



Ministério da  
**Ciência, Tecnologia  
e Inovação**



sid.inpe.br/mtc-m19/2014/01.21.19.20-TDI

## METODOLOGIA PARA A EVOLUÇÃO DE COMUNIDADES EM REDES COMPLEXAS DINÂMICAS

Sandy Moreira Porto

Dissertação de Mestrado do Curso  
de Pós-Graduação em Computação  
Aplicada, orientada pelo Dr. Mar-  
cos Gonçalves Quiles, aprovada em  
25 de fevereiro de 2014.

URL do documento original:

<<http://urlib.net/8JMKD3MGP7W/3FK7CDP>>

INPE  
São José dos Campos  
2014

**PUBLICADO POR:**

Instituto Nacional de Pesquisas Espaciais - INPE

Gabinete do Diretor (GB)

Serviço de Informação e Documentação (SID)

Caixa Postal 515 - CEP 12.245-970

São José dos Campos - SP - Brasil

Tel.:(012) 3208-6923/6921

Fax: (012) 3208-6919

E-mail: pubtc@sid.inpe.br

**CONSELHO DE EDITORAÇÃO E PRESERVAÇÃO DA PRODUÇÃO INTELLECTUAL DO INPE (RE/DIR-204):****Presidente:**

Marciana Leite Ribeiro - Serviço de Informação e Documentação (SID)

**Membros:**

Dr. Antonio Fernando Bertachini de Almeida Prado - Coordenação Engenharia e Tecnologia Espacial (ETE)

Dr<sup>a</sup> Inez Staciarini Batista - Coordenação Ciências Espaciais e Atmosféricas (CEA)

Dr. Gerald Jean Francis Banon - Coordenação Observação da Terra (OBT)

Dr. Germano de Souza Kienbaum - Centro de Tecnologias Especiais (CTE)

Dr. Manoel Alonso Gan - Centro de Previsão de Tempo e Estudos Climáticos (CPT)

Dr<sup>a</sup> Maria do Carmo de Andrade Nono - Conselho de Pós-Graduação

Dr. Plínio Carlos Alvalá - Centro de Ciência do Sistema Terrestre (CST)

**BIBLIOTECA DIGITAL:**

Dr. Gerald Jean Francis Banon - Coordenação de Observação da Terra (OBT)

**REVISÃO E NORMALIZAÇÃO DOCUMENTÁRIA:**

Marciana Leite Ribeiro - Serviço de Informação e Documentação (SID)

Yolanda Ribeiro da Silva Souza - Serviço de Informação e Documentação (SID)

**EDITORAÇÃO ELETRÔNICA:**

Maria Tereza Smith de Brito - Serviço de Informação e Documentação (SID)

André Luis Dias Fernandes - Serviço de Informação e Documentação (SID)



Ministério da  
**Ciência, Tecnologia  
e Inovação**



sid.inpe.br/mtc-m19/2014/01.21.19.20-TDI

## METODOLOGIA PARA A EVOLUÇÃO DE COMUNIDADES EM REDES COMPLEXAS DINÂMICAS

Sandy Moreira Porto

Dissertação de Mestrado do Curso  
de Pós-Graduação em Computação  
Aplicada, orientada pelo Dr. Mar-  
cos Gonçalves Quiles, aprovada em  
25 de fevereiro de 2014.

URL do documento original:

<<http://urlib.net/8JMKD3MGP7W/3FK7CDP>>

INPE  
São José dos Campos  
2014

Dados Internacionais de Catalogação na Publicação (CIP)

---

Porto, Sandy Moreira.

P838m Metodologia para a evolução de comunidades em redes complexas dinâmicas / Sandy Moreira Porto. – São José dos Campos : INPE, 2014.

xxii + 86 p. ; (sid.inpe.br/mte-m19/2014/01.21.19.20-TDI)

Dissertação (Mestrado em Computação Aplicada) – Instituto Nacional de Pesquisas Espaciais, São José dos Campos, 2014.

Orientador : Dr. Marcos Gonçalves Quiles.

1. redes complexas. 2. detecção de comunidades. 3. geração de redes. I. Título.

CDU 519.179.2

---



Esta obra foi licenciada sob uma Licença [Creative Commons Atribuição-NãoComercial 3.0 Não Adaptada](https://creativecommons.org/licenses/by-nc/3.0/).

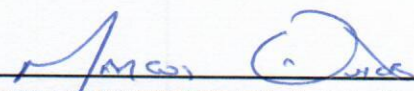
This work is licensed under a [Creative Commons Attribution-NonCommercial 3.0 Unported License](https://creativecommons.org/licenses/by-nc/3.0/).

Aprovado (a) pela Banca Examinadora  
em cumprimento ao requisito exigido para  
obtenção do Título de **Mestre** em  
**Computação Aplicada**

Dr. Rafael Duarte Coelho dos Santos

  
\_\_\_\_\_  
**Presidente / INPE / SJCampos - SP**

Dr. Marcos Gonçalves Quiles

  
\_\_\_\_\_  
**Orientador(a) / UNIFESP / São José dos Campos - SP**

Dr. Elbert Einstein Nehrer Macau

  
\_\_\_\_\_  
**Membro da Banca / INPE / São José dos Campos - SP**

Dr. Márcio Porto Basgalupp

  
\_\_\_\_\_  
**Convidado(a) / UNIFESP / São Paulo - SP**

*Este trabalho foi aprovado por:*

*maioria simples*

*unanimidade*

Aluno (a): **Sandy Moreira Porto**

**São José dos Campos, 25 de Fevereiro de 2014**



*A meus irmãos Matheus e Igor.*





## **AGRADECIMENTOS**

Aos meus pais, Ana Lúcia e Ciro, pelo apoio, amor e dedicação. Ao meu orientador, Dr. Quiles, pela paciência, incentivo e confiança.

Aos meus amigos, colegas de estudo e familiares, por todos os momentos de descontração e alegria .



## RESUMO

Em computação, o termo *benchmark* refere-se ao ato de submeter um programa de computador à uma série de testes com a finalidade de avaliar a sua performance relativa. No estudo das redes complexas não havia, até a escrita dessa dissertação, trabalho publicado que propusesse uma metodologia capaz de gerar redes dinâmicas com objetivos de *benchmark* para algoritmos de detecção de comunidades. Detecção de comunidades em redes dinâmicas exige que os algoritmos estejam preparados para lidar com as evoluções que essas podem sofrer. Sabe-se que essas modificações alteram a estrutura da rede, pois o número de vértices, arestas, a densidade e o número de comunidades, por exemplo, podem ser alterados ao longo do tempo. Neste trabalho, uma metodologia capaz de simular o comportamento e evolução das comunidades em redes dinâmicas é proposta. As redes geradas por essa metodologia podem ser utilizadas como *benchmark* para algoritmos de detecção de comunidades dinâmicos. Por fim, para avaliar a metodologia proposta, alguns algoritmos de detecção de comunidades são testados com as redes geradas.



# **METHODOLOGY FOR THE EVOLUTION OF COMUNITIES IN DYNAMIC COMPLEX NETWORKS**

## **ABSTRACT**

Benchmark, in computing, is the term related to the act of submitting a computer program to a series of tests in order to evaluate its relative performance. In the complex networks scenario, on the best of our knowledge, there was no published work that proposes a methodology for generating dynamics networks as a benchmark for community detection algorithms. Community detection in dynamic networks requires algorithms prepared to deal with the evolutions that communities may undergo. It is well known that these changes alter the network structure, for instance, the number of vertices, edges, the density, and the number of communities, may change over time. Here, we propose a new methodology for simulating the behavior and evolution of communities in dynamic networks. The networks generated by our method can be used as a benchmark for dynamic community detection algorithms. Finally, to evaluate the proposed methodology, some community detection algorithms are tested with these networks.



