ISCTE-IUL
Licenciatura em Ciência de Dados

Redes não orientadas - Geração de redes aleatórias e deteção de comunidades

Trabalho de Grupo II realizado no âmbito da Unidade Curricular de Análise de Redes do 3º ano da Licenciatura em Ciência de Dados

Allan Kardec Rodrigues, 103380, CDC1
aksrs@iscte-iul.pt

André Plancha, 105289, CDC2
Andre_Plancha@iscte-iul.pt

Diogo Freitas, 104841, CDC1
daafs@iscte-iul.pt

João Francisco Botas, 104782, CDC1
Joao_Botas@iscte-iul.pt

Marco Esperança, 110451, CDC1
mdeao@iscte-iul.pt

16 de janeiro 2024
Versão 1.0.1

Índice

Introdução

A análise de redes tem desempenhado um papel fundamental no estudo de sistemas complexos e interativos, oferecendo um conjunto diversificado de ferramentas para compreender a estrutura e dinâmica das interações entre entidades. Este trabalho aborda diversas facetas da teoria das redes; Na primeira parte (**Q1**) exploramos um modelo de geração de redes aleatórias e na segunda parte (**Q2**) aplicamos, comparamos e comentamos técnicas de deteção de comunidades perante uma componente gigante.

Na **Q1**, investigamos o modelo de geração de redes aleatórias conhecido como Passeio Aleatório (*Random Walk Model*). Este modelo é semelhante ao modelo Barabási-albert (BA), mas este inclui um parâmetro de probabilidade p de fecho triádico. Nós explorámos e comparámos o modelo, gerando 10 redes aleatórias por configuração explorada. Explicitamente, foram comparados os resultados de 2 configurações de geração diferentes, variando na rede inicial.

Na **Q2**, na aplicação de métodos de deteção de comunidades na componente gigante de uma zona residencial. Utilizámos quatro abordagens distintas para identificar comunidades dentro dessa rede, avaliando o número de comunidades, as suas dimensões e a qualidade das partições obtidas por cada método.

Q1 : Geração de redes aleatórias com o Passeio Aleatório

O modelo BA é um modelo de ligação preferencial que parte de uma rede completa inicial, acrescentando-se nodos um a um, priviligiando-se as ligações a nodos com maior grau com o propósito de aumentar a heterogeneidade e de criar *hubs*. É um modelo de crescimento preferencial usado na análise de redes complexas, como as redes sociais, redes de computadores e outras áreas, pois os nós ligam-se a outros nós populares, simulando redes reais (pelo menos em termos de heterogeneidade). Contudo, as redes geradas pelo modelo BA apresentam coeficientes de *clustering* reduzidos porque a probabilidade de um nodo receber uma ligação é proporcional ao seu grau, não tendo em conta a existência de nodos adjacentes. Desta forma, não é propício à formação de triângulos, caracteristica muito importante para a coesão da rede, estando esta presente em redes reais nas mesmas áreas; a existência de triângulos são indícios de existência de comunidades. Para gerar as redes, foi utilizado o Passeio Aleatório (*Random Walk Model*), que tende a aumentar o número de triângulos através de um mecanismo que favorece a introdução de ligação entre nós adjacentes, desta forma, tornando-se o modelo mais relevante para esta questão.

Nosso objetivo foi gerar 10 conjuntos de redes aleatórias, 2 vezes cada, utilizando o modelo PA, e em seguida, comentar, comparar e interpretar os resultados. Nessas 2 vezes, iremos usar os mesmos parâmetros, exceto na rede completa inicial (configuração inicial), onde no conjunto de **a)** iremos usar uma clique com 10 nodos e no conjunto **b)** iremos usar uma clique com 20 nodos. Para os outros parâmetros, usámos a probabilidade $p = 0.8$, o número de ligações introduzidas por iteração $m = 3$ ligações e um número de nodos desejado total de $n = 200$ nodos[1].

[1]De forma a reproduzir os resultados, usámos uma $seed = 1$

Passeio Aleatório

O Algoritmo 1 apresenta uma representação do modelo de PA que implementamos, precendendo à sua descrição.

PasseioAleatório(g, p, m, n)
Altera g
Argumentos:
- g: rede completa inicial
- $0 < p < 1$
- $m > 1$ int: número de ligações
- n int: número de nodos desejado total

```
1  repetir até graph.node_count ≥ n
2      rand_n ← random(g.nodes)
3      novo_n ← g.new_node()
4      vizs ← rand_n.vizinhos
5      g.link(novo_v, rand_n)
6      repetir m − 1 vezes
7          rand_viz ← random(vizs \ (novo_n ∪ novo_n.vizinhos))
8          if u() < p then
9                  g.link(rand_viz, novo_n)
10             else
11                 g.link(novo_n, random(g.nodes \ (novo_n ∪ rand_viz ∪ novo_n.vizinhos))
```

Algoritmo 1: Passeio Aleatório

1 - Inicialização:

Começa-se com uma clique de x nodos como a configuração inicial da rede.

2 - Geração das Redes:

a) Escolhe-se aleatoriamente um nodo existente na rede (`rand_n`).

b) Adiciona-se um novo nodo (`novo_n`) à rede.

c) Obtêm-se os vizinhos do nodo selecionado aleatoriamente (`vizs`).

d) Cria-se uma conexão entre o novo nodo e o nodo selecionado aleatoriamente (`rand_n`).

e) Repete-se $m - 1$ vezes ($m = 3$ neste caso) de f) a h)

Como m indica o número de ligações desejadas, mas já se criou uma ligação em d), apenas se vai adicionar mais $m - 1$ ligações.

f) Escolhe-se aleatoriamente um vizinho do nodo selecionado (`rand_viz`) que não seja o novo nodo ou um vizinho do novo nodo.

g) Com uma probabilidade p ($p = 0.8$ neste caso), conecta-se o novo nodo ao vizinho selecionado (`rand_viz`), formando um triângulo.

h) Com uma probabilidade de $1 - p$, conecta-se o novo nodo a um nodo aleatório na rede, exceto o novo nodo, o vizinho selecionado e os vizinhos do novo nodo.

i) Voltar a b) até que o número de nodos na rede seja igual a n ($n = 200$ neste caso).

Dada a probabilidade ser elevada ($p = 0.8$), espera-se que se formem muitos triângulos e, consequentemente, que o coeficiente de *clustering* da rede seja significativo. Isto deve-se ao facto de que em 80% das vezes o nodo será ligado a um adjacente do nodo escolhido. A probabilidade elevada vai proporcionar uma maior heterogenidade e, consequentemente, à criação de hubs, isto é, o algoritmo terá uma maior tendência a conectar nodos a vizinhos, e não a nodos aleatórios.

Demonstração do algoritmo

De forma a demonstrar o algoritmo, apresentamos a configuração inicial de **Q1** após uma primeira iteração na Figura 2. A configuração inicial de **Q1** é uma clique inicial de 10 nodos e encontra-se na Figura 1. Uma clique é uma (sub)rede completa, ou seja, todos os nodos estão ligados a todos os outros nodos da subrede. Por isso, o grau de todos os nodos de uma clique será $\forall_i \in V : k_i = N - 1$, sendo V o conjunto de todos os nodos, k_i o grau (número de ligações) do nodo i, e N no número de nodos ($\#V = N$). Neste caso, uma clique de 10 nodos significa que $\forall_i \in V : k_i = 10 - 1 = 9$.

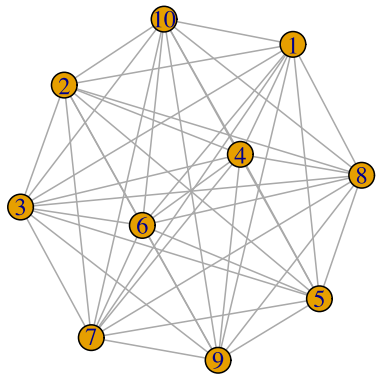


Figura 1: Clique inicial com 10 nodos

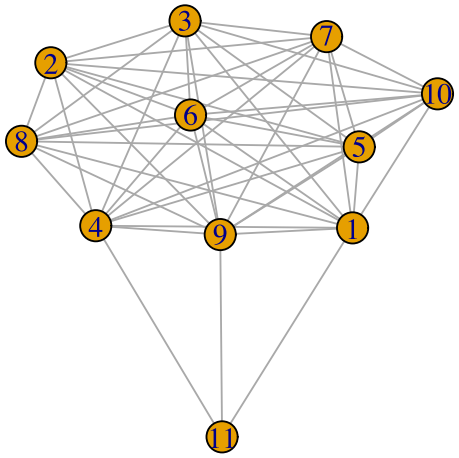


Figura 2: Rede após uma iteração

Nesta primeira iteração, `novo_n` foi o nodo 11, `rand_n` foi o nodo 9, e `rand_vizs` foram os nodos 4 e 1, ambos ligados a 9 porque ambos os números gerados entre 0 e 1 (`u()`) foram $< p$. A primeira iteração vai sempre ligar a vizinhos do nodo escolhido independente do valor de p, porque todos os nodos de V são vizinhos do nodo escolhido. O algoritmo passará para a próxima iteração, escolhendo um nodo `rand_n` e criando um novo `novo_n` (nodo 12).

Comparação de configuração inicial de a) e b) *à priori*

A configuração inicial de **b)** foi uma clique de 20 nodos, representada na Figura 3. A configuração inicial de **a)** encontra-se na Figura 1.

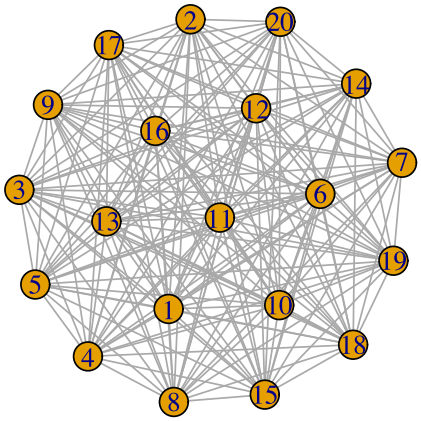


Figura 3: Clique inicial com 20 nodos

Se visualizarmos a clique inicial da Figura 3, percebemos que esta rede terá, inicialmente, um número de ligações mais elevado face à clique com 10 nodos, mostrada anteriormente. Por ser uma rede completa, o número de ligações corresponde ao valor máximo da mesma ($L = L_{\max} = \frac{N(N-1)}{2} = \frac{20\times19}{2} = 190$ ligações). Isto indica, desde já, que teremos mais triângulos iniciais na situação de **b)** (embora o coeficiente de *clustering* será 1 em ambas inicialmente), o que irá fazer com que o numerador do coeficiente de *clustering* comecerá maior na **b)** do que na **a)**. No entanto, a inclusão de novos nodos e ligações a cada iteração irá também criar novos potenciais triângulos, o que vai diminuir o coeficiente de *clustering* também. Ainda assim, como a **b)** começa com mais nodos, terá menos iterações, o que afeta também o coeficiente de *clustering* final, embora não seja tão óbvio se é positivamente ou negativamente. Com isto tudo, parece difícil prever o que vamos observar nesta métrica.

Sobre a distância mínima média $\left(\text{distância média}, <l> = \frac{\sum(l_{i,j})}{N(N-1)}\right)$[2], não pensamos que a configuração inicial fosse afetar muito, porque começamos com cliques e $\forall i, j \in V : l_{i,j} = 1$; portanto as distâncias média devem acabar igual.

Sobre a heterogeneidade $\left(\kappa = \frac{<k^2>}{<k>^2}\right)$, logo após a primeira iteração, a heterogeneidade começa maior na situação **b)**, pois o novo nodo introduzido vai ter grau $k = 3$, enquanto que o resto dos nodos vão ter grau $k = 20$ ($k = 10$ na situção **a)**). No entanto, suspeitamos que este vão se aproximar ao longo das iterações, e $\kappa_{\text{a)}} \approx \kappa_{\text{b)}}$ após ambas chegarem a 200 nodos.

[2] $i \neq j$, equivalente a $2\frac{\sum(l_{i,j})}{N(N-1)}$ se consinsiderarmos que i nunca toma o valor que j tomou.

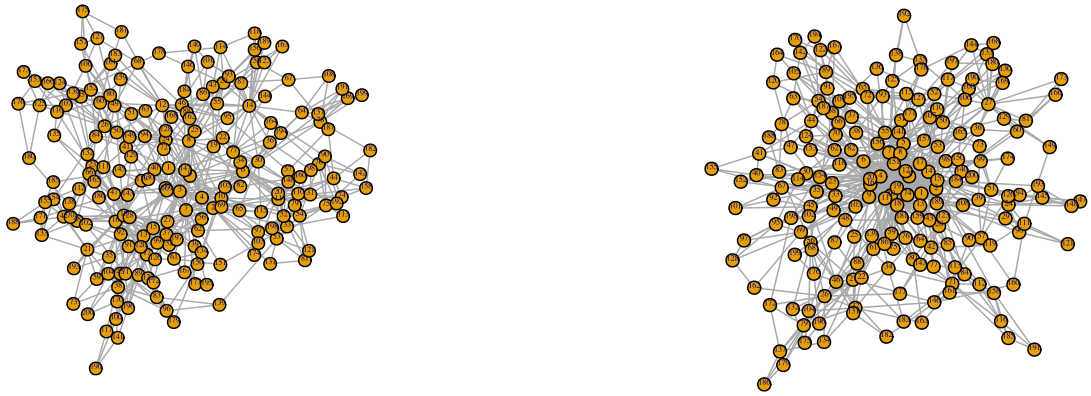


Figura 4: Redes nº 7 finais para as 2 situações cliques, na esquerda **a)**, na direita **b)**

a) Resultados das redes geradas a partir de cliques com 10 nodos

Após a execução do algoritmo para 10 cliques, calculamos os valores da distância média, coeficiente de *clustering*, coeficiente de *clustering* médio (C.C. Médio), heterogenidade, e se há presença de *hubs* na rede gerada (*Hubs?*), arredondamos os valores e apresentámo-los na Tabela 1. A coluna de presença de hubs indica se o valor é significativamente maior que 1 (consideramos significativo quando $\kappa > 1.5$).

