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

15 de janeiro 2024
Versão 1.0.0

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, explorando modelos de geração de redes aleatórias e aplicando técnicas de deteção de comunidades em uma rede social real.

Na primeira parte, investigamos o modelo de geração de redes aleatórias conhecido como Passeio Aleatório. Este modelo é explorado em duas configurações distintas - uma clique com 10 nodos e outra com 20 nodos - com o intuito de caracterizar as redes geradas quanto à distância média, coeficiente de *clustering* e à presença de *hubs*. Essa exploração oferecerá uma compreensão mais profunda das propriedades estruturais emergentes nesses contextos.

A segunda parte deste estudo concentra-se na aplicação de métodos de deteção de comunidades na componente gigante de uma rede social. Utilizamos quatro abordagens distintas para identificar comunidades dentro dessa rede, avaliando o número de comunidades, suas dimensões e a qualidade das partições obtidas por cada método.

Ao longo deste trabalho, exploramos conceitos fundamentais da teoria de redes e aplicamos técnicas analíticas para compreender a estrutura e organização das redes estudadas. Os resultados obtidos proporcionarão uma visão abrangente das propriedades estruturais das redes e da eficácia dos métodos de deteção de comunidades empregados.

Q1 : Geração de redes aleatórias com o Passeio Aleatório (*Random Walk Model*)

O modelo Barabási-Albert ou BA é um modelo de ligação preferencial onde, a partir de uma configuração inicial, clique de pequena dimensão (rede ou subrede completa), se vão acrescentando nodos um a um, priviligiando-se as ligações a nodos com maior grau. Desta forma, é um modelo de crescimento preferencial usado em redes complexas como redes sociais, redes de computadores, entre outras áreas, em que os nós vão se adicionando gradualmente à rede e preferindo conectar aos nodos já existentes, com base na sua popularidade atual na rede. 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 e não tem em conta a existência de nodos adjacentes. Desta forma, não é propício à formação de triângulos, muito importante para a coesão da rede e presente em redes reais. Assim, para gerar as redes utilizou-se o Passeio Aleatório (*Random Walk Model*), de forma a aumentar o número de triângulos, através de um mecanismo que favorece a introdução de ligação entre nodos adjacentes.

Em seguida, irão ser geradas e analisadas 10 redes aleatórias de 200 nodos cada uma, a partir de uma clique com 10 nodos e 20 nodos como configuação inicial, respetivamente. O objetivo é analisar as propriedades e características dessas redes resultantes, comparando como as diferentes configurações iniciais podem afetar a estrutura e os padrões de conexão observados.

Em baixo, apresenta-se o pseudo-código para o algoritmo desenvolvido:

```
args:
  - g grafo
  - 0 < prob < 1
  - m > 1 int: número de ligações (3)
  - n int: número de nodos desejado (200)



\ - exceto
```

```
U - reunião




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 U novo_n.vizinhos))
8         if u() < p
9              g.link(rand_viz, novo_n)
10        else
11             g.link(novo_n, random(g.nodes \ (novo_n U rand_viz U novo_n.vizinhos))
```

1- Inicialização:

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

2 - Geração das Redes:

a) Repete-se até que o número de nós na rede seja igual a 200 (n):

b) Escolhe-se aleatoriamente um nó existente na rede (rand_n).

c) Adiciona-se um novo nó (novo_n) à rede.

d) Obtêm-se os vizinhos do nó selecionado aleatoriamente (vizs).

e) Cria-se uma conexão entre o novo nó e o nó selecionado aleatoriamente (rand_n).

f) Repete-se (m - 1) vezes, onde m é 3 neste caso:

g) Escolhe-se aleatoriamente um vizinho do nó selecionado (rand_viz) que não seja o novo nó ou um vizinho do novo nó.

h) Com uma probabilidade p (0,8 neste caso), conecta-se o novo nó ao vizinho selecionado (rand_viz), formando um triângulo.

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

h) Voltar a b) até té que o número de nós na rede seja igual a 200 (n).