## LISTA DE FIGURAS

	<u>Pág.</u>
2.1 Desenho do rio e das setes pontes de Königsberg, e sua representação como um grafo. Fonte: Figura extraída de Albert e Barabási (2002). . . . .	5
2.2 Uma rede do tipo anel com $N = 12$ e $k = 6$ . (a) $p = 0$ , (b) $p = 0.15$ e (c) $p = 1$ . Fonte: Figura extraída de Viana (2007). . . . .	12
2.3 O nascimento de uma rede livre de escala. No tempo $t = 1$ , três nós iniciais conectados. No tempo $t = 2$ um quarto nó (branco) é adicionado e neste ponto esse nó tem igual probabilidade de se conectar com qualquer um dos três nós já presentes no sistema. No tempo $t = 3$ um quinto nó é adicionado mas dessa vez o nó tem probabilidade maior de se conectar com os vértices representados por círculos maiores, o tamanho do círculo no desenho é proporcional ao grau do nó. O processo continua até o tempo $t = 8$ quando atinge a configuração final desse exemplo. Fonte: Figura extraída de Barabási (2009). . . . .	13
3.1 Limite da Modularidade. A divisão natural em comunidades está representada pelos cliques $K_l$ , mas o maior valor da modularidade é obtido se as comunidades forem definidas por pares de cliques, como indicado pelas linhas pontilhadas. Fonte: Figura extraída de Fortunato (2010). . . . .	21
3.2 Rede Dinâmica com 4 <i>snapshots</i> , $s = \{1, 2, 3, 4\}$ , representados por matrizes $A_{ijs}$ que armazenam conexões <i>intra-snapshots</i> (linhas sólidas). As conexões <i>inter-snapshots</i> (linhas pontilhadas) são armazenadas por matrizes $B_{jrs}$ , que representam o acoplamento entre o vértice $j$ nos <i>snapshots</i> $r$ e $s$ . Fonte: Figura extraída de Mucha et al. (2010). . . . .	22
3.3 Ilustração de um dendrograma com o corte que representa o valor máximo da modularidade. Fonte: Figura adaptada de Raghavan et al. (2007). . . . .	30
3.4 Uma rede qualquer, com 20 vértices divididos em 3 módulos, representadas pelas cores vermelha, verde e azul. Abaixo, um possível caminho percorrido por um <i>random walker</i> com 50 passos. . . . .	31
3.5 Configuração dos códigos que descrevem o caminho descrito na Figura 3.4, utilizando os valores que estão na Tabela 3.1. . . . .	33

3.6	Passos do algoritmo apresentado em (QUILES et al., 2013). (a) Rede inicial. (b) Inserção das partículas em posições aleatórias. (c) Estado de equilíbrio e interpretação dos conglomerados em comunidades. (d) Nova estrutura de rede, com 200 arestas adicionadas entre as comunidades representadas pelas cores azul claro e azul escuro. (e)-(h) Movimentações das partículas. (i) Novo estado de equilíbrio. Fonte: Figura extraída de Quiles et al. (2013). . . . .	39
3.7	Exemplo de rede com comunidades de diferentes tamanhos. Fonte: Figura extraída de Danon et al. (2006). . . . .	43
3.8	Figura criada a partir da rede gerada utilizando o código fonte do <i>benchmark</i> LFR disponível em (FORTUNATO, 2013). . . . .	45
4.1	Possíveis transformações que uma comunidade pode sofrer numa rede complexa dinâmica. Fonte: Figura extraída de Palla et al. (2007). . . . .	48
4.2	Passos da função <i>born</i> . A comunidade sofrendo a transformação é a que está colorida de roxo. . . . .	49
4.3	Passos da função <i>extinction</i> . A comunidade sofrendo a transformação é a que está colorida de verde. . . . .	50
4.4	Passos da função <i>growth</i> . A comunidade sofrendo a transformação é a que está colorida de azul. . . . .	51
4.5	Passos da função <i>contraction</i> . A comunidade sofrendo a transformação é a que está colorida de azul. . . . .	52
4.6	Passos da função <i>merge</i> . As comunidades sofrendo a transformação são as que estão coloridas de azul. . . . .	53
4.7	Passos da função <i>split</i> . As comunidades sofrendo a transformação são as que estão coloridas de verde claro, verde, roxo, rosa, azul e azul claro. . . . .	54
5.1	Gráfico gerado com os resultados obtidos nos Experimentos Individuais da função <i>born</i> . . . . .	65
5.2	Gráfico gerado com os resultados obtidos nos Experimentos Individuais da função <i>extinction</i> . . . . .	65
5.3	Gráfico gerado com os resultados obtidos nos Experimentos Individuais da função <i>growth</i> . . . . .	66
5.4	Gráfico gerado com os resultados obtidos nos Experimentos Individuais da função <i>contraction</i> . . . . .	66
5.5	Gráfico gerado com os resultados obtidos nos Experimentos Individuais da função <i>merge</i> . . . . .	67
5.6	Gráfico gerado com os resultados obtidos nos Experimentos Individuais da função <i>split</i> . . . . .	67
5.7	Exemplo de uma amostragem de iterações. . . . .	68



5.8	Gráficos gerados com os resultados obtidos pelo Experimento Completo 1. . .	70
5.9	Gráficos gerados com os resultados obtidos pelo Experimento Completo 2. . .	71
5.10	Gráficos gerados com os resultados obtidos pelo Experimento Completo 3. . .	72
5.11	Gráficos gerados com os resultados obtidos pelo Experimento Completo 4. . .	73



## LISTA DE TABELAS

	<u>Pág.</u>
3.1 Códigos de cada um dos vértices e módulos para codificação de um nível e codificação de dois níveis. O número de visitas foi calculado a partir do caminho descrito na Figura 3.4. . . . . .	32
5.1 Parâmetros de entrada usados nos Experimentos Individuais. . . . .	64
5.2 Valores médios dos desempenhos dos algoritmos de acordo com os experimentos. . . . .	74
5.3 Porcentagem de vezes que um algoritmo atingiu o valor máximo nos experimentos. . . . .	74



## LISTA DE ALGORITMOS

4.1.1 Função Born . . . . .	56
4.2.1 Função Extinction . . . . .	57
4.3.1 Função Growth . . . . .	58
4.4.1 Função Contraction . . . . .	59
4.5.1 Função Merge . . . . .	60
4.6.1 Função Split . . . . .	61



## SUMÁRIO

	<u>Pág.</u>
<b>1 INTRODUÇÃO</b> . . . . .	<b>1</b>
1.1 Problemática . . . . .	2
1.2 Motivação . . . . .	2
1.3 Objetivos . . . . .	2
1.3.1 Objetivo Geral . . . . .	2
1.3.2 Objetivos Específicos . . . . .	2
1.4 Contribuição . . . . .	2
1.5 Estrutura do Trabalho . . . . .	3
<b>2 REDES COMPLEXAS</b> . . . . .	<b>5</b>
2.1 Propriedades Matemáticas do Grafos . . . . .	6
2.2 Propriedades das Redes Complexas . . . . .	8
2.2.1 Modelos de Redes Complexas . . . . .	10
2.3 Redes Complexas Dinâmicas . . . . .	13
<b>3 DETECÇÃO DE COMUNIDADES</b> . . . . .	<b>15</b>
3.1 Métodos de Validação . . . . .	16
3.1.1 Métodos com Critérios Internos . . . . .	17
3.1.1.1 Função <i>Performance P</i> . . . . .	18
3.1.1.2 Função Modularidade <i>Q</i> . . . . .	18
3.1.1.3 Limites da Modularidade <i>Q</i> . . . . .	20
3.1.1.4 Modularidade Dinâmica <i>Q<sub>D</sub></i> . . . . .	21
3.1.1.5 Função Modularidade com Penalidade de Divisão <i>Q<sub>pd</sub></i> . . . . .	22
3.1.1.6 Função Modularidade com Densidade <i>Q<sub>ds</sub></i> . . . . .	24
3.1.2 Métodos com Critérios Externos . . . . .	25
3.1.2.1 Índice <i>Rand</i> Corrigido . . . . .	25
3.1.2.2 Informação Mútua Normalizada . . . . .	27
3.1.3 Métodos com Critérios Relativos . . . . .	27
3.1.3.1 Índice Dunn . . . . .	28
3.2 Algoritmos de Detecção de Comunidades . . . . .	29
3.2.1 Algoritmo Fast Greedy . . . . .	29
3.2.2 Algoritmo Infomap . . . . .	30
3.2.3 Algoritmo Label Propagation . . . . .	34