Tabela 1: Resulados obtidos a partir dos cliques com 10 nodos

Nº da rede gerada	Distância média	Coeficiente de *Clustering*	C.C. Médio	Heterogeneidade	*Hubs*?
1	3.221	0.264	0.510	1.723	**Sim**
2	3.319	0.275	0.465	1.568	**Sim**
3	3.324	0.277	0.449	1.549	**Sim**
4	3.258	0.259	0.434	1.612	**Sim**
5	3.204	0.261	0.485	1.672	**Sim**
6	3.163	0.252	0.479	1.725	**Sim**
7	3.442	0.302	0.482	1.556	**Sim**
8	3.293	0.268	0.448	1.586	**Sim**
9	3.234	0.276	0.491	1.653	**Sim**
10	3.178	0.245	0.458	1.759	**Sim**

Da Tabela 1 podem retirar-se as seguintes observações:

- **Distância média**: Os valores da distância média variam entre, aproximadamente, 3.16 e 3.44, com uma média de distâncias médias de ≈ 3.2637 e um desvio padrão (sd) de ≈ 0.0838. Ou seja, em média, a distância mais pequena entre quaisquer dois nodos na rede é em torno de 3 unidades de comprimento de caminho. Esses valores são relativamente próximos entre si, indicando consistência na distância média entre as redes geradas. Além disso, as distâncias média são argumentavelmente pequenas, ainda que $3.2637 > \log_{10}(200) \approx 2.3$, sendo que os valores estão próximos ($|\log_{10}(200) - 3.2637| \approx 0.76 < 1$);
- **Coeficiente de *clustering***: Os valores do coeficiente de *clustering* variam entre, aproximadamente, de 0.245 e 0.302, com ume média de ≈ 0.2679 e um sd de ≈ 0.0158, indicando que os vizinhos de um nodo tendem a estar interconectados em proporções semelhantes entre as redes. A consistência desses valores sugere uma tendência

similar de agrupamento local nos diferentes contextos de rede. O valor desta medida está no limiar de se designar elevado e caracteriza-se pela existência de alguns triângulos;

- **Heterogeneidade**: A medida de heterogeneidade varia aproximadamente entre 1.548 e 1.759, sendo que valores mais altos sugerem uma distribuição mais desigual ou heterogénea dos graus dos nodos na rede. No entanto, esses valores são relativamente próximos, indicando que a diferença na distribuição de graus entre as redes geradas não é muito acentuada. O valor desta medida é um tanto quanto elevado e, por isso, indica a presença de *hubs*;
- **Identificação de *Hubs***: Todos os parâmetros de heteregeneidade são significativamente maior do que 1, pelo que há nodos que se destacam com uma conectividade significativamente maior em relação aos demais, sendo suscetível a existência de *hubs* para as 10 redes geradas.

Os resultados indicam uma consistência notável entre as redes geradas, com distâncias médias e coeficientes de *clustering* bastante próximos entre si. A presença de hubs em todas as redes sugere a existência de nodos de alta conectividade, que podem ser de grande importância na estrutura e na dinâmica destas redes.

b) Resultados das redes geradas a partir de cliques com 20 nodos

Seguindo o mesmo procedimento de **a)** (apenas mudando a configuração inicial), na Tabela 2 encontram-se os resultados com as mesmas métricas, seguido da análise. A comparação de resultados foi feita em **c)**.

Tabela 2: Resulados obtidos a partir dos cliques com 20 nodos

Nº da rede gerada	Distância média	Coeficiente de *Clustering*	C.C. Médio	Heterogeneidade	*Hubs*?
1	3.079	0.425	0.520	2.148	**Sim**
2	3.019	0.412	0.540	2.228	**Sim**
3	3.031	0.424	0.508	2.158	**Sim**
4	2.993	0.412	0.542	2.232	**Sim**
5	3.087	0.435	0.544	2.12	**Sim**
6	2.973	0.401	0.513	2.265	**Sim**
7	3.048	0.427	0.523	2.14	**Sim**
8	3.02	0.417	0.530	2.204	**Sim**
9	3.092	0.431	0.547	2.126	**Sim**
10	3.084	0.425	0.509	2.144	**Sim**

- **Distância Média**: Os valores da distância média variam entre 2.973 e 3.092, aproximadamente, com média ≈ 3.0426 e sd ≈ 0.04204. Isso indica que, em média, a distância entre quaisquer dois nodos na rede é em torno de 3 unidades de comprimento de caminho. Assim como na análise anterior, esses valores são relativamente próximos, sugerindo consistência na distância média entre as redes geradas. Além disso, as distâncias média são argumentavelmente pequenas, ainda que $3.0426 > \log_{10}(200) \approx 2.3$, sendo que os valores estão próximos ($|\log_{10}(200) - 3.0426| \approx 0.74 < 1$);
- **Coeficiente de *Clustering***: Os valores de coeficiente de *clustering* entre, aproximadamente, 0.401 e 0.436, com ume média de ≈ 0.4208 e um sd de ≈ 0.0103. Isso sugere que, nos diferentes contextos de rede, os vizinhos de um nó tendem a estar interconectados em proporções semelhantes. Assim como na análise anterior, a consistência desses valores indica uma tendência similar de agrupamento local entre as redes. O valor do coeficiente de *clustering* é relativamente elevado, indicando a existência de triângulos na rede gerada.
- **Heterogeneidade**: A medida de heterogeneidade varia entre, aproximadamente, 2.121 e 2.265. Valores mais altos indicam uma distribuição mais desigual ou heterogénea dos graus dos nodos na rede. Assim como na análise anterior, embora haja uma diferença, os valores são relativamente próximos entre as redes geradas. O valor é elevado, o que contribui para a existência de *hubs* na rede;

- **Identificação de *Hubs***: Da mesma forma que na análise anterior, a coluna "Hubs?" indica que em todas as redes geradas (indicado por "Sim"), há nodos que se destacam com uma conectividade significativamente maior em relação aos demais, sendo suscetível a existência de *hubs*.

Esta análise mostra padrões semelhantes aos identificados anteriormente, com consistência nas métricas de distância média, coeficiente de *clustering* e presença de *hubs* nas redes geradas a partir de uma clique com 20 nodos. A diferença nos valores indica variações nas características estruturais, mas a tendência geral parece ser comparável à análise anterior com uma clique de 10 nodos.

c) Comparação e comentários entre as duas cliques

Apesar da consistência de métricas anteriormente referidas, é de assinalar a existência de **coeficientes de *clustering*** e de **heterogeneidade** superiores para as redes geradas a partir das cliques de 20 nodos, comparativamente às redes geradas a partir das cliques com 10 nodos.

Para analisar mais a fundo, decidimos analisar a evolução das métricas ao longo das iterações, através das Figuras 5, 6 e 7.

A distância mínima média (distância média) corresponde à média dos comprimentos dos caminhos mais curtos da rede, e esta foi cerca de 3 unidades tanto para as redes geradas em **a)** e **b)**, contudo, este valor foi ligeiramente inferior para as redes geradas a partir de uma clique inicial de 20 nodos. Ambos os valores revelaram uma argumentável proximidade a $\log_{10}(N)$, como notado anteriormente, o que leva a elas serem distâncias pequenas.

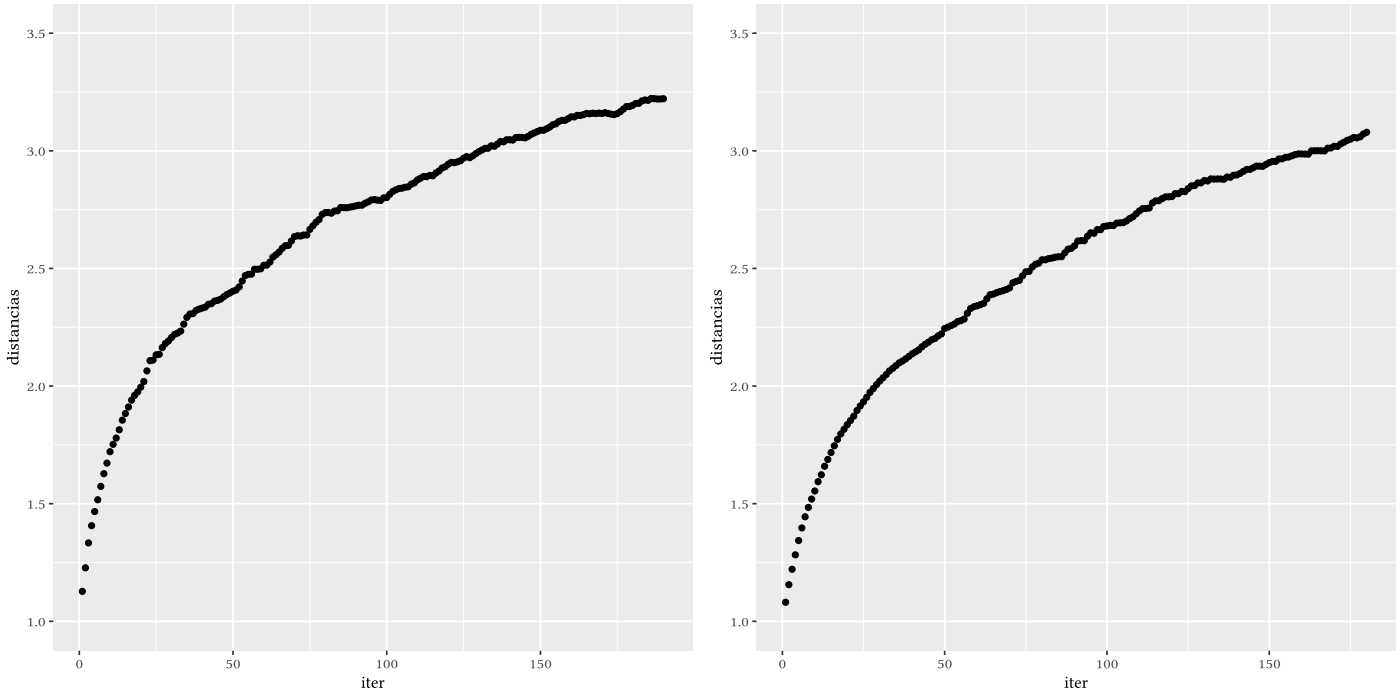


Figura 5: Evolução da distância média ao longo das iterações da rede gerada nº 7, na esquerda **a)**, na direita **b)**

Na Figura 5, encontra-se a evolução das distâncias médias ao longo das iterações, que podemos ver que se aproxima de uma função logarítmica, em que as distâncias médias vão aumentando progressivamente até começarem a estabilizar lentamente. Isto indica que as distâncias são pequenas em ambas as situações, em todos os pontos da iteração. Não parece haver grandes diferenças entre as situações também, com semelhantes distâncias médias em todo o processo.[3] As conclusões a que chegamos são semelhantes as previsões anteriores.

[3]É importante notar que na interpretação e comparação da evolução da métrica ao longo das iterações, o número de nodos N em cada iteração é diferente para as situações, sendo que começam com N diferentes, e não se pode diretamente comparar os métodos no mesmo ponto de iteração. Nós tivémos isso em conta na interpretação e comparação das métricas.

O coeficiente de *clustering* da rede (coeficiente de *clustering*) mede a tendência dos vizinhos de um nodo estarem interconectados, e ela foi maior nas redes geradas a partir de uma clique de 20 nodos. Uma clique inicial maior (com 20 nodos) oferece mais oportunidades para conexões e interações entre os nodos quando novos nodos são adicionados. Assim, há uma maior probabilidade dos nodos estarem conectados entre si, aumentando o coeficiente de *clustering*. Para além disso, a subrede completa/clique de 20 nodos começa com um maior número de triângulos e de ligações, comprovando a veracidade da intuição enunciada acima.