Dada a probabilidade de aceitação ser elevada (0.8), espera-se que o coeficiente de *clustering* da rede seja significativo e, consequentemente, se formem muitos triângulos.

A clique inicial de 10 nodos é a seguinte:

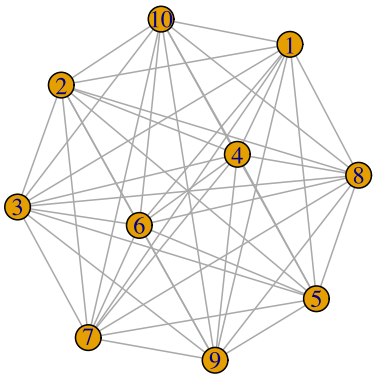


Figura 1: Clique inicial com 10 nodos

Para ficar mais claro o resultado da execução do algoritmo, apresentamos a rede após a adição de um nodo, na primeira iteração:

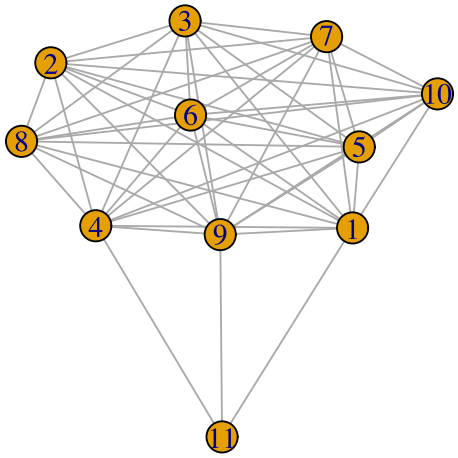


Figura 2: Rede após uma iteração

A clique inicial de 20 nodos é a seguinte:

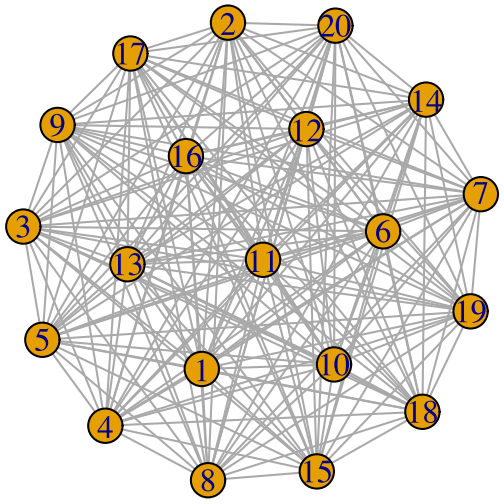


Figura 3: Clique inicial com 10 nodos

a) Clique com 10 nodos

Tabela 1: Tabela com os resulados obtidos a partir de uma clique com 10 nodos

Nº da rede gerada	**Distância média**	**Coeficiente de *Clustering***	**Heterogeneidade**	***Hubs*?**
1	3.221407	0.2644068	1.722520	**Sim**
2	3.319296	0.2754468	1.567850	**Sim**
3	3.324271	0.2769935	1.548549	**Sim**
4	3.257588	0.2588944	1.611739	**Sim**
5	3.204020	0.2605954	1.672285	**Sim**
6	3.163015	0.2518193	1.724899	**Sim**
7	3.442362	0.3015933	1.556481	**Sim**
8	3.292915	0.2680661	1.585829	**Sim**
9	3.233518	0.2756297	1.653249	**Sim**
10	3.178342	0.2454862	1.758741	**Sim**

Da tabela pode retirar-se que:

- **Distância média**: Os valores da distância média variam entre, aproximadamente, 3.16 e 3.44. Assim, sugere que, em média, a distância entre quaisquer dois nós 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.
- **Coeficiente de *clustering***: Os valores do coeficiente de *clustering* variam entre cerca de 0.245 e 0.302, 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.
- **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 nós 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.
- **Identificação de *hubs***: Todos os parâmetros de heteregoneidade são significativemente maior do que 1, pelo que há nós que se destacam com uma conectividade significativamente maior em relação aos demais, sendo suscetível a existência de *hubs*.