3.2.4	Algoritmo Walktrap . . . . .	35
3.2.5	Detecção de Comunidades em Redes Complexas Dinâmicas . . . . .	36
3.3	Avaliação de Algoritmos . . . . .	38
3.3.1	<i>Benchmark</i> GN . . . . .	41
3.3.2	<i>Benchmark</i> LFR . . . . .	42
<b>4</b>	<b>METODOLOGIA . . . . .</b>	<b>47</b>
4.1	Função Born . . . . .	48
4.2	Função Extinction . . . . .	50
4.3	Função Growth . . . . .	50
4.4	Função Contraction . . . . .	52
4.5	Função Merge . . . . .	53
4.6	Função Split . . . . .	54
<b>5</b>	<b>EXPERIMENTOS E RESULTADOS . . . . .</b>	<b>63</b>
5.1	Experimentos Individuais . . . . .	63
5.2	Experimentos Completos . . . . .	68
5.2.1	Experimento Completo 1 . . . . .	69
5.2.2	Experimento Completo 2 . . . . .	69
5.2.3	Experimento Completo 3 . . . . .	70
5.2.4	Experimento Completo 4 . . . . .	71
5.3	Conclusões dos Experimentos . . . . .	72
<b>6</b>	<b>CONCLUSÕES E TRABALHOS FUTUROS . . . . .</b>	<b>77</b>
	<b>REFERÊNCIAS BIBLIOGRÁFICAS . . . . .</b>	<b>79</b>



## 1 INTRODUÇÃO

Uma rede é uma coleção de pontos ligados aos pares por linhas (NEWMAN, 2011), onde os pontos são chamados nós ou vértices, e as linhas chamadas arestas ou conexões. A origem desse campo de estudo data de 1736, quando Leonhard Euler publicou sua solução para o quebra-cabeça de Königsberg (COSTA et al., 2011). A partir do século XX redes têm sido usadas em diversos campos da ciência como uma forma de modelagem de sistemas complexos. Quando uma rede representa um sistema complexo, ela pode ser chamada de *Rede Complexa*. No estudo das redes complexas, uma rede é dita possuir uma estrutura de comunidade se é possível dividi-la em grupos de nós com conexões mais densas entre si do que com o resto da rede. Detectar tais comunidades, assim como sua evolução no tempo, é fundamental para o entendimento do sistema que ela representa (BANSAL et al., 2011).

O problema de detecção de comunidades envolve encontrar tais grupos mais conectados em uma dada rede e se tornou um problema algorítmico popular nos últimos anos. O termo *comunidade* tem sido bastante usado na literatura em diferentes contextos e conotações. Estudos sociais foram provavelmente os primeiros a utilizá-lo, onde a noção de comunidade era usada para denotar grupos de pessoas com interesses ou atividades em comum (PAPADOPOULOS et al., 2012). Ou seja, em redes complexas, as comunidades, também chamadas *módulos* ou *clusters*, são grupos de vértices que provavelmente compartilham propriedades em comum e/ou seguem regras similares dentro da rede (FORTUNATO, 2010).

No entanto, muito dos estudos na área de detecção de comunidades em redes complexas são baseados na premissa de que tais redes são estáticas, ou seja, redes onde os vértices e arestas permanecem fixos ao longo do tempo. E apesar de tais trabalhos terem evoluído e contribuído significativamente para o entendimento das redes complexas assim como sua estrutura de comunidades, eles não estão preparados para lidar com a maioria das redes reais, que têm com maior característica a dinamicidade (BANSAL et al., 2011). Dentro de uma rede, novas conexões estão sempre se formando, enquanto outras estão se dissolvendo, traçando novas topologias para a rede e certamente traçando novas estruturas de comunidades. Essas comunidades podem se dividir, se mesclar, crescer, diminuir, nascer e morrer. Exemplos de tais redes dinâmicas são a *Web*, onde novos *sites* e *links* se formam a cada minuto, e as redes sociais como Twitter e Facebook que recebem cada vez mais usuários, que formam novos laços de amizades frequentemente.

## 1.1 Problemática

Com o surgimento cada vez maior de algoritmos para redes dinâmicas, é importante que exista um cenário comum para que análises de desempenho e acurácia possam ser realizados. Tais cenários, denominados *benchmarks*, possibilitam avaliar o desempenho de um dado algoritmo em situações nas quais os resultados são conhecidos. Especificamente no contexto de detecção de comunidades, um *benchmark* pode ser definido como um conjunto de redes com estrutura de comunidades bem definidas.

## 1.2 Motivação

Até a escrita deste trabalho não havia na literatura uma metodologia capaz de simular o comportamento das comunidades em Redes Complexas Dinâmicas, com objetivos claros de *benchmark*. Também foi observado que não há trabalho publicado que compare o desempenho de algoritmos em Redes Dinâmicas.

## 1.3 Objetivos

### 1.3.1 Objetivo Geral

O objetivo geral deste trabalho é propor uma metodologia capaz de simular o comportamento e evolução de comunidades em Redes Complexas Dinâmicas.

### 1.3.2 Objetivos Específicos

- Criação de uma função para cada comportamento possível de uma comunidade em Redes Complexas Dinâmicas;
- Verificar se a metodologia consegue manter a estrutura em comunidades apesar das transformações sofridas na rede;
- Testar algoritmos de detecção de comunidades com a metodologia proposta;
- Analisar o desempenho dos algoritmos testados.

## 1.4 Contribuição

Apesar da grande contribuição e avanços que os estudos em Redes Complexas estáticas obtiveram, a maioria das redes do mundo real tem como característica a constante evolução no tempo. Essa evolução implica em mudanças inclusive na estrutura em comunidades da rede e essas mudanças podem ser simples, como o crescimento ou contração do tamanho de uma comunidade, ou mais complexas, como a junção de duas ou mais

comunidades numa única, ou ainda a divisão de uma comunidade em duas ou mais sub-comunidades. As mudanças ainda podem fazer com que uma comunidade desapareça da rede ou com que ocorra o surgimento de uma nova comunidade.

Sendo assim, a contribuição desse trabalho é um passo inicial para a área de *benchmark* em Redes Dinâmicas. A metodologia proposta simula as possíveis transformações que uma comunidade pode sofrer durante sua evolução no tempo. Sendo assim, pode ser usada para simular o comportamento de Redes Dinâmicas com estrutura em comunidades para comparação de desempenho de algoritmos de detecção.

## 1.5 Estrutura do Trabalho

No Capítulo 2, uma revisão bibliográfica em Redes Complexas será apresentada. Propriedades matemáticas dos grafos e das redes serão explicadas, assim como os modelos de redes existentes na literatura. Nesse mesmo capítulo, uma breve introdução as Redes Dinâmicas será apresentada. No Capítulo 3.2, o assunto tratado serão os algoritmos de detecção de comunidades. Antes, porém, as comunidades em redes serão introduzidas, assim como algumas medidas intrínsecas a essa característica. Posteriormente, algoritmos de detecção serão apresentados, os mesmos que serão utilizados no capítulo de experimentos. Por fim, técnicas de detecção em redes dinâmicas existentes e alguns algoritmos para detecção de comunidades em redes dinâmicas serão descritos.

No Capítulo 3.1, algumas métricas utilizadas para a validação de resultados serão revisadas. As técnicas serão divididas de acordo com os critérios utilizados na validação, que podem ser internos, externos ou relativos. No Capítulo 3.3, serão mostrados as técnicas de geração de *benchmark* para detecção de comunidades em Redes Complexas. Incluindo o *benchmark* GN, *benchmark* LFR, e suas variações, que são os mais famosos e utilizados na literatura. No Capítulo 4, a metodologia proposta neste trabalho será apresentada.

No Capítulo 5, alguns experimentos realizados com a metodologia são apresentados. Por fim, as conclusões finais e possíveis trabalhos futuros serão descritos no Capítulo 6.



## 2 REDES COMPLEXAS

A primeira vez que se tem conhecimento de um problema modelado em um grafo data de 1736 quando Leonhard Euler publicou um artigo demonstrando que não existia solução para o quebra-cabeça *Setes Pontes de Königsberg* (EULER, 1736; FORTUNATO, 2010; ROCHA, 2007; RODRIGUES, 2007). A Figura 2.1 mostra um esquema do quebra-cabeça, cujo objetivo era descobrir se era possível passar pelas duas ilhas e duas margens de um rio utilizando as setes pontes que as ligavam sem que nenhuma se repetisse durante o trajeto. Euler provou que tal caminho era impossível, pois existiam mais do que dois vértices (que representavam as porções de terra) com grau ímpar (número de arestas, nesse caso, pontes, conectadas aos vértices) e para ser possível realizar o que ficou conhecido como trajetória de Euler, o grafo deve conter no máximo dois vértices de grau ímpar (EULER, 1736). Para que um grafo seja considerado Euleriano, ele deve ter um circuito de Euler (o caminho deve terminar no vértice inicial), ou seja, todos os vértices devem possuir grau par (DIESTEL, 2005).