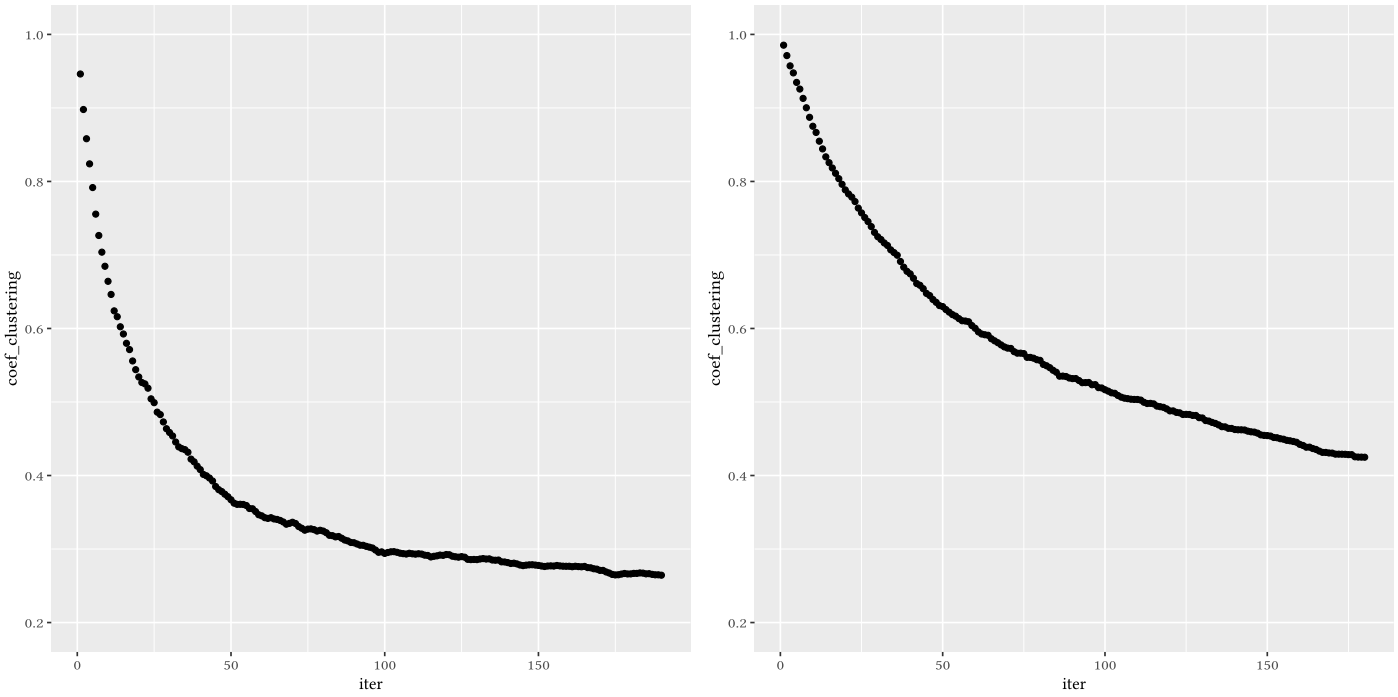


Figura 6: Evolução do coeficiente de *clustering* ao longo das iterações da rede gerada nº 7, na esquerda **a)**, na direita **b)**

A Figura 6 representa a evolução do coeficiente de *clustering* ao longo das iterações. Reparamos que os coeficientes de *clustering* encontram-se perto de 1 no início do processo (porque a distância média inicial é 1, porque é uma rede completa). Ao longo das iterações é evidente que este coeficiente diminui porque os novos nodos introduzidos vão aumentar o número de triângulos potênciais, mesmo com uma probabilidade p alta. Ainda assim, como a probabilidade é alta, o coeficiente de *clustering* começa a zar a um certo ponto, não deixando que ela desça de 0.24 e 0.4 em **a)** e **b)**, respetivamente; no entanto, parece até que o coeficiente de *clustering* não estabilizou ainda, revelando que é possível que o número desça ainda um bocado (não suficiente para o valor estagnado da situação **a)**, sendo que este fez um decréscimo mais abrupto). Ou seja, a proporção de ternos fechados face aos ternos existentes das redes manter-se-á quase constante para iterações futuras, com tendência a diminuir à medida que os novos nodos são adicionados. Podemos concluir que embora os valores comecem a estagnar a um certo ponto, a configuração inicial é crucial para um coeficiente de *clustering* elevado quando p é elevado.

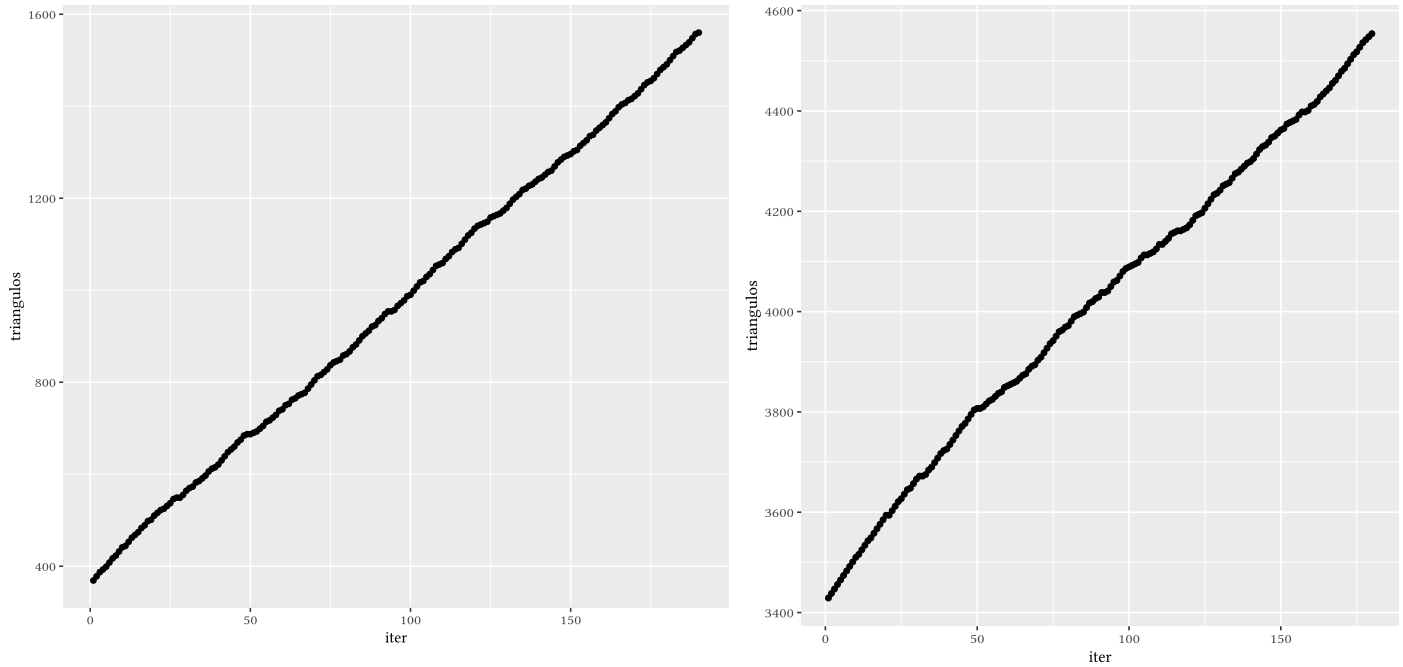


Figura 7: Evolução do número de triângulos ao longo das iterações da rede gerada nº 7, na esquerda **a)**, na direita **b)**

Quanto à Figura 7, observamos que na iteração 0 - configuração inicial - os valores são distintos devido ao número de ligações existentes em cada uma das cliques. A evolução da formação de triângulos ao longo das iterações tem um crescimento linear e estritamente positivo, devido ao facto de adicionarmos constantemente 3 ligações e estas terem sempre a mesma probabilidade de formar triângulos pelos adjacentes do nodo escolhido para a ligação com o novo nodo. O número de triângulos final é proporcional à dimensão da rede completa inicial, pelo número de ligações que existem.

Quanto à heterogeneidade, ela foi relativamente maior para as redes geradas a partir de uma clique inicial de maior dimensão (20 nodos), ao contrário do esperado. Ao começar com uma clique inicial maior pode permitir uma maior diversidade no crescimento da rede, o que significa que ao adicionar novos nodos e conexões à clique inicial maior (20 nodos), a probabilidade de estabelecer ligações com nodos de diferentes graus é potencialmente mais alta em comparação com a clique de 10 nodos. Desta forma, alguns nodos podem acumular mais conexões do que outros, criando uma diferença maior entre os nodos mais conectados e os menos conectados, sendo mais propício a uma maior probabilidade de apresentar uma distribuição de graus mais heterogénea. É possível que para maiores nodos desejados n, este de facto se aproximem ainda mais, de acordo com o esperado, e $n = 200$ não seja suficiente para essa observação.

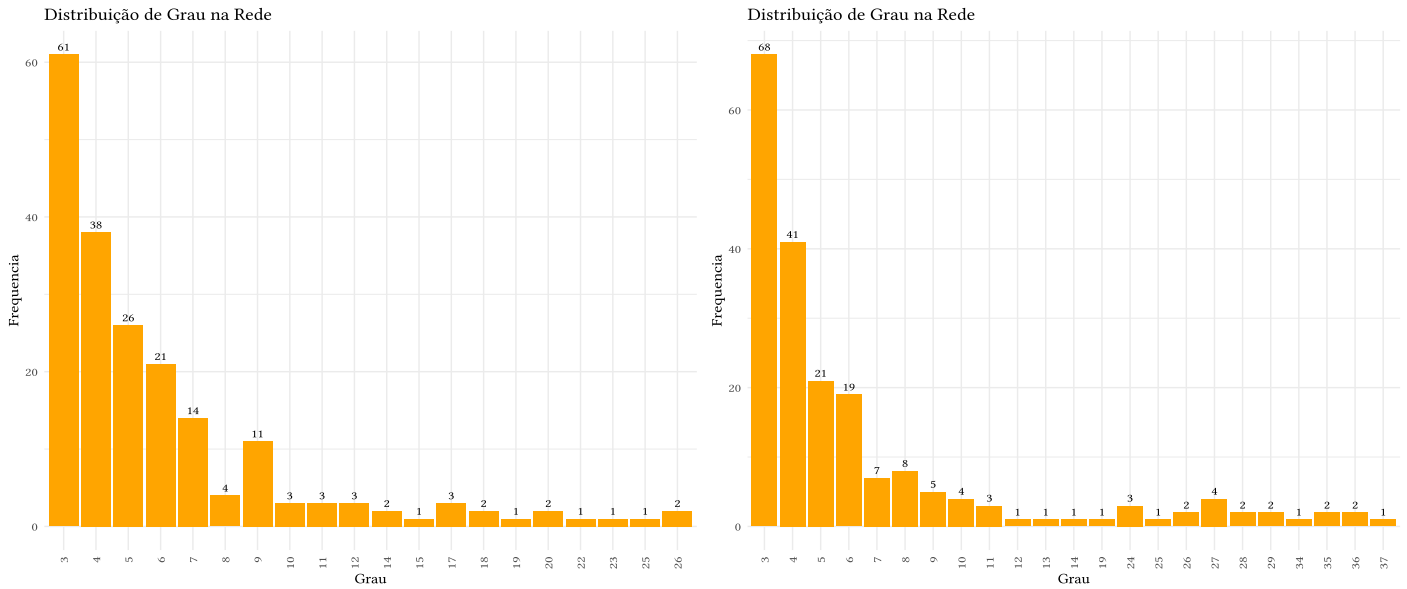


Figura 8: Distribuição de grau da rede gerada nº 7 para as 2 cliques, na esquerda **a)**, na direita **b)**

Para este valor de p, relativamente elevado, já seria expectável a formação de *hubs*. Para comprovar a heterogeneidade e, posteriormente esta presença de *hubs*, visualizamos na Figura 8 que ambas têm caudas longas, variando entre grau $k = 3$ e $k = 26$, para **a)**, ou $k = 37$, para **b)**. Os nodos com grau acima de 10 para a distribuição à esquerda e 20 para a distribuição à direita fazem parte dos nodos presentes na configuração inicial ou de eventuais nodos que foram formando várias ligações no decorrer das iterações do algoritmo.

Concluindo, a escolha do tamanho inicial da clique influenciou significativamente a estrutura e a conectividade das redes geradas, demonstrando que a configuração inicial desempenha um papel crucial na formação de agrupamentos locais e na distribuição de graus dos nodos em redes complexas, especialmente em métricas como a heterogenidade e coeficiente de *clustering*, embora não tenha afetado muito a distância média e a existência de *hubs*.

Q2: Deteção de comunidades na componente gigante de uma zona residencial

Numa rede, os nodos podem estar agrupados em subconjuntos. Se num subconjunto se observa que os seus elementos têm mais ligações dentro do subconjunto do que ligações para outros nodos, então o subconjunto constitui uma **comunidade**, ***cluster*** ou **módulo**.

Para esta questão, pretendia-se o **estudo de comunidades**, utilizando a **componente gigante** de uma rede de habitantes de uma zona residencial e a existência de contacto social direto entre os habitantes, usando vários métodos de deteção de comunidades e a avaliação e comparação dos resultados.

Para avaliar as partições pode-se comparar com redes em que não existem comunidades, como é o caso das redes aleatórias, onde não é expectável as encontrar. Assim, pode comparar-se o número de ligações internas de cada subrede da partição com o número esperado de ligações existentes numa rede aleatória. 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. Desta forma, esta medida calcula um valor que pretende medir as diferenças entre a rede em estudo e a estrutura de uma rede aleatória.

Uma partição de uma rede é uma divisão em subredes em que cada nodo pertence a exatamente uma subrede e cada subrede tem pelo menos um nodo. Apesar da identificação subredes bem separadas ser um objetivo, o seu ênfase está na separação e não na coesão que também é muito importante para avaliar os métodos de deteção de comunidades.

De forma a cumprir este objetivo, serão aplicados quatro métodos de deteção de comunidades: **remoção de pontes**, **propagação de etiquetas**, **otimização da modularidade (método *Fast Greedy*)** e de **Louvain**.

No método de **remoção de pontes** começa-se por remover pontes, isto é, ligações cuja remoção divide uma rede conexa em duas subredes. Sendo assim, é necessário um método que permita a identificação de pontes, como é o caso do algoritmo de Girvan e Newman, cuja medida é baseada na intermediação de ligações (número de caminhos mais curtos que passam por uma ligação). Em cada iteração do algoritmo avalia-se a ligação com maior intermediação, removendo-a, recalculando-se depois a intermediação de ligações, com vista a escolher a próxima ligação a ser retirada.

O método de **propagação de etiquetas** é um métodos 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.

Relativamente à **otimização de modularidade (método *Fast Greedy*)**, é uma técnica de deteção de comunidades em redes que trabalha de forma hierárquica e aglomerativa. Este algoritmo inicia considerando cada nó uma comunidade distinta e procura combinar iterativamente comunidades vizinhas para maximizar a modularidade, uma medida de quão bem os nodos estão agrupados, face a uma rede aleatória que não aparenta ter uma estrutura de comunidades bem definida. O termo "*fast*" refere-se à abordagem eficiente deste método ao selecionar, em cada etapa, a fusão que mais melhora imediatamente a estrutura das comunidades, ou seja, a escolha da fusão que resultará no maior ganho imediato da função de modularidade.