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 nós de alta conectividade, que podem ser de grande importância na estrutura e na dinâmica dessas redes.

b) Clique com 20 nodos

Tabela 2: Tabela com os resulados obtidos a partir de uma clique com 20 nodos

Nº da rede gerada	Distância média	Coeficiente de *Clustering*	Heterogeneidade	*Hubs*?
1	3.078744	0.4248927	2.148245	**Sim**
2	3.019095	0.4121870	2.227998	**Sim**
3	3.030704	0.4239161	2.158191	**Sim**
4	2.992915	0.4116804	2.231938	**Sim**
5	3.086734	0.4351405	2.120285	**Sim**
6	2.973467	0.4007937	2.264965	**Sim**
7	3.048090	0.4269252	2.139989	**Sim**
8	3.020302	0.4167045	2.203978	**Sim**
9	3.092261	0.4307954	2.125915	**Sim**
10	3.084020	0.4249649	2.143929	**Sim**

- **Distância Média**: Os valores da distância média variam aproximadamente entre 2.973 e 3.092. Isso indica que, em média, a distância entre quaisquer dois nós 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.
- **Coeficiente de *Clustering***: Os valores do coeficiente de clustering variam entre cerca de 0.401 e 0.436. 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.
- **Heterogeneidade**: A medida de heterogeneidade varia aproximadamente entre 2.121 e 2.265. Valores mais altos indicam uma distribuição mais desigual ou heterogênea dos graus dos nós na rede. Assim como na análise anterior, embora haja uma diferença, os valores são relativamente próximos entre as redes geradas.
- **Identificação de *Hubs***: Da mesma forma que na análise anterior, a coluna "Hubs?" indica que em todas as redes geradas (indicado por "Sim"), há nós 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 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 de uma clique de 20 nodos, comparativamente às redes geradas a partir de uma clique com 10 nodos:

O **coeficiente de *clustering***, que mede a tendência dos vizinhos de um nodo estarem interconectados, 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*.

Quanto à **heterogeneidade**, ela também foi maior para as redes geradas a partir de uma clique inicial de maior dimensão (20 nodos). 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.

Assim, 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.

Q2: Deteção de comunidades na componente gigante do Trabalho de Grupo I

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** da rede do **Trabalho de Grupo I**, cujos nodos representam os habitantes da zona residencial e cada ligação indica a existência de contacto social direto entre dois habitantes.

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 ao algoritmo de Girvan e Newmann, 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.

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.

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.

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. 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 de seguida:

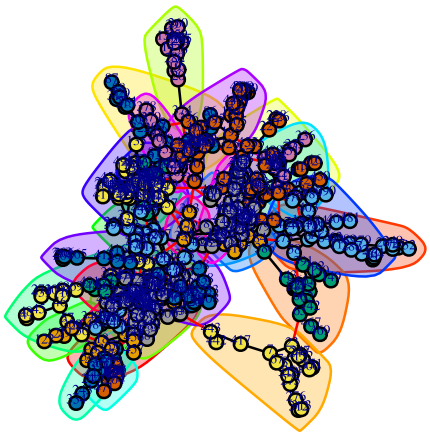


Figura 4: Representação visual das comunidades geradas pelo método de remoção de pontes

A tabela obtida com a frequência do tamanho das comunidades para este método foi a seguinte:

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

Assim, 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

Nesta método a representação visual obtida é a seguinte:

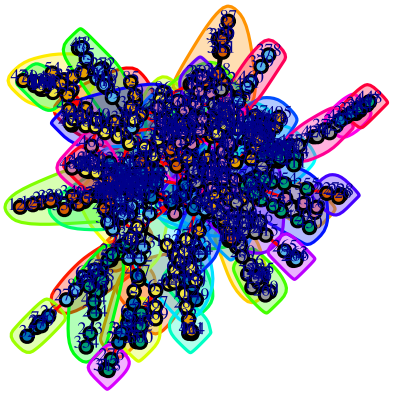


Figura 5: Representação visual das comunidades geradas pelo método de propagação de etiquetas