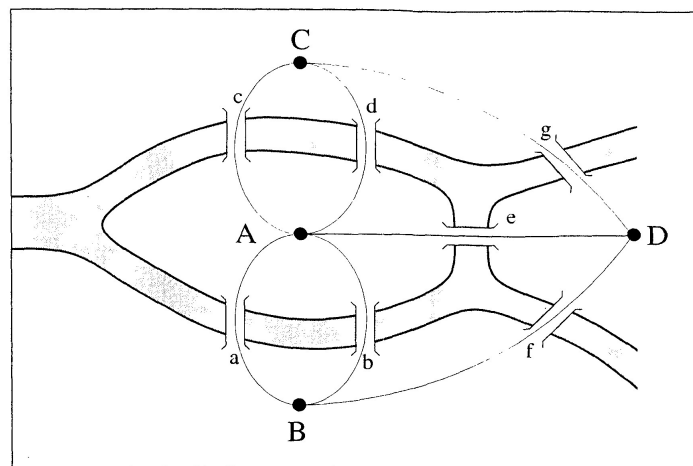


Figura 2.1 - Desenho do rio e das setes pontes de Königsberg, e sua representação como um grafo. Fonte: Figura extraída de Albert e Barabási (2002).

Após a publicação de Euler, muito se estudou e formulou sobre a Teoria dos Grafos, como ficou conhecido esse campo de estudo. Em 1998, Bollobás publicou *Modern Graph Theory* onde estão reunidas as principais propriedades matemáticas da Teoria dos Grafos (BOLLOBÁS, 1998). A área se tornou bastante difundida, principalmente pela capacidade que os grafos têm de representação de uma grande variedade de problemas. Nas próximas seções serão descritas algumas das propriedades matemáticas encontradas no livro de Bollobás e em outras referências (FORTUNATO, 2010; FEOFILOFF et al., 2011; DIESTEL,

2005; ROCHA, 2007; RODRIGUES, 2007), serão introduzidos o conceito de Redes Complexas, alguns modelos de redes encontrados na literatura e, por fim, uma introdução ao foco desse trabalho que são as redes complexas dinâmicas.

## 2.1 Propriedades Matemáticas do Grafos

Na Teoria dos Grafos, um *grafo* é a representação de um conjunto de objetos onde alguns pares de objetos são conectados por linhas. Esses objetos são representados por abstrações chamadas *vértices*, e *arestas* (TRUDEAU, 1993). Geralmente, um grafo é diagramado como um conjunto de pontos, que representam os vértices, ligados por linhas, que representam as arestas. As arestas podem ser direcionadas em alguns casos, indicando que há uma relação de um vértice para o outro, mas não necessariamente que essa relação é simétrica. Nesse caso as arestas podem ser chamadas *arcos*.

Matematicamente, um grafo  $G$  consiste em dois tipos de elementos, nomeados vértices e arestas. Cada aresta tem dois pontos finais no conjunto de vértices, e é dito *ligar* ou *conectar* esses dois pontos. Assim sendo, uma aresta pode ser definida como um conjunto de dois vértices, ou um par ordenado no caso direcionado. Os dois pontos de uma aresta são considerados *adjacentes* ou *vizinhos* um ao outro.

Um *vértice* é desenhado como um *nó* ou um *ponto*. O conjunto de vértices de  $G$  é geralmente denotado como  $V(G)$  ou simplesmente  $V$ . A *ordem* de um grafo é o tamanho do conjunto de vértices, ou seja,  $|V(G)|$ ,  $|V|$ , ou simplesmente,  $n$ .

Uma *aresta*, que é um conjunto de dois elementos, é desenhado como uma linha conectando dois vértices. Uma aresta que conecta dois vértices  $v$  e  $w$ , pode ser denotada como  $(v, w)$  ou  $vw$ . No caso de grafos direcionados  $vw \neq wv$ . O conjunto de vértices de um grafo  $G$  é geralmente denotado  $E(G)$  ou simplesmente  $E$ . O *tamanho* de um grafo é o tamanho do conjunto de arestas, ou seja,  $|E(G)|$ ,  $|E|$  ou simplesmente  $m$ . Um grafo é dito *ponderado* se existe um conjunto  $W$  onde valores, chamados *pesos*, são associados a cada uma das arestas do grafo.

O grau de um vértice  $v$ , denotado  $d(v)$  ou  $d_v$ , é o número de arestas que tem como ponto incidente este vértice. O grau mínimo de um grafo  $G$  é o valor  $\delta(G) = \min\{d(v) : v \in V\}$  e o grau máximo de  $G$  é o valor  $\Delta(G) = \max\{d(v) : v \in V\}$ . Um grafo é dito *regular* se todos os seus vértices têm o mesmo grau, ou seja,  $\delta(G) = \Delta(G)$ , e um grafo é dito *k-regular* se  $d(v) = k$  para todo vértice  $v \in V$ . Uma proposição útil na teoria dos grafos é que a soma dos graus dos vértices de um grafo é igual ao dobro de número de arestas, conforme ilustra a Equação 2.1.

$$\sum_v d(v) = 2|E| \quad (2.1)$$

Seja  $V^2$  o conjunto de todos os pares não-ordenados dos elementos de  $V$ , se  $|E| = |V^2|$ , então esse grafo é dito *completo*. Um grafo  $G'$  é dito *subgrafo* de  $G$  se  $V' \subset V$  e  $E' \subset E$ , e nesse caso,  $G' \subset G$ . O conjunto dos vértices adjacentes a  $v$  é denotado  $\Gamma(v)$  e é chamado *vizinhança* de  $v$ . O grau do vértice  $v$  é o tamanho do conjunto da vizinhança,  $d(v) = |\Gamma(v)|$ . No caso de grafos direcionados, ou *orientados*, o grau de um vértice  $v$  pode ser dividido em grau de entrada e grau de saída. O grau de entrada é o número de arestas que terminam em  $v$ , ou seja,  $d_{in}(v) = |\{wv : wv \in E\}|$ , e o grau de saída é o número de arestas que começam em  $v$ , ou seja,  $d_{out}(v) = |\{vw : vw \in E\}|$ .

Um *caminho* é uma sequência alternada entre vértices e arestas, começando e terminando com um vértice, onde cada vértice é incidente à aresta precedente e à aresta posterior na sequência. Um caminho também pode ser interpretado como um grafo  $C = (V, E)$  onde  $V = \{v_1, v_2, \dots, v_n\}$  e  $E = \{v_1v_2, v_2v_3, \dots, v_{n-1}v_n\}$ . Os vértices  $v_1$  e  $v_n$  são os *extremos* de  $C$ , e caso,  $v_1 = v_n$  e  $n \geq 3$ , então  $C$  é chamado *ciclo* ou *circuito*. Um grafo *conexo* é aquele em que há um caminho entre qualquer par de vértices, e se não for o caso, ele é chamado *desconexo*. Um grafo que não contém um ciclo é chamado *floresta*, e caso essa floresta seja um grafo conexo, então é chamado *árvore*.

Árvores são um grupo especial dos grafos, que têm grandes aplicações na computação. Numa árvore só existe um caminho ligando qualquer par de vértices. Se qualquer aresta for removida numa árvore, ela se tornará desconexa e se uma aresta for adicionada a uma árvore, formará um ciclo. Todo grafo conexo contém uma *árvore geradora* que é uma árvore que contém todos os vértices do grafo, e contém o número mínimo de arestas sem que exista um vértice desconectado, e o número máximo de arestas sem que se forme um ciclo. Em grafos ponderados é possível definir árvores geradoras *mínimas* e *máximas*, onde a soma dos pesos das arestas é mínimo e máximo, respectivamente.

Uma *matriz de adjacência*  $A$  é uma matriz  $n \times n$  em que cada elemento  $a_{ij}$  será igual a 1, caso exista ligação entre os vértices  $v_i$  e  $v_j$ , e será igual a 0 caso contrário. No caso de grafos não direcionados, a matriz de adjacência será sempre simétrica, e assimétrica no caso de grafos direcionados. Se o grafo for ponderado, a matriz pode ser estendida para armazenar também os pesos da aresta, ou seja, o elemento  $a_{ij}$  receberá o peso da aresta entre os vértices  $v_i$  e  $v_j$ , ou 0 caso não exista ligação. O grau de um vértice  $v_i$  pode ser calculado somando-se os valores da linha  $i$  da matriz.