O **algoritmo de Louvain** é um método aglomerativo que agrupa as comunidades em supernodos, sendo a solução inicial composta por conjuntos singulares. Cada nodo é inicialmente atribuído à comunidade de um de seus nodos vizinhos. A escolha do nodo vizinho é feita para maximizar o aumento da modularidade em relação à configuração atual. Esse processo é repetido várias vezes, realocando os nodos até que não seja mais possível aumentar o valor da modularidade. A rede é transformada em uma rede de supernodos, onde cada comunidade identificada no passo anterior é substituída por um supernodo. As conexões entre as comunidades são representadas por pesos, os quais são iguais à soma dos pesos das conexões entre os supernodos. Além disso, são criados lacetes aos supernodos com pesos equivalentes à soma dos graus internos dos nodos originais pertencentes a cada comunidade.

Nota: De forma a garantir a reprodutibilidade dos resultados definimos uma *seed* de 777 para o método de propagação de etiquetas e o algoritmo de Louvain.

a) Aplicação dos quatro métodos de deteção de comunidades

Remoção de pontes

Para este primeiro método, a representação visual das comunidades obtidas apresentam-se na Figura 9 e a tabela com a frequência do tamanho das comunidades apresentam-se na Tabela 3.

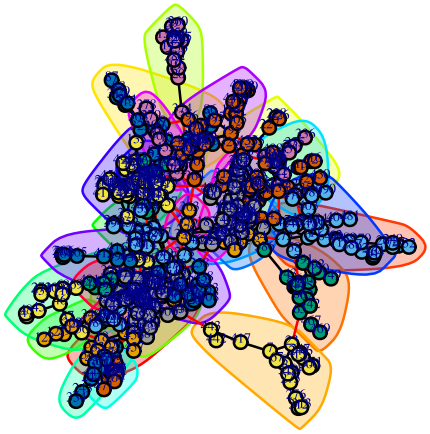


Figura 9: Representação visual das comunidades geradas pelo método de remoção de pontes

Tabela 3: Tabela com a frequência do tamanho de cada comunidade pelo método de remoção de pontes

3	5	6	7	9	11	12	15	16	17	18	20	25	30	31	33	45	62
1	1	2	1	2	2	1	2	1	1	6	1	1	1	1	1	1	1

O algoritmo identificou um total de 27 comunidades na subrede. A distribuição de tamanhos das comunidades varia consideravelmente, com tamanhos que variam de 3 a 62 nodos por comunidade, sendo a frequência do tamanho das comunidades muito variada, tirando o facto de existirem 6 comunidades de dimensão 18. A modularidade obtida foi de 0.8396989, que indica uma estrutura de comunidades forte e bastante diferente de uma rede aleatória, com o número de ligações internas de cada subrede (comunidade) bastante superior ao número esperado numa rede aleatória, sugerindo uma organização significativa na rede.

Propagação de etiquetas

Para o segundo método, a representação visual das comunidades obtidas apresentam-se na Figura 10 e a tabela com a frequência do tamanho das comunidades apresentam-se na Tabela 4.

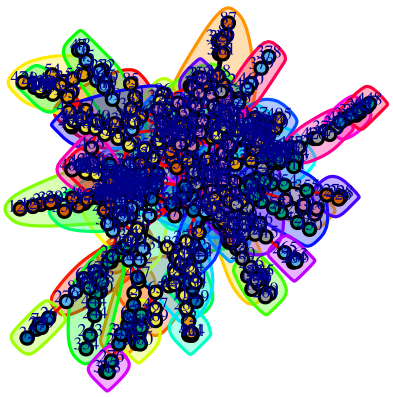


Figura 10: Representação visual das comunidades geradas pelo método de propagação de etiquetas

Tabela 4: Tabela com a frequência do tamanho de cada comunidade pelo método de propagação de etiquetas

3	4	5	6	7	8	9	10	11	12	13	14	16	18	19	20	22	23	33
4	8	9	3	2	2	3	2	3	3	1	2	1	1	1	1	1	2	1

Este método identificou 52 comunidades na rede. A distribuição de tamanhos das comunidades varia significativamente, com tamanhos que variam de 3 a 33 nodos por comunidade, havendo destaque para comunidades com dimensão até 5. A modularidade é de 0.7895216, o que indica uma boa estrutura de comunidades na componente gigante, com o número de ligações internas de cada subrede (comunidade) bastante superior ao número esperado numa rede aleatória, sugerindo uma organização significativa na rede.

Otimização de modularidade (método *Fast Greedy*)

Para o terceiro método, a representação visual das comunidades obtidas apresentam-se na Figura 11 e a tabela com a frequência do tamanho das comunidades apresentam-se na Tabela 5.

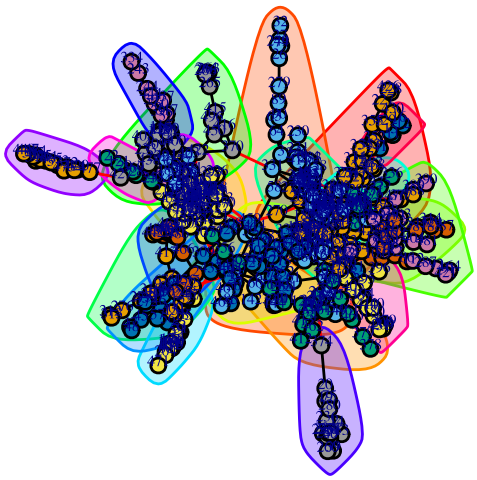


Figura 11: Representação visual das comunidades geradas pelo método *Fast Greedy*

Tabela 5: Tabela com a frequência do tamanho de cada comunidade pelo método *Fast Greedy*

5	6	11	13	14	15	16	18	19	20	26	30	35	37	57	60	63
1	2	2	1	1	1	2	2	1	1	1	1	1	1	1	1	1

Através do método *Fast Greedy*, foi possível detetar 21 comunidades, com a dimensão das comunidades a variar bastante, sendo que a menor possui 5 nodos e a maior 63 nodos, sendo a frequência da dimensão das comunidades quase equitativa. Obteve-se a modularidade de 0.8384792, sugerindo uma estrutura de comunidade muito forte e bastante diferente de uma rede aleatória, com o número de ligações internas de cada subrede (comunidade) bastante superior ao número esperado numa rede aleatória, sugerindo uma organização significativa na rede.

Algoritmo de Louvain

Para o quarto método, a representação visual das comunidades obtidas apresentam-se na Figura 12 e a tabela com a frequência do tamanho das comunidades apresentam-se na Tabela 6.

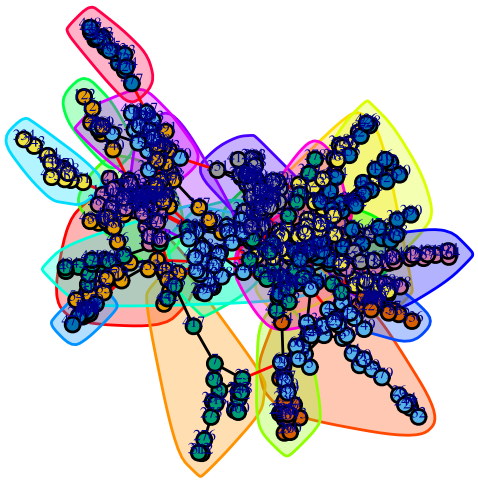


Figura 12: Representação visual das comunidades geradas pelo algoritmo de Louvain

Tabela 6: Tabela com a frequência do tamanho de cada comunidade pelo algoritmo de Louvain

6	11	12	14	16	18	19	25	26	27	29	35	44	46	52
1	1	1	2	3	2	1	1	3	1	1	1	1	1	1

Neste último método, foram identificadas 21 comunidades na componente gigante, com a dimensão das comunidades a variar entre 5 e 62, sendo a frequência da dimensão das comunidades muito equilibrada. A modularidade obtida foi de 0.8434784, o que indica uma estrutura de comunidades muito forte e significativamente diferente da estrutura de uma rede aleatória, com uma separação nítida entre os agrupamentos dos nodos, sendo o número de ligações internas de cada subrede (comunidade) bastante superior ao número esperado numa rede aleatória, sugerindo uma organização significativa na rede.

b) Comparação e comentário dos resultados obtidos com os quatro métodos

Na Tabela 7, apresenta-se uma tabela com alguns dos resultados obtidos para os quatro métodos de deteção de comunidades:

Tabela 7: Tabela com alguns dos resultados obtidos com os quatro métodos

Método	Nº comunidades	Dimensão mínima	Dimensão máxima	Modularidade
Remoção de pontes	27	3	62	0.8396989
Propagação de etiquetas	52	3	33	0.7895216
Otimização de modularidade (método *Fast Greedy*)	21	5	63	0.8384792
Algoritmo de Louvain	21	6	52	0.8434784

A partir da tabela é possível efetuar a seguinte análise:

Número de Comunidades:

- Propagação de Etiquetas: Resultou no maior número de comunidades (52).
- Remoção de Pontes, *Fast Greedy* e Algoritmo de Louvain: Geraram um número semelhante de comunidades em torno de 20 a 30.
- O número de comunidadeds é gigante, mesmo para o número de nodos presentes na componente ($N = 496$). As suas análises individuais tornam-se impossíveis com este número. O número de comunidades para os métodos estudados não parecem ser realísticos, sendo que isto representa apenas uma zona residencial e o contacto entre elas.

Dimensão das Comunidades:

- Remoção de Pontes: Apresentou a maior variação na dimensão das comunidades, indo de 3 a 62 nodos.
- Propagação de Etiquetas: Mostrou uma grande variação também, com tamanhos variando de 3 a 33 nodos.
- *Fast Greedy* e Algoritmo de *Louvain*: Tiveram uma variação da dimensão das comunidades semelhante, com a dimensão mínima em torno de 5 ou 6 nodos e a maior chegando a 60 ou 50 nodos, respetivamente.
- A grande diferença de entre o mínimo de comunidades e máximo de comunidades, mostra que existem comunidades na zona residencial com poucos indivíduos e outros com muitas pessoas, o que poderá representar uma parte da zona residencial mais habitada, e outras mais isoladas.

Modularidade:

- Algoritmo de Louvain: Apresentou a maior modularidade (0.843), seguido pelo método de remoção de pontes (0.8397) e pelo *Fast Greedy* (0.838).
- Propagação de Etiquetas: Teve um valor de modularidade um pouco mais baixos, mas ainda significativo (0.7895).
- As modularidades são todas elevadas, o que indica que a rede não parece ser aleatória e as comunidades sejam criadas aleatóriamente.

É de realçar que não se procedeu a uma análise detalhada de coesão e separação devido ao elevado número de comunidades detetado em cada um dos métodos.

Primeiramente, é de destacar que todos os métodos de deteção de comunidades apresentam modularidades elevadas, sendo indicativo de uma estrutura bastante diferente de uma rede que não tem comunidades como é o caso de uma rede aleatória, com o número de ligações internas de cada subrede (comunidade) bastante superior ao número esperado numa rede aleatória.

Assim, apesar de todos os métodos terem identificado comunidades na rede, o Algoritmo de Louvain e o método *Fast Greedy* destacaram-se pela consistência e pela maior qualidade das comunidades identificadas, demonstrando uma estrutura mais coesa e modular na rede.

Conclusão

Concluir esta análise de redes envolve destacar a riqueza estrutural e organizacional presente nos sistemas estudados. Os resultados obtidos através da aplicação de diferentes modelos de geração de redes aleatórias proporcionaram uma compreensão mais profunda das propriedades emergentes, como distância média, coeficiente de *clustering* da rede (coeficiente de *clustering*) e a presença de *hubs*, revelando padrões consistentes e nuances interessantes em relação ao tamanho inicial da clique, especialmente no modelo estudado.

Ao explorar os métodos de deteção de comunidades na componente gigante da rede social investigada, identificamos 27 comunidades através da remoção de pontes, 52 na propagação de etiquetas, 21 na otimização de modularidade (método *Fast Greedy*) e também 21 com o algoritmo de Louvain. Cada método ofereceu uma visão única das estruturas comunitárias presentes na rede, com variações na distribuição de tamanhos das comunidades e índices de modularidade. Essas discrepâncias podem refletir diferentes perspetivas na identificação de agrupamentos de nodos, oferecendo *insights* valiosos sobre a organização intrínseca da rede. A forte modularidade observada em todos os métodos sugere uma estrutura de comunidades bem definida na componente gigante da rede, com nodos agrupados de maneira significativa e clara, no entanto, o elevado número de comunidades indica valores não relísticos, especialmente no contexto provisionado. A diversidade nas dimensões das comunidades evidencia a presença de agrupamentos heterogéneos, onde alguns conjuntos de nodos apresentam ligações mais densas entre si do que com o restante da rede.

No contexto do estudo de redes, essas descobertas ressaltam a importância de aplicar múltiplos métodos para identificar comunidades, evidenciando a complexidade e riqueza das interações entre os elementos do sistema. Essa abordagem multidisciplinar contribui para uma compreensão mais completa das estruturas sociais e dos padrões de interconexão, desempenhando um papel fundamental na análise de sistemas complexos.

Esses resultados não apenas aprofundaram o nosso conhecimento sobre as características estruturais das redes, a sua geração aleatória e a deteção de comunidades mas também destacam a relevância e a diversidade de ferramentas analíticas disponíveis para investigar esses sistemas complexos e interativos.