A tabela ocom a frequência do tamanho das comunidades para este método apresenta-se de seguida:

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 tamnhos 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 este terceiro método temos a seguinte representação visual:

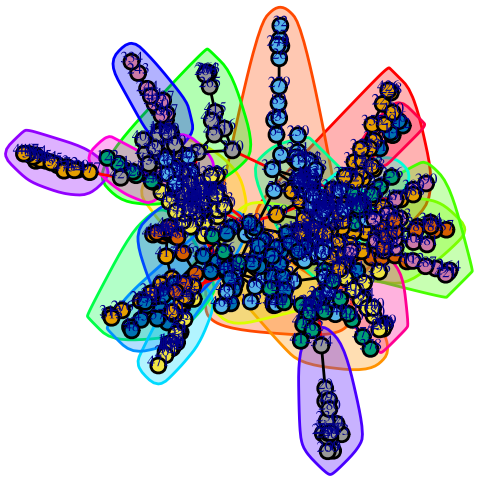


Figura 6: Representação visual das comunidades geradas pelo método *Fast Greedy*

A frequência obtida para o tamanho das comunidades foi a seguinte:

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

Finalmente, para o quarto método obtivemos a seguinte representação visual:

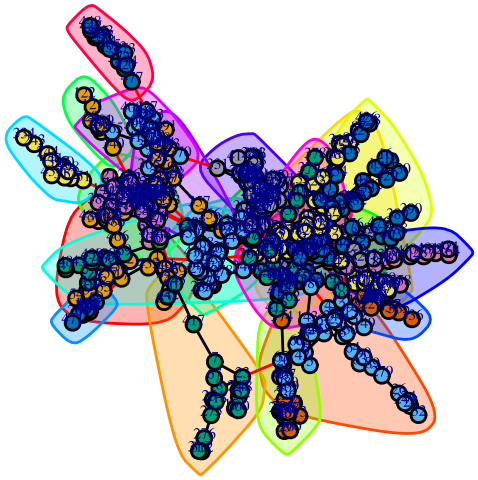


Figura 7: Representação visual das comunidades geradas pelo algoritmo de Louvain

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

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.

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.

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).

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* e a presença de *hubs*, revelando padrões consistentes e nuances interessantes em relação ao tamanho inicial da clique.

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. 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, mas também destacam a relevância e a diversidade de ferramentas analíticas disponíveis para investigar esses sistemas complexos e interativos.

Apêndices

Questão 1

Apêndice A1 - Importação das bibliotecas necessárias

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

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
# 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
}
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)
```

```r
# 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)


# 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)
  )
}

calculate_metrics_graphs(graphs)
```

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

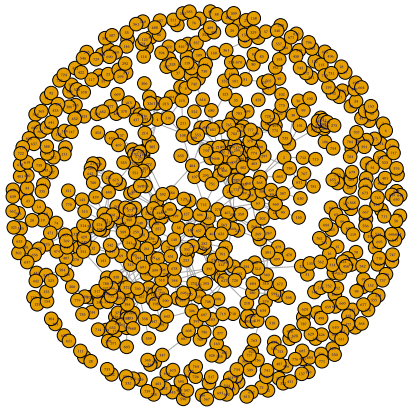
Apêndice C1 - Clique com 20 nodos

```
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)