## 2.2 Propriedades das Redes Complexas

A partir do sec. XX, grafos têm sido utilizados em diversas áreas para a representação de sistemas. Muitas dessas modelagens envolvem sistemas complexos, com milhões ou mesmo bilhões de vértices conectados entre si, e que às vezes seguem regras invisíveis para um observador externo (FORTUNATO, 2010; NEWMAN, 2003), quando esse é o caso, podemos dizer que esse grafo se trata de uma *Rede Complexa*.

Redes complexas não são apenas um conjunto de vértices e arestas. Cada vértice e aresta podem representar objetos diferentes, ou ainda podem existir outras informações associadas a essas estruturas (NEWMAN, 2003). Por exemplo, numa rede social, se os vértices correspondem as pessoas, estes podem ter informações associadas a eles, como por exemplo, gênero, idade, e se as arestas correspondem aos relacionamentos entre essas pessoas, estas podem ter informações como a característica do relacionamento (amoroso, familiar, amizade ou trabalho) ou então a intensidade do relacionamento, entre outras coisas. Outra característica importante das redes complexas é o fato de que elas não são estáticas e estão sempre evoluindo no tempo, modificando sua estrutura (RODRIGUES, 2007).

No estudo das Redes Complexas existem medidas e propriedades que podem ajudar a classificar e caracterizar o formato geral da rede. Essas medidas podem ajudar a dizer qual o comportamento dos vértices na rede, se existe ou não uma hierarquia entre eles, entre outras coisas. Nesta seção serão apresentadas algumas dessas medidas.

A primeira medida relaciona-se à *conectividade*: para cada vértice  $v_i$  um valor  $k_i$  pode ser determinado. Esse valor pode ser considerado sinônimo do grau do vértice caso a estrutura não tenha múltiplas arestas entre um mesmo par de vértices. Utilizando a matriz de adjacência  $A$ , onde  $a_{ij}$  é a  $i$ -ésima linha da  $j$ -ésima coluna de  $A$ , de um grafo qualquer  $G$ , o valor de  $k_i$  pode ser calculado pela seguinte equação:

$$k_i = \sum_{j=1}^n a_{ij} \quad (2.2)$$

A conectividade pode ser considerada a característica mais fundamental de um vértice. A partir da conectividade de cada vértice, a *conectividade média*  $\langle k \rangle$  do grafo pode ser calculada. Essa medida, apesar de simples, pode dizer muito sobre a rede estudada. A Equação 2.3 mostra como a conectividade média pode ser obtida em função dos valores de  $k_i$ :



$$\langle k \rangle = \frac{1}{n} \sum_{i=1}^n k_i \quad (2.3)$$

Outra medida importante relacionada à conectividade dos vértices é a *distribuição da conectividade*  $P(k)$ , que se refere à probabilidade de que um vértice escolhido aleatoriamente tenha grau ou conectividade  $k$ . Essa medida é utilizada quando tenta se determinar a qual modelo a rede tratada pertence. Alguns modelos de redes serão discutidos na Seção 2.2.1.

Para cada vértice  $v_i$  na rede, também pode ser determinado o *coeficiente de clusterização*  $cc_i$ . Essa medida indica a quantidade de ligações entre seus vizinhos  $e_i$  em relação ao número total de possíveis ligações, ou seja,  $k_i(k_i - 1)$ . O valor do coeficiente de clusterização pode ser obtido a partir da Equação 2.4:

$$cc_i = \frac{2e_i}{k_i(k_i - 1)} = \frac{\sum_{j=1}^n \sum_{l=1}^n a_{ij} a_{jl} a_{li}}{k_i(k_i - 1)} \quad (2.4)$$

Outras medidas que podem ser obtidas para caracterizar uma rede estão relacionadas com a distância entre os vértices. O *comprimento do caminho* que conecta dois vértices é obtido pela quantidade de arestas ao longo deste caminho. Já o comprimento do *menor caminho* entre dois vértices  $v_i$  e  $v_j$ , é calculado a partir de todos os caminhos existentes entre os dois vértices, e aquele(s) que tiver(em) tamanho mínimo representa(m) o menor caminho,  $d_{ij}$ . Uma matriz  $D$  pode ser definida onde cada elemento  $d_{ij}$  representa o valor do menor caminho entre os vértices  $v_i$  e  $v_j$ . A média dos valores da matriz  $D$  exprime o *caminho característico da rede*, ou ainda o *menor caminho médio*, e pode ser calculado segundo a Equação 2.5.

$$\langle d \rangle = \frac{1}{n(n-1)} \sum_{i=1}^n \sum_{j=1}^n d_{ij}, \text{ com } i \neq j \quad (2.5)$$

Medidas de centralidade servem para quantificar a importância de um vértice ou aresta no que se diz respeito à comunicação na rede. *Betweenness centrality* (FREEMAN, 1977) é uma medida útil baseada no fato de que um ponto em uma rede de comunicação é central se ele faz parte dos menores caminhos entre pares de outros pontos. A Equação 2.6 mostra como o *betweenness* de um ponto  $v$  na rede pode ser calculado.

$$b(v) = \sum_{s \neq v \neq t} \frac{\sigma_{st}(v)}{\sigma_{st}} \quad (2.6)$$

Em que  $\sigma_{st}$  é o número total de menores caminhos entre os pontos  $s$  e  $t$ , e  $\sigma_{st}(v)$  é o número de menores caminhos que passam por  $v$ . Portanto, se o valor de  $b(v)$  for alto, o ponto  $v$ , que no caso de um grafo pode tanto ser uma aresta como um vértice, exerce uma posição central na rede, e isso significa que esse ponto tem grande influência na comunicação da rede. Para mensurar o quão alto seria a influência desse ponto na rede, pode-se normalizar a função  $b(v)$ , como demonstrado na Equação 2.7.

$$normal(b(v)) = \frac{b(v) - \min(b)}{\max(b) - \min(b)} \quad (2.7)$$

E assim, quanto mais o valor normalizado de  $b(v)$  for próximo de 1, mais importante o ponto  $v$  será na rede. Na próxima seção, as medidas descritas serão utilizadas para caracterizar alguns dos modelos mais famosos de redes complexas. Às vezes, é possível determinar qual é o modelo da rede somente obtendo-se esses valores característicos, por isso, é importante definir essas medidas para as redes estudadas.

### 2.2.1 Modelos de Redes Complexas

Ao longo da segunda metade do século XX, modelos para a modelagem de sistemas reais em grafos, ou seja, redes complexas, foram introduzidas na literatura. O primeiro modelo foi sugerido por Erdos e Rényi em 1959 (ERDOS; RÉNYI, 1959; ERDOS; RÉNYI, 1960), e que ficou conhecido por *grafos aleatórios* ou modelo *ER*. Nesse modelo existem dois parâmetros, o número total de vértices  $n$  e uma probabilidade  $p$ . Cada par de vértices é conectado de acordo com essa probabilidade  $p$  independente dos outros vértices (COSTA et al., 2007; FORTUNATO, 2010). O número esperado de arestas para esse modelo é de  $pn(n-1)/2$  e a conectividade média  $\langle k \rangle = p(n-1)$ . A distribuição  $P(k)$  desse tipo de modelo segue uma distribuição de Poisson, que pode ser representada por  $P(k) = \langle k \rangle^k e^{-\langle k \rangle} / k!$  (NEWMAN, 2003), em que todos os vértices possuem grau próximo de  $\langle k \rangle$ . O menor caminho médio nesse tipo de rede tende a ser bem pequeno crescendo proporcionalmente ao logaritmo de  $n$  (RODRIGUES, 2007; VIANA, 2007). Quando a probabilidade de formação de ligações for zero,  $p = 0$ , a rede será completamente desconectada. Se  $p = 1$ , a rede será um grafo completo e, portanto, o valor médio do coeficiente de clusterização será máximo,  $\langle cc \rangle = 1$  (ROCHA, 2007).

Uma rede pode ter como característica um *menor caminho médio* bem pequeno. Esse

efeito é conhecido como *small-world*, introduzido em 1967 por Stanley Milgram (MILGRAM, 1967). Ele conduziu um experimento onde ele enviou centenas de cartas para pessoas escolhidas aleatoriamente nos EUA, e na carta, ele perguntava se a pessoa conhecia certo destinatário final pelo primeiro nome. Se sim, este deveria enviar a carta para o destinatário final, se não, a pessoa deveria mandar a carta para algum conhecido que tinha mais chances de conhecer o destinatário. Antes de enviar a carta, a pessoa deveria assinar seu nome na mesma. Ao final do experimento, o destinatário final deveria enviar a carta de volta a Milgram. O resultado do experimento foi que Milgram percebeu que as cartas eram em média assinadas por somente 6 pessoas antes de serem enviadas de volta, ou seja, existiam em média apenas 6 graus de separação entre quaisquer duas pessoas escolhidas aleatoriamente nos EUA (ALBERT; BARABÁSI, 2002).