Apêndices

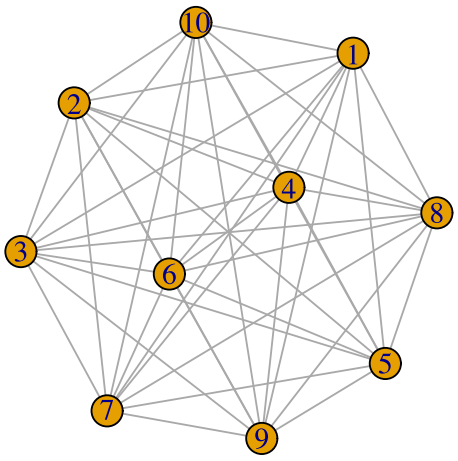
Apêndice A - Importação das bibliotecas necessárias

```
library(igraph)
library(conflicted)
library(magrittr)
library(icecream)
library(restorepoint)
library(tibble)
library(ggplot2)
```

Questão 1

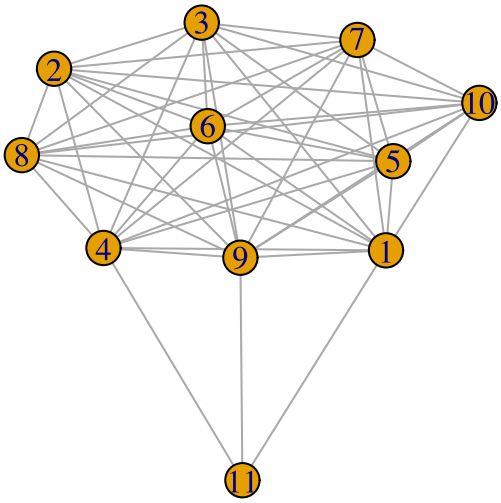
Apêndice B1 - Clique com 10 nodos e algoritmo genérico

```
## Clique com 10 nodos

set.seed(1)
graph.full(10, directed = F) -> g
set.seed(1)
g |> plot()
```




```
get_neighbors <- function(graph, node) {
  # dps n da pra faer vetor deles se n fizer isto
  neighbors(graph, node) %>% as.integer()
}

# 0 ⩽ p ⩽ 1
```

```r
# 1 ⩽ n_ligacoes

# Função para fazer uma iteração do modelo de Random Walk
random_walk_model_iter ← function(graph, p = 0.8, n_ligacoes = 3,debug_ = F, i = FALSE) {
  if(debug_) {
    ic_enable()
  } else {
    ic_disable()
  }
  if (isFALSE(i)) {
    restore.point(paste0("iter_",i), to.global = TRUE)
  }
  graph %<>% add_vertices(1)
  # vars
  nodo_ligado      ← sample(1:(vcount(graph) - 1), 1)
  ic(nodo_ligado)
  nodo_novo        ← vcount(graph)
  ic(nodo_novo)


  neighbors_de_ligado ← get_neighbors(graph, nodo_ligado)
  ic(neighbors_de_ligado)
  graph %<>% add_edges(c(nodo_novo, nodo_ligado))

  # dps de ter o primeiro link, comecar o loop
  for (i in 2:n_ligacoes) {
    ic(i)
    # escolher um vizinho do nodo ligado (ainda não escolhido)
    nodo_para_aceitar ← setdiff(neighbors_de_ligado, get_neighbors(graph, nodo_novo)) %>% sample(1)
    ic(nodo_para_aceitar)
    u ← ic(runif(1))
    if(ic(u < p)) { # aceitar o vizinho
      graph %<>% add_edges(c(nodo_para_aceitar %>% as.integer(), nodo_novo))
    } else { # escolher outro sem ser o vizinho escolhido (ver nota)
      nodo_para_ligar ← 1:vcount(graph) %>%
        # todos excepto o rejeitado, os já ligados e o próprio nodo novo
        setdiff(c(nodo_para_aceitar, get_neighbors(graph, nodo_novo), nodo_novo)) %>%
        sample(1)
      graph %<>% add_edges(c(nodo_para_ligar, nodo_novo))
    }
  }
  graph
}
set.seed(1)
g %>% random_walk_model_iter(debug_ = T) %>%  plot()
```

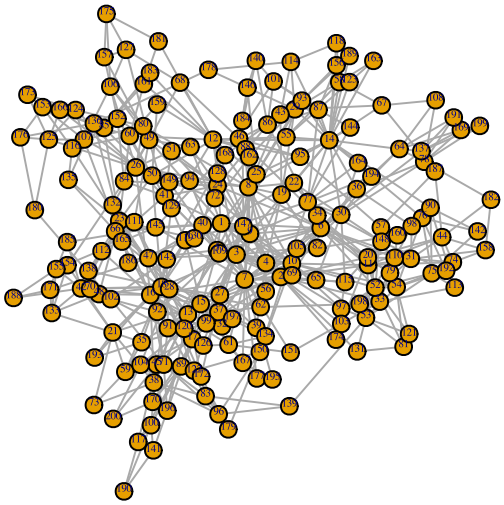


,

```
# Função para fazer o modelo de Random Walk para uma rede com 200 nodos
random_walk_model ← function(g, nodes_wanted = 200, p = 0.8, n_ligacoes = 3, debug_ = F, seed =
1) {
  set.seed(seed)
  for (i in 1:(nodes_wanted - vcount(g))) {
    g %<>% random_walk_model_iter(p = p, n_ligacoes = n_ligacoes, i = ifelse(debug_, i, FALSE))
  }
  g
}

random_walk_model_w_stats ← function(g, nodes_wanted = 200, p = 0.8, n_ligacoes = 3, debug_ = F,
seed = 1) {
  set.seed(seed)
  metricas ← tribble(~iter, ~distancias,  ~coef_clustering, ~triangulos)
  for (i in 1:(nodes_wanted - vcount(g))) {
    g %<>% random_walk_model_iter(p = p, n_ligacoes = n_ligacoes, i = ifelse(debug_, i, FALSE))
    metricas %<>% add_row(tibble(
      iter = i,
      distancias = mean_distance(g),
      coef_clustering = transitivity(g, type = "global"),
      triangulos = sum(count_triangles(g))
    ))
  }
```

```
  list(graph = g, metricas = metricas)
}


tab ← random_walk_model_w_stats(g, 200, 0.8, 3, T, 1)
tab

$metricas
# A tibble: 190 × 4
    iter distancias coef_clustering triangulos
   <int>      <dbl>           <dbl>      <dbl>
 1     1       1.13           0.946        369
 2     2       1.23           0.898        378
 3     3       1.33           0.858        387
 4     4       1.41           0.824        393
 5     5       1.47           0.792        399
 6     6       1.52           0.756        408
 7     7       1.57           0.726        417
 8     8       1.63           0.704        423
 9     9       1.67           0.685        432
10    10       1.72           0.664        441
# ℹ 180 more rows
# ℹ Use `print(n = ...)` to see more rows


ggplot(tab$metricas, aes(x = iter, y = distancias)) + geom_point()
ggplot(tab$metricas, aes(x = iter, y = coef_clustering)) + geom_point()
ggplot(tab$metricas, aes(x = iter, y = triangulos)) + geom_point()

g_dps ← g %>% random_walk_model()
ic_enable()
ic(vcount(g_dps) == 200)
ic(is.simple(g_dps))
ic(ecount(g_dps) == (200-vcount(g))*3 + ecount(g))
g_dps %>% plot(vertex.size = 7, vertex.label.cex = 0.35)




# o que esta em cima mas funcao
# Gerar o número de redes desejado (how_many) com determinado nº nodos (nodes_wanted - 200)
make_graphs ← function(initial_g, how_many, nodes_wanted = 200, p = 0.8, n_ligacoes = 3, debug_ =
F, seed = 1) {
  set.seed(seed)
```

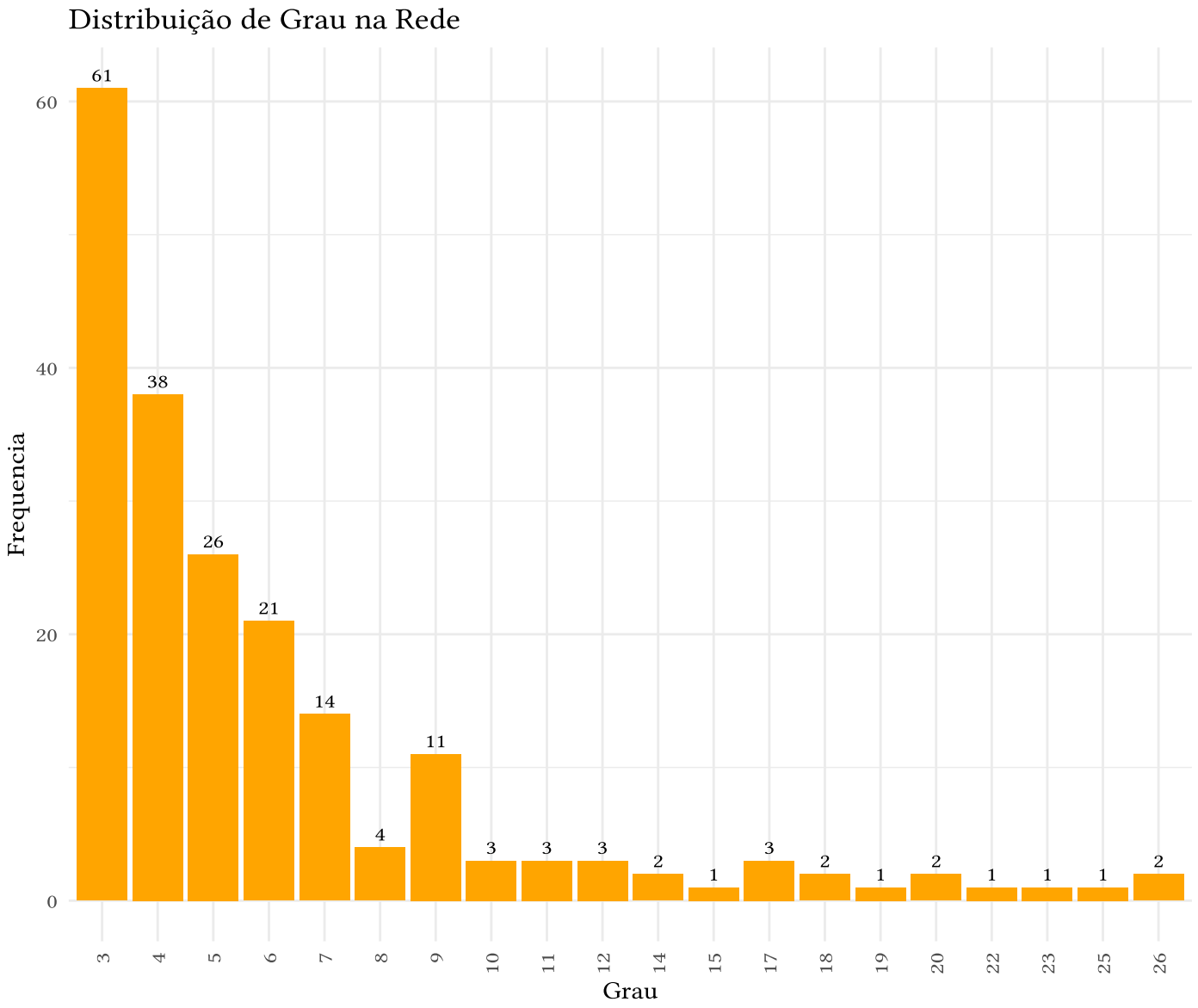
```
  graphs <- list()
  for (i in 1:how_many) {
    g_dps <- initial_g
    for (j in 1:(nodes_wanted - vcount(initial_g))) {
        g_dps %<>% random_walk_model_iter(p = p, n_ligacoes = n_ligacoes, i = ifelse(debug_, j,
FALSE))
    }
    graphs[[i]] <- g_dps
  }
  graphs
}
make_graphs(g, 10) -> graphs
graphs[[7]] %>% plot(vertex.size = 7, vertex.label.cex = 0.35)
```



```
(dist_graus_rede_7 <- table(degree(graphs[[7]])))
3  4  5  6  7  8  9 10 11 12 14 15 17 18 19 20 22 23 25 26
61 38 26 21 14  4 11  3  3  3  2  1  3  2  1  2  1  1  1  2

(degree_df <- data.frame(table(degree(graphs[[7]]))))
Var1 Freq
1    3   61
2    4   38
3    5   26
4    6   21
5    7   14
```

```
6     8    4
7     9   11
8    10    3
9    11    3
10   12    3
11   14    2
12   15    1
13   17    3
14   18    2
15   19    1
16   20    2
17   22    1
18   23    1
19   25    1
20   26    2
colnames(degree_df) ← c("Grau", "Frequência")
degree_df
Grau Frequência
1     3         61
2     4         38
3     5         26
4     6         21
5     7         14
6     8          4
7     9         11
8    10          3
9    11          3
10   12          3
11   14          2
12   15          1
13   17          3
14   18          2
15   19          1
16   20          2
17   22          1
18   23          1
19   25          1
20   26          2



ggplot(degree_df, aes(x = factor(Grau), y = Frequência)) +
  geom_bar(stat = "identity", fill = "orange") +
  geom_text(aes(label = Frequência), vjust = -0.5, size = 3) +
  labs(x = "Grau", y = "Frequencia", title = "Distribuição de Grau na Rede") +
  theme_minimal() +
```

```
  theme(axis.text.x = element_text(angle = 90, vjust = 0.5, hjust = 1))
ggsave("distribuicao_grau_rede_7_clique_10_nodos.svg")
```




```
# calcular as métricas
calculate_metrics ← function(graph) {
  mean_distance ← mean_distance(graph, directed = FALSE, unconnected = TRUE)
  clustering_coef ← transitivity(graph, type = "global")
  deg ← degree(graph, mode = "all")
  ht ← mean(deg^2)/mean(deg)^2

  tibble(
    Mean_Distance = mean_distance,
    Clustering_Coefficient = clustering_coef,
    Heterogeneity = ht
  )
}
calculate_metrics_graphs ← function(graphs) {
  lapply(graphs, calculate_metrics) → results
```

```
  tibble(
    Rede = 1:length(graphs),
    Distancia_Media = sapply(results, \(x) x$Mean_Distance),
    Coeficiente_de_Clustering = sapply(results, \(x) x$Clustering_Coefficient),
    Heterogeneidade = sapply(results, \(x) x$Heterogeneity)
  )
}

(res_clique_inicial_10_nodos <- calculate_metrics_graphs(graphs))
```

```
# A tibble: 10 × 5
    Rede Distancia_Media Coeficiente_de_Clustering Coeficiente_de_Clusterin…¹ Heterogeneidade
   <int>           <dbl>                     <dbl>                     <dbl>           <dbl>
 1     1            3.22                     0.264                     0.510            1.72
 2     2            3.32                     0.275                     0.465            1.57
 3     3            3.32                     0.277                     0.449            1.55
 4     4            3.26                     0.259                     0.434            1.61
 5     5            3.20                     0.261                     0.485            1.67
 6     6            3.16                     0.252                     0.479            1.72
 7     7            3.44                     0.302                     0.482            1.56
 8     8            3.29                     0.268                     0.448            1.59
 9     9            3.23                     0.276                     0.491            1.65
10    10            3.18                     0.245                     0.458            1.76
# ℹ abbreviated name: ¹Coeficiente_de_Clustering_Medio
```

```
mean(res_clique_inicial_10_nodos$Distancia_Media)
```

```
3.263673
```

```
sd(res_clique_inicial_10_nodos$Distancia_Media)
```

```
0.08383555
```

