.label=NA, vertex.size=4,
     vertex.frame.color=NA, edge.width= 1)
```



```
# Combine todos os resultados em um único dataframe
resultados_completos <- bind_rows(all_results_cluster_edge_betweenness,
                                  all_results_cluster_fast_greedy,
                                  all_results_cluster_louvain,
                                  all_results_cluster_label_prop)
ftable <- flextable(resultados_completos)
ftable <- border_remove(x = ftable) %>%
  hline(i= 1, part = "header", border = fp_border(color = "navy", width = 2)) %>%
  hline_bottom(part = "body", border = fp_border(color = "navy", width = 1)) %>%
  vline(j=1:3, border = fp_border(color = "white", width = 5)) %>%
  align(j= 1:4, align = "center", part = "all") %>%
  bg(i=1, j = 2:4, bg = "navy", part = "header") %>%
  color(i=1, j = 2:4, color = "white", part = "header") %>%
  set_header_labels(Número_de_Comunidades = "Número de Comunidades",
                    Dimensão_de_Cada_Comunidade = "Dimensão das Comunidade",
                    Modularidade = 'Modularidade') %>%
  bold(bold = TRUE, part = "header") %>%  bold(j = 1, bold = TRUE, part = "body") %>%
  color(i=1, j = 1, color = "navy", part = "header") %>%
  autofit() %>% hrule(rule = "exact", part = "all")
ftable
```

Método	Número de Comunidades	
Remoção de Pontes [1]	27	18
Remoção de Pontes [2]	27	18
Remoção de Pontes [3]	27	18
Remoção de Pontes [4]	27	18
Remoção de Pontes [5]	27	18
Fast Greedy [1]	21	
Fast Greedy [2]	21	
Fast Greedy [3]	21	
Fast Greedy [4]	21	
Fast Greedy [5]	21	
Algoritmo de Louvain [1]	21	
Algoritmo de Louvain [2]	20	
Algoritmo de Louvain [3]	21	
Algoritmo de Louvain [4]	19	
Algoritmo de Louvain [5]	21	
Propagação de Etiquetas [1]	58	9, 22, 12, 12, 9, 4, 5, 12, 3, 47, 6, 13, 7,
Propagação de Etiquetas [2]	56	13, 9, 8, 5, 8, 5, 3, 7, 3, 47, 6, 12, 7, 8,
Propagação de Etiquetas [3]	53	13, 12, 11, 13, 6, 7, 5, 12, 19, 4, 57
Propagação de Etiquetas [4]	57	14, 10, 9, 11, 5, 13, 5, 6, 53, 6, 13, 8, 6
Propagação de Etiquetas [5]	57	27, 10, 7, 9, 5, 7, 5, 16, 46, 6, 13, 6, 8