Em 1998, Wattz e Strogatz mostraram que era possível redes complexas terem coeficiente de clusterização alto ao mesmo tempo em que mantinham o efeito *small-world*. Ou seja, eles observaram que a presença de ciclos de ordem três (triângulos) em redes reais eram muito maiores do que no modelo aleatório proposto por Erdos e Rényi (WATTS; STROGATZ, 1998). Esse fato mostrou que as redes reais, na verdade, não são completamente aleatórias e que existe algum tipo de lei de formação por trás da construção destas (RODRIGUES, 2007). Esse modelo ficou conhecido como *small-world de Watts-Strogatz*, ou simplesmente modelo WS. Para obtê-lo deve-se: (1) começar com uma rede regular<sup>1</sup> de tamanho  $N$  com  $k$  vizinhos; (2) cada aresta deve ser redirecionada para qualquer outro vértice na rede seguindo uma probabilidade  $p$  mas obedecendo duas restrições, nenhum vértice pode se ligar com ele mesmo, e não pode haver mais de uma conexão entre um par qualquer de vértices da rede (WATTS; STROGATZ, 1998; BARABÁSI et al., 1999). A Figura 2.2 mostra alguns exemplos do modelo. Quando  $p = 0$  a rede se comporta como uma rede regular e o menor caminho médio cresce linearmente com  $N$ . Quando  $p = 1$  o sistema se torna um grafo randômico e o menor caminho médio cresce proporcional ao logaritmo de  $N$  (modelo Erdos e Renyi).

Tanto no modelo proposto por Erdos e Renyi, como no proposto por Wattz e Strogatz, o número de vértices é fixo, além do que a probabilidade de quaisquer dois vértices se conectarem é randômica e uniforme. Até 1999, as redes eram descritas utilizando um desses dois modelos, e na falta de dados ou mesmo *hardware* capaz de tal processamento, eles não puderam ser testados de fato no mundo real (BARABÁSI et al., 1999) (BARABÁSI; ALBERT, 1999). A partir de 1999, cada vez mais dados e capacidade de processamento

---

<sup>1</sup>Redes Regulares são um tipo especial de modelo onde cada vértice só se conecta com  $k$  vizinhos mais próximos. É um modelo muito usado na representação de sistemas geográficos, onde só existe ligação entre vértices que são vizinhos geograficamente.

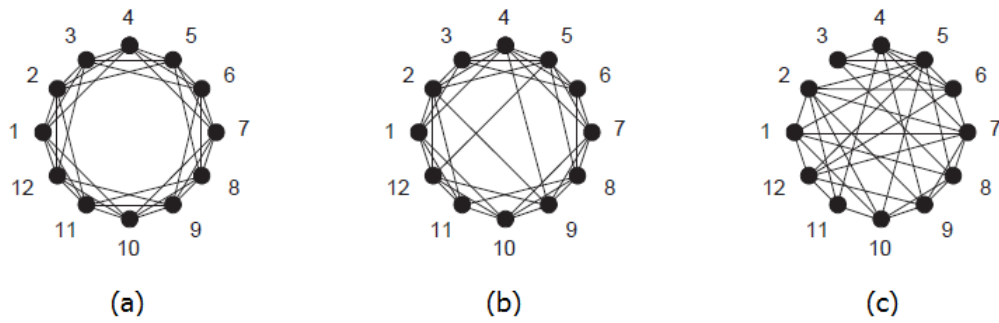


Figura 2.2 - Uma rede do tipo anel com  $N = 12$  e  $k = 6$ . (a)  $p = 0$ , (b)  $p = 0.15$  e (c)  $p = 1$ . Fonte: Figura extraída de Viana (2007).

foram incorporados nos estudos, e Barabási e Albert (BA) (BARABÁSI; ALBERT, 1999) demonstraram que a probabilidade  $P(k)$  que um vértice na rede esteja conectado a outros  $k$  vértices segue uma *lei de potência*, descrita como  $P(k) \sim k^{-\gamma}$ . Barabási e Albert argumentaram que existem dois aspectos genéricos nas redes que eles observaram que não estão incorporadas aos dois modelos apresentados anteriormente. Primeiro, a maioria das tais redes são *abertas*, ou seja, nós podem ser adicionados (e também retirados), ao contrário, como já foi citado, dos modelos ER e WS, onde o número de vértices é fixo. Segundo, que as tais redes exibem um comportamento chamado *ligação preferencial*, que significa que um novo vértice ao ser adicionado a uma rede já existente tem maior probabilidade de ser ligado a um vértice que já tem um grande número de conexões.

O modelo é baseado em dois passos, como descrito em (BARABÁSI et al., 2000):

- a) *Crescimento*: Iniciando com um pequeno número  $N_0$  de vértices, a cada passo adicionar um novo vértice com  $m (\leq N_0)$  arestas, que serão conectadas aos vértices que já estão presentes no sistema.
- b) *Ligação Preferencial*: Para decidir com quais vértices da rede o novo vértice irá se conectar, assume-se que a probabilidade  $\Pi(k_i)$  de que o novo vértice se conecte ao vértice  $i$  depende da conectividade  $k_i$  deste vértice, tal que:

$$\Pi(k_i) = k_i / \sum_j k_j \quad (2.8)$$

Nas redes geradas pelo modelo BA, ou também chamados de modelos livre de escala, os vértices mais conectados, tendem a receber mais conexões. Após algum tempo, é possível notar a existências de vértices centrais na rede, chamados *hubs*. A Figura 2.3 mostra

Modelo Livre de Escala

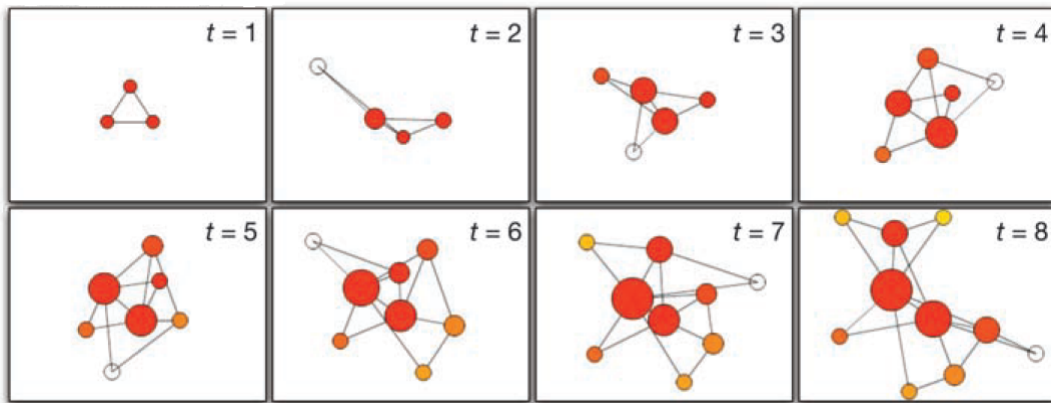


Figura 2.3 - O nascimento de uma rede livre de escala. No tempo  $t = 1$ , três nós iniciais conectados. No tempo  $t = 2$  um quarto nó (branco) é adicionado e neste ponto esse nó tem igual probabilidade de se conectar com qualquer um dos três nós já presentes no sistema. No tempo  $t = 3$  um quinto nó é adicionado mas dessa vez o nó tem probabilidade maior de se conectar com os vértices representados por círculos maiores, o tamanho do círculo no desenho é proporcional ao grau do nó. O processo continua até o tempo  $t = 8$  quando atinge a configuração final desse exemplo. Fonte: Figura extraída de Barabási (2009).

ao longo da evolução da rede o surgimento desses *hubs*. No tempo  $t = 8$  percebe-se a existência de círculos mais bem relacionados que os restantes, representados na figura por círculos maiores pois o tamanho do círculo é proporcional ao grau do vértice. Graças ao *crescimento* e à *ligação preferencial* um processo que é conhecido na literatura como *o rico fica mais rico* ou no termo em inglês *rich-get-richer*, é observado. Isso significa que os vértices mais bem conectados adquirem ainda mais conexões do que aqueles que são pouco conectados, levando a concentração das conexões em apenas alguns nós, os *hubs* (BARABÁSI, 2009).