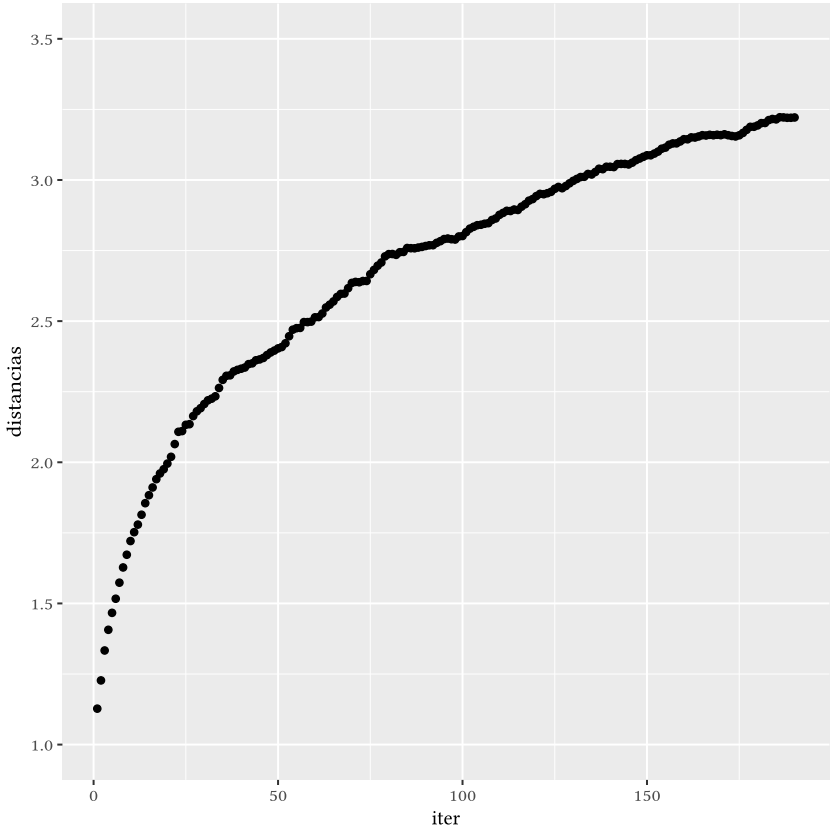
```
mean(res_clique_inicial_10_nodos$Coeficiente_de_Clustering)
```

```
0.2678931
```

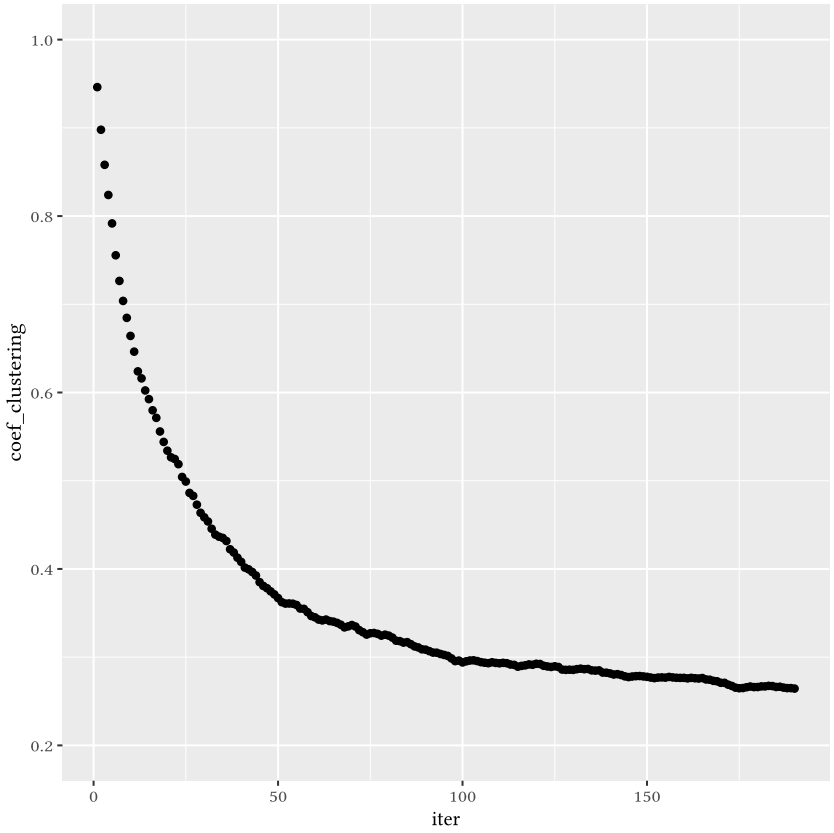
```
sd(res_clique_inicial_10_nodos$Coeficiente_de_Clustering)
```

```
0.01578428
```

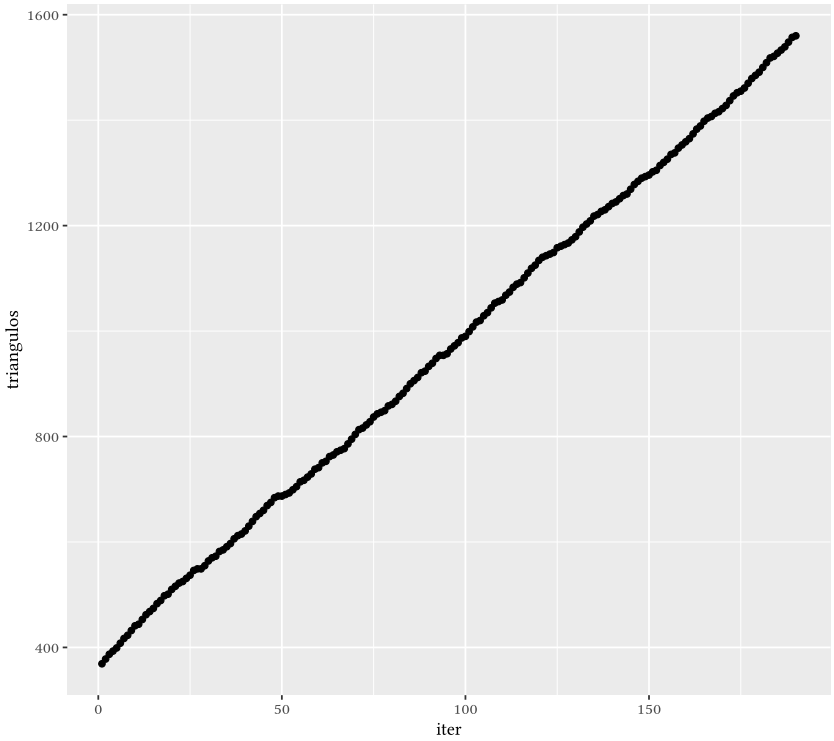
```
ggplot(tab$metricas, aes(x = iter, y = distancias)) + geom_point() + ylim(1, 3.5)
ggsave("distancias_rede_clique_inicial_10_nodos.svg")
```




```
ggplot(tab$metricas, aes(x = iter, y = coef_clustering)) + geom_point() + ylim(0.2,1)
ggsave("coef_clustering_rede_clique_inicial_10_nodos.svg.svg")
```

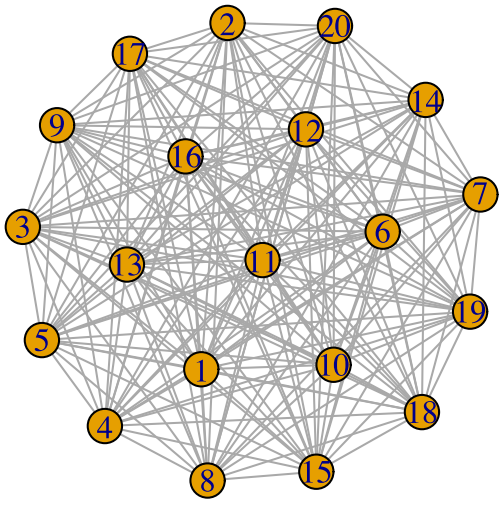



```
ggplot(tab$metricas, aes(x = iter, y = triangulos)) + geom_point()
ggsave("triangulos_rede_clique_inicial_10_nodos.svg")
```



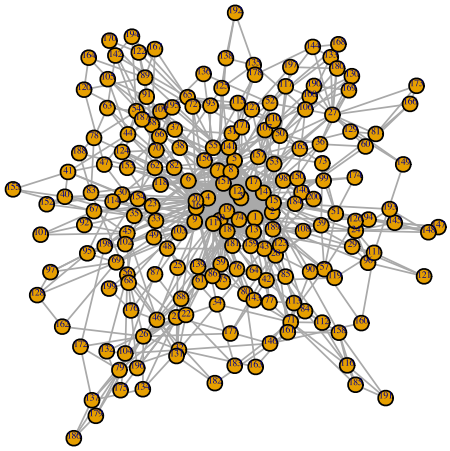
Apêndice C1 - Clique com 20 nodos

```
set.seed(1)
graph.full(20, directed = F) -> g2
set.seed(1)
g2 %>% plot()
```



,

```
make_graphs(g2, 10, seed = 1) → graphs2
calculate_metrics_graphs(graphs2)
graphs2[[7]] %>% plot(vertex.size = 7, vertex.label.cex = 0.35)
ggsave("rede_clique_inicial_20_nodos.svg")
```



,

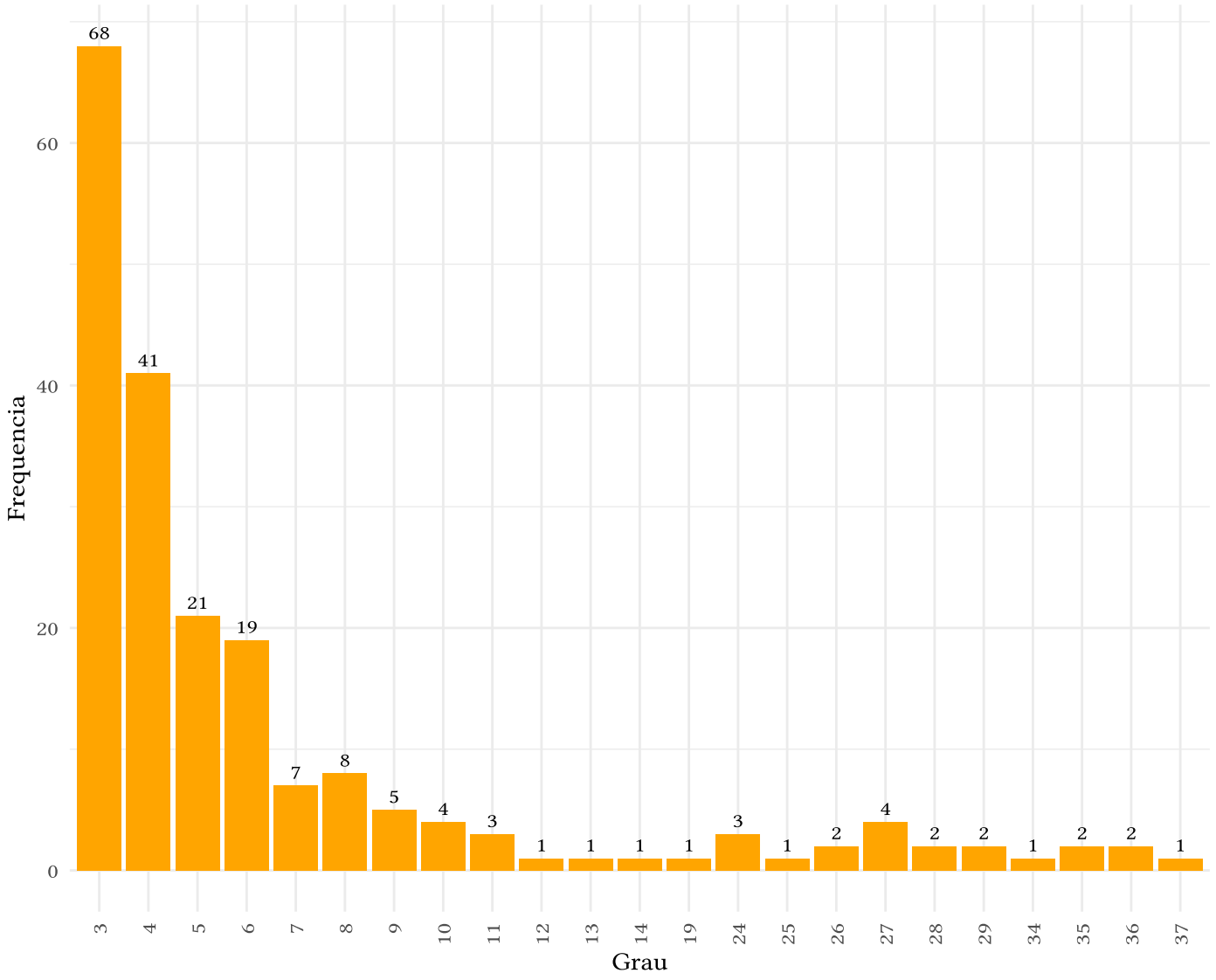
```
(dist_graus_rede_7 ← table(degree(graphs2[[7]])))
3  4  5  6  7  8  9 10 11 12 13 14 19 24 25 26 27 28 29 34 35 36 37
68 41 21 19  7  8  5  4  3  1  1  1  1  3  1  2  4  2  2  1  2  2  1

(degree_df ← data.frame(table(degree(graphs2[[7]]))))
 Var1 Freq
1     3   68
2     4   41
3     5   21
4     6   19
5     7    7
6     8    8
7     9    5
8    10    4
9    11    3
10   12    1
11   13    1
12   14    1
13   19    1
14   24    3
15   25    1
```

```
16   26    2
17   27    4
18   28    2
19   29    2
20   34    1
21   35    2
22   36    2
23   37    1
```

```
colnames(degree_df) ← c("Grau", "Frequência")
degree_df
```

```
Grau Frequência
1     3         68
2     4         41
3     5         21
4     6         19
5     7          7
6     8          8
7     9          5
8    10          4
9    11          3
10   12          1
11   13          1
12   14          1
13   19          1
14   24          3
15   25          1
16   26          2
17   27          4
18   28          2
19   29          2
20   34          1
21   35          2
22   36          2
23   37          1
```

```
ggplot(degree_df, aes(x = factor(Grau), y = Frequência)) +
  geom_bar(stat = "identity", fill = "orange") +
  geom_text(aes(label = Frequência), vjust = -0.5, size = 3) +
  labs(x = "Grau", y = "Frequencia", title = "Distribuição de Grau na Rede") +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 90, vjust = 0.5, hjust = 1))
ggsave("distribuicao_grau_rede_7_clique_20_nodos.svg")
```