# A tibble: 10 × 4
    Rede Distancia_Media Coeficiente_de_Cluste…¹ Heterogeneidade
   <int>           <dbl>                  <dbl>           <dbl>
 1     1            3.08                  0.425            2.15
 2     2            3.02                  0.412            2.23
 3     3            3.03                  0.424            2.16
 4     4            2.99                  0.412            2.23
 5     5            3.09                  0.435            2.12
 6     6            2.97                  0.401            2.26
 7     7            3.05                  0.427            2.14
 8     8            3.02                  0.417            2.20
 9     9            3.09                  0.431            2.13
10    10            3.08                  0.425            2.14
# ℹ abbreviated name: ¹Coeficiente_de_Clustering
```

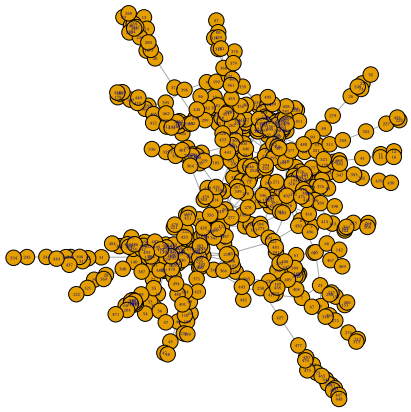
Apêndice A2 - Rede e componente gigante

```
#### 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)
```



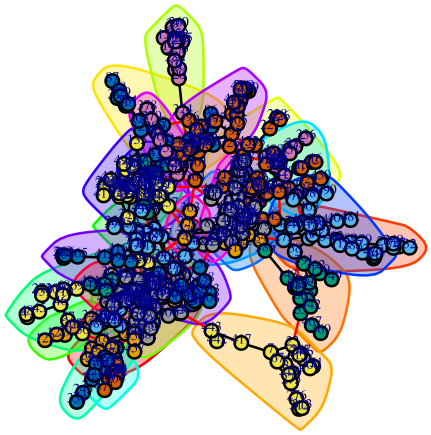
```
vcount(componente_gigante)
496
ecount(componente_gigante)
984
```

Questão 2

Apêndice B2 - 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)
```

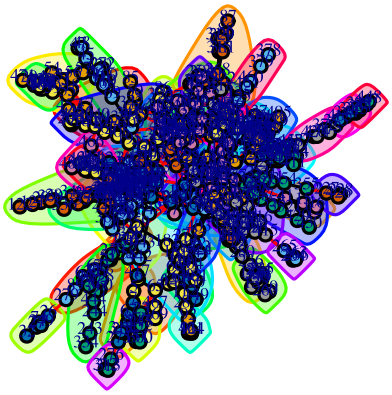


```
# numero de comunidades
length(crm)
27

# 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 C2 - 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)
```

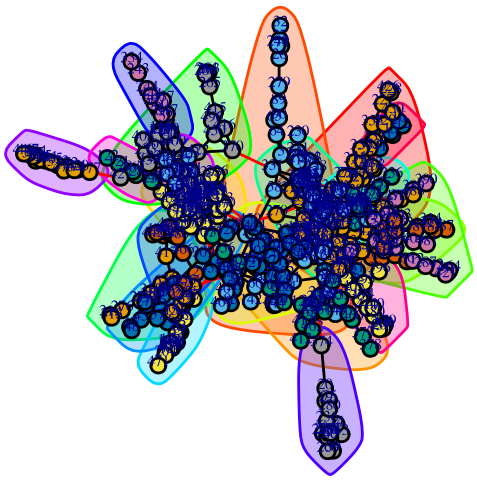


```
# numero de comunidades
length(cpe)
52

# 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
```

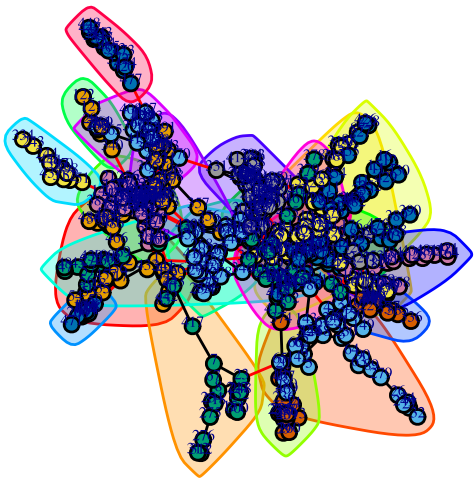
Apêndice D2 - Otimização de modularidade (método *Fast Greedy*)

```
## 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 E2 - 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
```