### 2.3 Redes Complexas Dinâmicas

Muitos dos estudos clássicos na teoria das redes complexas são baseados em grafos estáticos, que são grafos que não mudam ao longo do tempo. Hoje em dia, graças aos avanços tecnológicos, existe a possibilidade de se estudar grandes redes sociais, biológicas ou mesmo a Web com informações temporais precisas no que diz respeito à aparição, duração e frequência de objetos na rede (vértices e arestas) (NICOSIA et al., 2012).

Redes complexas dinâmicas são redes que sofrem mudanças ao longo do tempo. A partir de uma rede inicial, vértices e arestas podem ser adicionadas e novas conexões entre vértices já existentes podem aparecer. Assim como, vértices podem morrer, conexões podem

ser desfeitas, um grafo conectado pode se tornar desconectado, e qualquer consequência derivada de qualquer ação sofrida pela rede é possível. Interações sociais e atividades humanas são intermitentes e conforme o tempo avança é inevitável que uma rede que represente um fenômeno real não sofra alterações estruturais (NICOSIA et al., 2012).

A evolução da rede, quando considerada, é normalmente estudada criando-se uma série de grafos estáticos, cada um desses grafos contendo toda a transformação ocorrida na rede durante um certo período de tempo. Ou seja, agrega-se várias mudanças ocorridas na rede durante um período de tempo, e o “resultado” das mudanças é exibido num grafo estático, sem informações adicionais sobre quais estruturas tiveram transformações. Mas em (MAITY et al., 2012) é afirmado que uma análise estática desses grafos agregados não é capaz de capturar o real comportamento da rede e as correlações que ocorrem ao longo do tempo.

No próximo capítulo comunidades em Redes Complexa serão tratadas. O que são, como se formam e como podem ser detectadas serão os assuntos discutidos. Além disso, alguns algoritmos da literatura que serão usados para experimentos neste trabalho serão explicados e, por fim, algumas técnicas de detecção de comunidades em Redes Complexas Dinâmicas serão apresentadas.

### 3 DETECÇÃO DE COMUNIDADES

Muitos sistemas complexos podem ser representados por redes, ou grafos, onde as partes elementares do sistema são substituídas por vértices, e suas interações são representadas por arestas. Sistemas complexos são normalmente organizados em compartimentos que têm suas próprias regras ou funções. Na representação por redes, esses compartimentos aparecem como grupos de vértices com uma alta densidade de conexão entre eles, enquanto conexões entre esses compartimentos são relativamente esparsas. Esses subgrafos são chamados comunidades, ou módulos, e ocorre na maioria dos sistemas (LANCICHINETTI; FORTUNATO, 2009b). Estrutura em comunidades parece ser comum a vários sistemas, ou redes, e a habilidade de encontrar e analisar tais comunidades podem prover valorosa ajuda para entender e visualizar a estrutura global do sistema (NEWMAN; GIRVAN, 2004).

Detecção de comunidades também é importante na classificação dos vértices. Quando os módulos e suas fronteiras são identificados, vértices com uma posição central, ou seja, que possuem muitas arestas ligando-o à própria comunidade, têm papel fundamental no controle e estabilidade daquele grupo. Já se o vértice estiver numa posição de fronteira entre os módulos, ele representa um ponto de mediação entre dois grupos, e será o elemento que irá liderar as comunicações entre eles (FORTUNATO, 2010).

Uma medida que pode ser muito útil para a caracterização dos vértices na estrutura de comunidades é a *integração* (ou no termo em inglês *embeddedness*) que significa o quão bem o elemento está relacionado ou integrado à comunidade que pertence. Ou seja, quantos dos vizinhos de um elemento pertencem à mesma comunidade que ele. O *grau interno*  $k_{int}$  de um nó é o número de conexões daquele nó que pertencem à mesma comunidade, e em oposição, o *grau externo*  $k_{ext}$  corresponde às conexões com vértices de outras comunidades (ORMAN et al., 2012). A *integração*  $e$  de um vértice pode ser definida como a razão entre seu grau interno e seu grau total, conforme a Equação 3.1.

$$e = k_{int}/k \quad (3.1)$$

Quando um vértice tem valor de integração 1, isso significa que todos os seus vizinhos estão em sua própria comunidade. O valor da integração igual a 0, só é possível se o vértice em questão fosse uma comunidade isolada. Em redes do mundo real, a maioria dos vértices têm grau total baixo, e um valor de integração alto (ORMAN et al., 2012).

A *densidade*  $\rho$  de uma comunidade  $C$  é definida como a razão entre o número de arestas

que existem dentro de uma comunidade, representada por  $l_C$ , e o número total de arestas que podem existir dentro da comunidade, ou seja, o número de arestas caso todos os vértices da comunidade fossem ligados entre si. Sendo  $n_C$  o número de vértices dentro da comunidade  $C$ , a equação da *densidade* é:

$$\rho = \frac{l_C}{\frac{n_C(n_C-1)}{2}} = \frac{2l_C}{n_C(n_C-1)} \quad (3.2)$$

Por definição, a densidade de uma comunidade deve ser maior que a densidade total do grafo. Outra medida que remete ao papel de um vértice dentro da rede é a *dominância*. Ela mede a existência de um *hub* dentro da comunidade. *Hubs* são nós conectados à grande maioria dos vértices da comunidade. A presença de um *hub* numa comunidade  $C$  pode ser avaliado usando a seguinte equação:

$$h(C) = \max_C(k_{int})/(n_C - 1) \quad (3.3)$$

O numerador é o grau interno mais alto encontrado dentro da comunidade  $C$ , e o denominador é o grau máximo interno possível, caso de um vértice conectado a todos os vértices da comunidade. O valor de  $h(C) = 1$  significa que existe um *hub* conectado com todos os vértices existentes na comunidade.

Em geral encontrar a solução exata da divisão de uma rede em comunidades é considerado um problema *NP-completo* (FORTUNATO, 2010). Em geral, pouco se sabe sobre a estrutura de comunidades de um grafo. É incomum saber *a priori* em quantas comunidades uma rede pode ser dividida, ou qualquer outra indicação de qual vértice pertence a qual comunidade. A tarefa de detectar comunidades em um grafo não é trivial, e pode ser abordada de diversas maneiras.

A seguir serão apresentados alguns métodos que são utilizados para validar os resultados obtidos por algoritmos de agrupamento. Dentre eles serão introduzidas a medida de modularidade e a informação mútua normalizada que serão usadas em outros capítulos e seções deste trabalho.

### 3.1 Métodos de Validação

Algoritmos de detecção de comunidades podem ser encarados como algoritmos de agrupamento. Afinal, os dados estão sendo separados em grupos de acordo com suas similaridades. E como qualquer algoritmo de agrupamento, este deve passar por um processo



de validação de resultados. Esse processo deve, de forma objetiva, determinar se, no caso das redes, as comunidades são significativas, ou seja, se a solução é representativa para o conjunto de dados analisado. Uma estrutura de agrupamento é válida se não ocorreu por acaso ou se é rara em algum sentido (FACELI, 2007).

A validação do resultado de um agrupamento, em geral, é feita com base em índices estatísticos que julgam, de maneira qualitativa, o mérito das estruturas encontradas (JAIN; DUBES., 1988). Existem três tipos de critérios para investigar a validade de um agrupamento: (FACELI et al., 2005).

- **Crítérios internos:** Medem a qualidade de um agrupamento com base apenas nos dados originais.
- **Crítérios externos:** Avaliam um agrupamento de acordo com uma estrutura pré-especificada. Essa estrutura pré-especificada pode ser uma partição que se sabe previamente existir nos dados.
- **Crítérios relativos:** Tendo vários resultados de agrupamento, o índice irá tentar decidir qual deles é o melhor. Pode ser usado para decidir entre vários tipos de algoritmos ou para diferentes conjuntos de parâmetros de entrada.

A forma mais comum de aplicação de um índice com um critério relativo consiste no cálculo do seu valor para vários agrupamentos que estão sendo comparados, obtendo-se uma sequência de valores. O melhor resultado de agrupamento é determinado pelo valor que se destaca nessa sequência (FACELI, 2007).

Os critérios externos e internos são baseados em testes estatísticos. Seu objetivo é medir o quanto o resultado obtido confirma uma hipótese pré-especificada. Neste caso, são utilizadas testes de hipótese para determinar se uma estrutura obtida é apropriada para os dados (JAIN; DUBES., 1988).