,

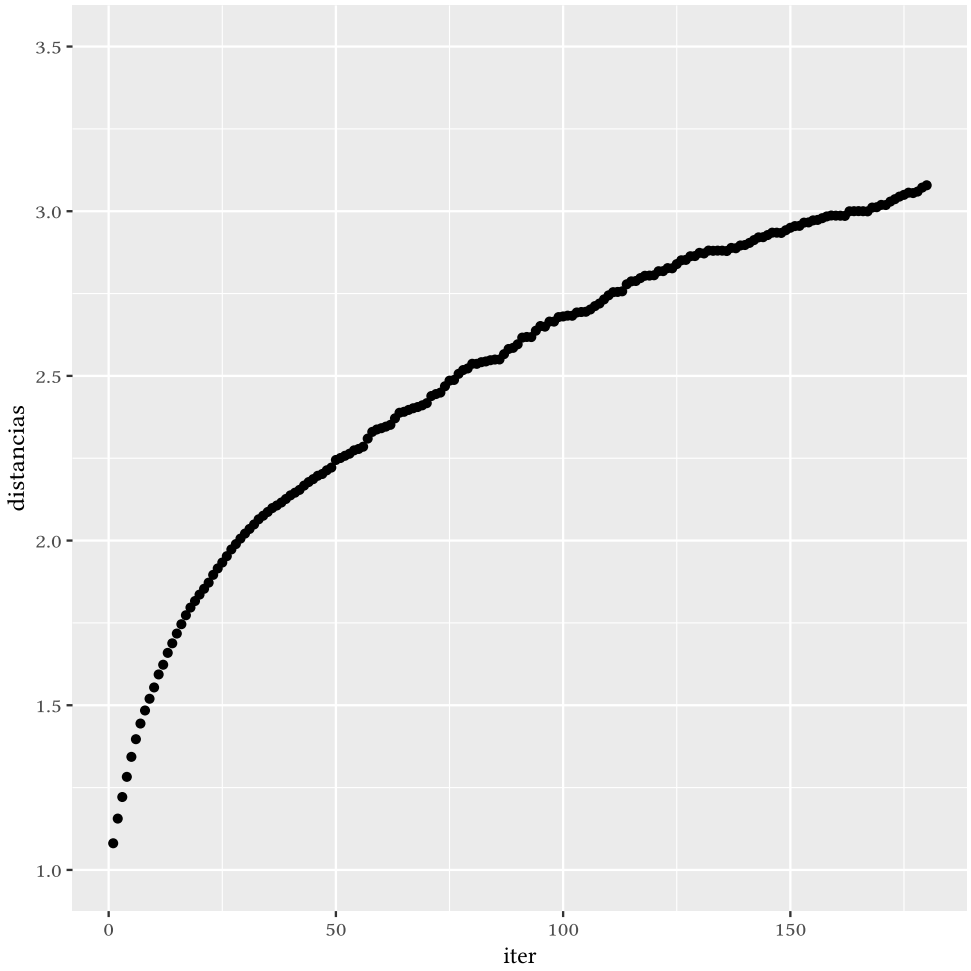
```
# A tibble: 10 × 5
    Rede Distancia_Media Coeficiente_de_Clustering Coeficiente_de_Clusterin…¹ Heterogeneidade
   <int>           <dbl>                     <dbl>                      <dbl>           <dbl>
 1     1            3.08                     0.425                      0.520            2.15
 2     2            3.02                     0.412                      0.540            2.23
 3     3            3.03                     0.424                      0.508            2.16
 4     4            2.99                     0.412                      0.542            2.23
 5     5            3.09                     0.435                      0.544            2.12
 6     6            2.97                     0.401                      0.513            2.26
 7     7            3.05                     0.427                      0.523            2.14
 8     8            3.02                     0.417                      0.530            2.20
 9     9            3.09                     0.431                      0.547            2.13
10    10            3.08                     0.425                      0.509            2.14
# ℹ abbreviated name: ¹Coeficiente_de_Clustering_Medio
```

```
mean(res_clique_inicial_20_nodos$Distancia_Media)
```

```
3.042633
```

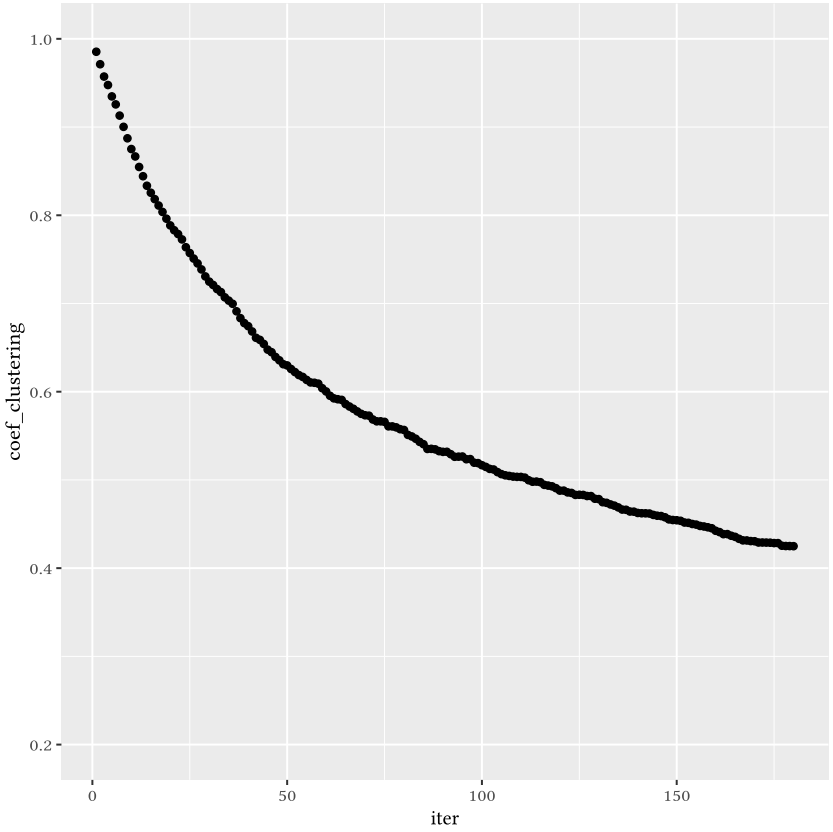
```
sd(res_clique_inicial_20_nodos$Distancia_Media)
0.04203878

mean(res_clique_inicial_20_nodos$Coeficiente_de_Clustering)
0.4208

sd(res_clique_inicial_20_nodos$Coeficiente_de_Clustering)
0.01033473
```

```
ggplot(tab2$metricas, aes(x = iter, y = distancias)) + geom_point() + ylim(1, 3.5)
ggsave("distancias_rede_clique_inicial_20_nodos.svg")
```



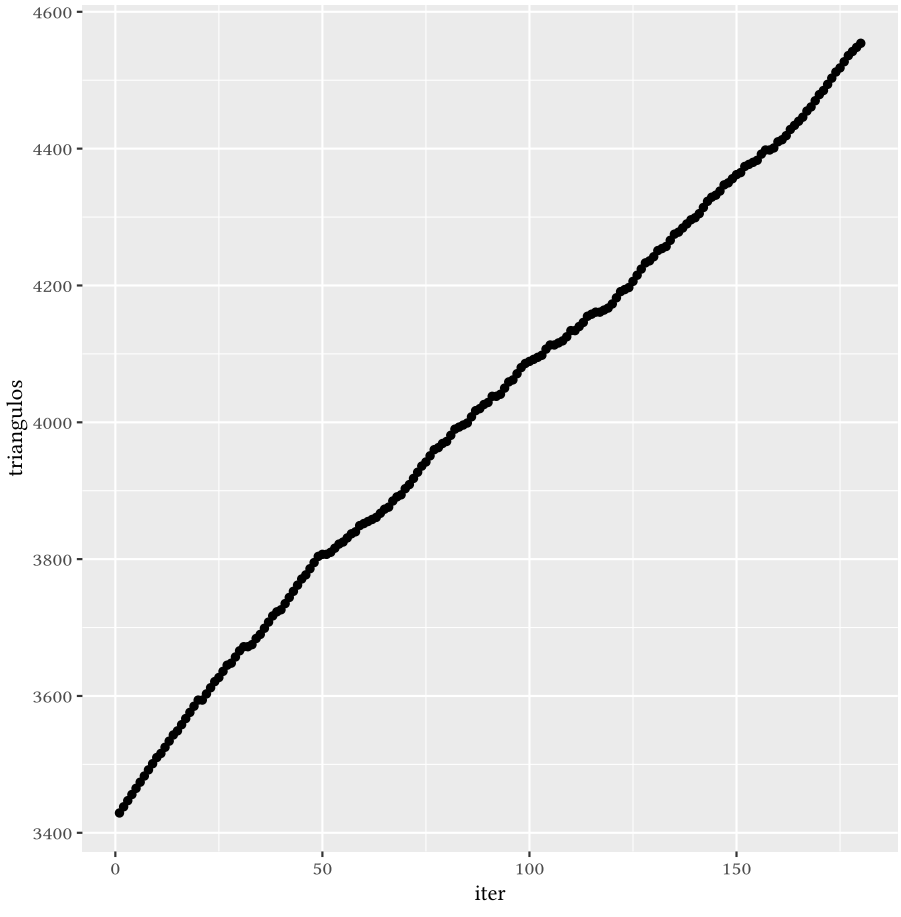
,

```
ggplot(tab2$metricas, aes(x = iter, y = coef_clustering)) + geom_point() + ylim(0.2,1)
ggsave("coef_clustering_rede_clique_inicial_20_nodos.svg")
```



,

```
ggplot(tab2$metricas, aes(x = iter, y = triangulos)) + geom_point()
ggsave("triangulos_rede_clique_inicial_20_nodos.svg")
```

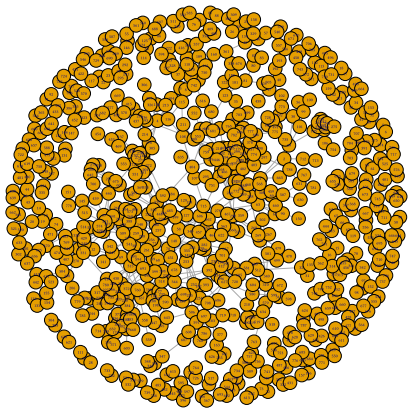


,

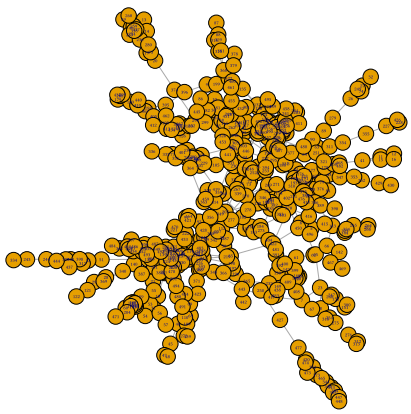
Questão 2

Apêndice B2 - Rede e componente gigante

```
#### Questão 2 ####



# leitura da rede
rede <- read_graph("trab_links.txt", format = c("edgelist"), directed=F) # com estes args para
reduzir tamanho do grafo
```



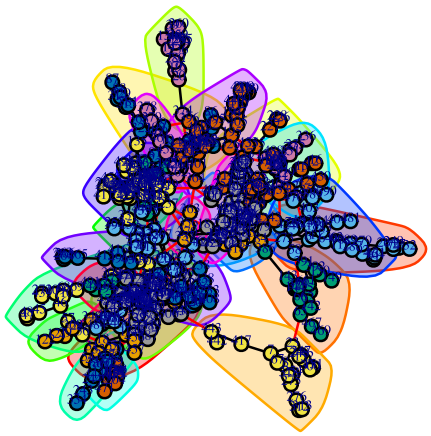
```
# filtragem da componente gigante
componentes <- components(rede)
maior_componente <- which.max(componentes$csize)
(componente_gigante <- induced.subgraph(rede, which(componentes$membership == maior_componente)))
plot(componente_gigante, vertex.size = 7, vertex.label.cex = .35)
```



```
vcount(componente_gigante)
496
ecount(componente_gigante)
984
```

Apêndice C2 - Método de remoção de pontes

```
## Identificação de comunidades através da remoção de pontes

crm ← cluster_edge_betweenness(componente_gigante)
plot(crm, componente_gigante, vertex.size = 7, vertex.label.cex = .35)
```



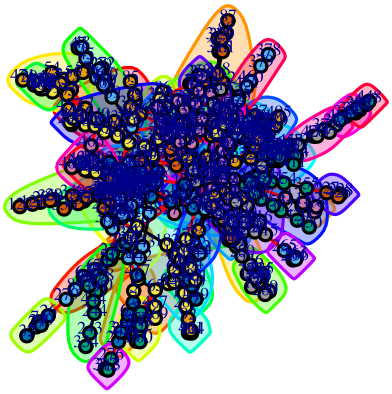
```
# numero de comunidades
length(crm)
27

# tamanho das comunidades
sizes(crm)
Community sizes
 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 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

print(paste("Comunidade de tamanho mínimo:", min(sizes(crm))))
"Comunidade de tamanho mínimo: 3"

print(paste("Comunidade de tamanho máximo:", max(sizes(crm))))
```

```
"Comunidade de tamanho máximo: 62"

# tabela com a frequência de cada tamanho de comunidade
table(sizes(crm))
3  5  6  7  9 11 12 15 16 17 18 20 25 30 31 33 45 62
1  1  2  1  2  2  1  2  1  1  6  1  1  1  1  1  1  1

membership(crm)
[1]  1  1  2  2  3  3  4  4  5  5  6  6  7  7  6  6  1  1  7  7  8  8  4  5  5  2  2  9  9  5  1  2
[33] 10 10  1  1 11  5  7  7  6  6  6  6 12 12 12 12  1  1 13 13  1  1  1 12 12  2 14 14  3  3  2  2
[65]  3  3  4 12 12  8 10 10 10  2  7  2 15 15 16 16  5  5 17 17  1  5  5  5  2  2 18 18  8  8 19 19
[97] 19 15 15 15 15 20 20  8 15 19  8  8 12 12 18 18  8  8  8 19 19 21 21 21  9  9 22 22 22 22 19
16
[129] 16 15  8  8 14 14 20 20 20  8  8  8  8  3  3  3  8  8 21 21  3  3  4  4 16 16 20 20 20 20 20
20
[161] 22 22 22 18 19 19 14 14 14 14 14 14 20 20 20  6  6  6 22 22 18 18 18 18 16 16 16 16 16 16 16
16
[193] 16 22 22  8  8  8  5 15 15 15 15 14  8  8 10 22 22  8 17 17  8  8  8  8  8  8  8  8  8  8  8
8
[225]  8  8  6  6 15  2  2 21 21  7  7  7 20 20 20 11 20 20 13 13 20  2  4  4  2  6  6 10 10  4  8
8
[257]  8 21  7  7 23 23 21 21 21 21 10 10 16 16 16 12 18 20  4  4 10 10  2  7  8 20 20 20 20 20 18
8
[289] 20 20 20 20 20 20  6 16 23 22 22  4 19 22 11 11 11  8  8 12 12 16  6  4  4  5  5 22 13  4 20
20
[321]  6  5  3  3 20  6  6 16  5  8 20 16 11 13 15 15  7  7 10  9  1  1 24 24 24 10  6  8 10  8 15
6
[353]  6  6  6 25 25 15 15 13 26 26 25 22 21 21 22  5  9 21 27 24 24  7 16 16  6 23 23 16  5 16 16
24
[385] 24  3  8  8 18 18  3  3  8 10 22 22 13 13 21 21 22 16 18 18 22 16 16 21 21 22 20 22 22 10 21
21
[417]  4 17 10 24 18 21 27 27 20  6 24 12  6  6 22 19  1 26 26 20 13 21 12 22 11  4  4 13 22 10 24
24
[449]  8 21 23  8  6 22 23  1  1 20 10 23 23 22 26 26 24 26  3  6  3  8 14 20 24 24  8 18 24  8 20
27
[481] 18  6  8 26  8 26  8  6 20 20 27  8  8 27 20 21

# modularidade
modularity(crm)
0.8396989
```

Apêndice D2 - Método de propagação de etiquetas

```
## Identificação de comunidades através da propagação de etiquetas

set.seed(777)
```

```
cpe ← cluster_label_prop(componente_gigante)
plot(cpe, componente_gigante, vertex.size = 7, vertex.label.cex = .35)
```



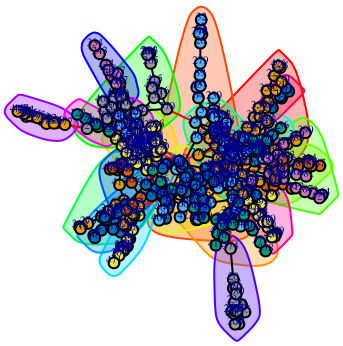
```
# numero de comunidades
length(cpe)
52

# tamanho das comunidades
sizes(cpe)

Community sizes
 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33
34
13  7  5  8  7 12  5  6  9 33  6  8  5 23 20  9 11 11 11 10  5 16 16  5  4 12 23 12 18  5 19 10 22
5
35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52
14 13  4  3  3 13  4  4  3  5  4  3  9  6  4  5  4  4

print(paste("Comunidade de tamanho mínimo:", min(sizes(cpe))))
"Comunidade de tamanho mínimo: 3"

print(paste("Comunidade de tamanho máximo:", max(sizes(cpe))))
"Comunidade de tamanho máximo: 33"
```

```
table(sizes(cpe))
3  4  5  6  7  8  9 10 11 12 13 14 16 18 19 20 22 23 33
4  8  9  3  2  2  3  2  3  3  3  1  2  1  1  1  1  2  1

membership(cpe)
[1]  1  1  2  3  4  4  5  5  6  6  7  7  8  8  7  7  1  9  8  8 10 10 11 12 12 13 13 14 14 12  9 13
[33] 15 15  1  1 15 15 16 16  7 17 17 17 18 18 18 18  1  1 19 19  9  9  9  9  9  2 20  9 21  4  2  2
[65]  4  4  5 18 18 10 15 15 15 13  8  3 22 22 23 23 15 12 24 24  1 12  6  6  2  2 25 25 10 10 26
26
[97] 26 22 22 22 22 27 27 14 22 26 10 14 18 18 28 28 14 29 14 26 26 30 30 30 14 14 31 31 31 31 26
32
[129] 32 22 10 10 20 20  6 33 33 10 14 10 10  4 34 34 10 10 30 30 21 21 11 11 32 32 33 33 33 33 27
33
[161] 31 31 31 25 26 26 20 20 20 20 20 20 33 33 33 17 23 35 36 36 28 28 28 28 32 32 32 23 23 23 23
23
[193] 23 36 36 37 37 37  6 22 22 22 22 20 14 14 15 36 36 29 24 24 29 29 29 29 29 29 37 14 14 14 10
10
[225] 10 29 38 38 22  3  3 29 29 16 16 16 33 27 33 26 27 27 19 19 33  3 11 11 13 35 35 39 39  5 10
14
[257] 10 40 16 16 41 41 42 42 42 42 15 15 23 33 23 18 28 27  5 43 15 15  2  8 10 27 27 27 33 27 28
10
[289] 27 27 27 27 27 27 17 23 41 31 31 11 26 31 44 44 44 10 10 10 18 23 35 43 43  6  6 31 19  5 33
27
[321] 35  6 34 34 33 12 12 23  6 10 27 32 44 19 22 22 16 16 15 14  1  1 45 45 45 15 35 10 15 29 22
35
[353] 35 35 35 46 46 28 28 19 47 47 46 31 29 29 36  6 14 40 29 48 48 31 49 49 17 50 50 49  6 32 32
51
[385] 51 34 10 10 17 17 21 21 29 39 36 36 19 19 40 40 31 49 17 28 36 23 23 40 40 36 27 31 31 15 40
40
[417]  5 24 15 48 25 40 29 52 27 38 40 10 35 35 36 26  1 47 47 27 19 40 18 36 44 10 10 19 31 15 51
51
[449] 14 40 41 14 12 31 50  1  1 27 15 50 50 31 47 47 45 47  4 35  4 14  9 33 48 48 10 17 48 10 33
52
[481] 28 35 29 47 10 47 14 17 33 33 52 14 14 52 33 40

# modularidade
modularity(cpe)
0.7895216
```

Apêndice E2 - Otimização de modularidade (método *Fast Greedy*)

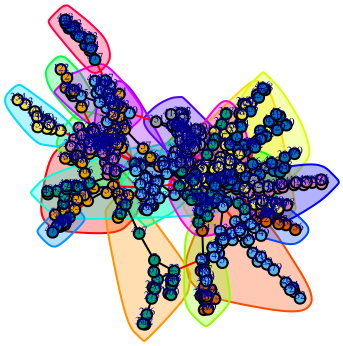
```
## Identificação de comunidades através da otimização da modularidade
## (método Fast Greedy)
```

```
com ← cluster_fast_greedy(componente_gigante)
plot(com, componente_gigante, vertex.size = 7, vertex.label.cex = .35)
```



Apêndice F2 - Algoritmo Louvain

```
## Identificação de comunidades através de otimização da modularidade
## (método Louvain)

set.seed(777)
cl ← cluster_louvain(componente_gigante)
plot(cl, componente_gigante, vertex.size = 7, vertex.label.cex = .35)
```



```
# numero de comunidades
length(cl)
21

# tamanho das comunidades
sizes(cl)
Community sizes
 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21
27 35 18 26 52 16 46 19  6 26 18 11 12 16 16 44 25 26 29 14 14

print(paste("Comunidade de tamanho mínimo:", min(sizes(cl))))
"Comunidade de tamanho mínimo: 6"

print(paste("Comunidade de tamanho máximo:", max(sizes(cl))))
"Comunidade de tamanho máximo: 52"

table(sizes(cl))
6 11 12 14 16 18 19 25 26 27 29 35 44 46 52
1  1  1  2  3  2  1  1  3  1  1  1  1  1  1

membership(cl)
[1]  1  1  2  2  2  2  3  3  4  4  5  5  6  6  5  5  1  1  6  6  7  7  3  8  8  2  2  9  9  4  1  2
[33] 10 10  1  1 10  4  6  6  5  5  5  5 11 11 11 11  1  1 12 12  1  1  1 11 11  2 13 13  2  2  2  2
[65]  2  2  3 11 11  7 10 10 10  2  6  2 14 14  5  5  4  8  5  5  1  8  4  4  2  2 15 15  7  7  8  8
[97]  8 14 14 14 14 16 16  7 14  8  7  7 11 11 15 15  7 17  7  8  8 18 18 18  9  9 19 19 19 19  8
20
[129] 20 14  7  7 13 13  4 16 16  7  7  7  7  2  2  2  7  7 18 18  2  2  3  3 20 20 16 16 16 16 16
16
[161] 19 19 19 15  8  8 13 13 13 13 13 13 16 16 16  5  5  5  4  4 15 15 15 15 20 20 20  5  5  5  5
5
[193]  5  4 19 17 17 17  4 14 14 14 14 13  7  7 10  4  4 17  5  5 17 17 17 17 17 17 17  7  7  7  7
7
[225]  7 17  5  5 14  2  2 18 18  6  6  6 16 16 16  8 16 16 12 12 16  2  3  3  2  5  5 10 10  3  7
7
[257]  7 18  6  6 19 19 18 18 18 18 10 10 11 11 11 11 15 16  3  3 10 10  2  6  7 16 16 16 16 16 15
7
[289] 16 16 16 16 16 16  5  5 19 19 19  3  8 19 10 10 10  7  7 11 11  5  5  3  3  4  4 19 12  3 16
16
[321]  5  4  2  2 16  8  8  5  4  7 16 20 10 12 14 14  6  6 10  9  1  1 21 21 21 10  5  7 10 17 14
5
[353]  5  5  5 17 17 15 15 12  1  1 17 19 18 18  4  4  9 18 17 21 21  6 20 20  5 19 19 20  4 20 20
21
[385] 21  2  7  7  5  5  2  2 17 10  4  4 12 12 18 18 19 20  5 15 19  5  5 18 18  4 16 19 19 10 18
18
[417]  3  5 10 21 15 18 17 17 16  5 18 11  5  5  4  8  1  1  1 16 12 18 11  4 10  3  3 12 19 10 21
```

```
21
[449]  7 18 19  7  8 19 19  1  1 16 10 19 19 19  1  1 21  1  2  5  2  7 13 16 21 21  7  5 21  7 16
17
[481] 15  5 17  1  7  1  7  5 16 16 17  7  7 17 16 18

# modularidade
modularity(cl)
0.8434784
```