Nas próximas seções serão apresentados alguns índices, métodos ou medidas de validação de agrupamento, divididos entre os possíveis critérios. Primeiramente, aqueles que podem ser considerados como internos, depois os externos e por fim, os relativos.

### 3.1.1 Métodos com Critérios Internos

Os métodos apresentados nesta seção devem utilizar somente os dados originais para validação do resultado obtido. Isso significa que não se sabe *a priori* qual é a partição ideal para a rede.

### 3.1.1.1 Função *Performance P*

Essa função conta o número de pares de vértices interpretados corretamente, ou seja, vértices que são conectados por uma aresta encontram-se na mesma comunidade e pares de vértices que não são conectados por uma aresta encontram-se em comunidades diferentes. O cálculo da *performance P* para uma partição  $p$  qualquer pode ser definida como (FORTUNATO, 2010):

$$P(p) = \frac{|\{(i, j) \in E, C_i = C_j\}| + |\{(i, j) \notin E, C_i \neq C_j\}|}{n(n-1)/2} \quad (3.4)$$

onde  $E$  é o conjunto de arestas da rede,  $n$  é o número total de vértices na rede, e  $C_i$  e  $C_j$  são as comunidades dos vértices  $i$  e  $j$ , respectivamente. Por definição,  $0 \leq P(p) \leq 1$ . A função *performance P* é uma função de qualidade porque atribui um número para cada partição, permitindo que as partições sejam comparadas. Aquela com o maior valor é considerada a melhor partição.

### 3.1.1.2 Função Modularidade $Q$

A função de modularidade  $Q$  é a mais popular dentre os métodos de validação de algoritmos de detecção. Ela foi apresentada por Newman e Girvan (NEWMAN; GIRVAN, 2004), e vários algoritmos utilizam essa medida inclusive como função objetivo para alcançar a melhor partição, como os algoritmos Fast Greedy e Walktrap (Seções 3.2.1 e 3.2.4, respectivamente).

A função de *modularidade* é baseada na ideia de que não é esperada uma estrutura de comunidades em grafos aleatórios. Portanto, caso a densidade de arestas num subgrafo seja comparada à densidade esperada num grafo similar, mas com ligações randômicas, a existência de comunidades pode ser revelada. A densidade esperada de arestas depende do modelo nulo escolhido. O *modelo nulo* é uma cópia do grafo original mas sem estrutura de comunidades (FORTUNATO, 2010).

Em (NEWMAN; GIRVAN, 2004) a função de modularidade é definida da seguinte forma: Seja uma matriz  $\mathbf{e}$   $k \times k$ , onde  $k$  é o número de comunidades numa dada partição da rede. Cada elemento  $e_{ij}$  irá representar a fração dentre todas as arestas que ligam as comunidades  $i$  e  $j$ . A partir da matriz  $\mathbf{e}$ , é definido o *traço*  $Tr \mathbf{e} = \sum_i e_{ii}$  que fornece a fração de arestas na rede que são intra-comunidade. Um valor alto do traço é esperado caso a divisão em comunidades tenha sido bem feita, mas apenas o traço não é um bom indicador, pois ele não é capaz de identificar, por exemplo, se duas ou mais comunidades foram

identificadas como uma só. Nesse caso o valor do traço seria ainda maior.

Então é definida a soma das linhas (ou colunas)  $a_i = \sum_j e_{ij}$ , que representa a fração de arestas que se ligam a vértices dentro da comunidade  $i$ . Ou seja, numa rede onde as arestas são colocadas sem levar em conta a qual comunidades elas pertencem, obteríamos  $e_{ij} = a_i a_j$ . Portanto, a equação da modularidade por ser descrita como (NEWMAN; GIRVAN, 2004):

$$Q = \sum_i (e_{ii} - a_i^2) = Tr \mathbf{e} - \|\mathbf{e}^2\| \quad (3.5)$$

Onde  $\|\mathbf{e}^2\|$  é a soma dos elementos da matriz  $\mathbf{e}^2$ . Essa medida calcula a fração de arestas na rede que conectam vértices do mesmo tipo (ou seja, que pertençam a mesma comunidade) menos o valor esperado numa rede com a mesma estrutura de vértices mas com ligações randômicas.

Em (FORTUNATO, 2010), a função de modularidade é reescrita da seguinte maneira:

$$Q = \frac{1}{2m} \sum_{ij} (A_{ij} - P_{ij}) \delta(C_i, C_j) \quad (3.6)$$

Onde  $A_{ij}$  é a matriz de adjacência,  $m$  é o número total de arestas do grafo, e  $P_{ij}$  representa o número esperado de arestas entre os vértices  $i$  e  $j$  no modelo nulo. A função  $\delta$  retorna 1, caso os vértices  $i$  e  $j$  estejam na mesma comunidade, e 0 caso contrário. A Equação 3.6 retira a dependência que o modelo nulo tinha sobre a matriz  $\mathbf{e}$  no caso da Equação 3.5e, portanto, o valor de  $P_{ij}$  pode ser manipulado.

O caso mais básico do modelo nulo seria o caso onde este teria o mesmo número de arestas, mas elas seriam colocadas com igual probabilidade entre os pares de vértices. Ou seja,  $P_{ij} = p = 2m/(n(n-1)), \forall i, j$ , que é um valor constante. No entanto, esse modelo não seria uma boa descrição de redes reais, pois o ideal é que o modelo nulo tivesse a mesma distribuição de graus do modelo original.

Nesse caso, um vértice pode ser ligado a qualquer outro vértice do grafo e a probabilidade que os vértices  $i$  e  $j$ , com graus  $k_i$  e  $k_j$ , sejam conectados, pode ser calculado de forma simples. A probabilidade  $p_i$  de que uma aresta incidente em  $i$  seja escolhida é  $k_i/2m$ . Portanto, a probabilidade de conexão entre os vértices  $i$  e  $j$  é  $p_i p_j$ . O resultado é  $k_i k_j / 4m^2$ , o que leva a um número esperado de  $P_{ij} = 2m p_i p_j = k_i k_j / 2m$  de arestas entre  $i$  e  $j$ . Sendo

assim, a equação da modularidade pode ser reescrita como:

$$Q = \frac{1}{2m} \sum_{ij} \left( A_{ij} - \frac{k_i k_j}{2m} \right) \delta(C_i, C_j) \quad (3.7)$$

Na função modularidade, quanto maior o número de arestas dentro de uma comunidade exceder o número esperado de arestas, melhor é definida essa comunidade. Ou seja, quanto maior for o valor da modularidade  $Q$ , melhor será a partição. No entanto, existe alguns problemas relacionados a função de modularidade, que serão mostrados a seguir.

### 3.1.1.3 Limites da Modularidade $Q$

O modelo nulo da modularidade assume que todo vértice  $i$  pode ser conectado com qualquer outro vértice  $j$  na rede. E, portanto, o número esperado de arestas entre estes é  $p_{ij} = k_i k_j / 2m$ . Da mesma forma, o número esperado de arestas entre duas comunidades  $C_i$  e  $C_j$  com grau total  $K_{C_i}$  e  $K_{C_j}$  é  $P_{C_i, C_j} = K_{C_i} K_{C_j} / 2m$ . A variação do valor da modularidade com respeito à junção de  $C_i$  e  $C_j$  numa única comunidade ou como comunidades separadas é  $\Delta Q_{C_i, C_j} = l_{C_i, C_j} / m - K_{C_i} K_{C_j} / 2m$ , sendo  $l_{C_i, C_j}$  o número de arestas que ligam as comunidades  $C_i$  e  $C_j$ .

Se  $l_{C_i, C_j} = 1$ , ou seja, se existe apenas uma aresta ligando as duas comunidades, espera-se que o valor da modularidade para o caso em que as duas comunidades são colocadas separadas seja maior, mas se o valor de  $K_{C_i} K_{C_j} / 2m < 1$ , então  $\Delta Q_{C_i, C_j} > 0$ . Ou seja, supondo que  $K_{C_i} \sim K_{C_j} = K$ , conclui-se que, se  $K \lesssim \sqrt{2m}$  e existir alguma conexão entre as comunidades  $C_i$  e  $C_j$ , o valor da modularidade será maior se elas forem colocadas numa mesma comunidade (FORTUNATO; BARTHÉLEMY, 2007; FORTUNATO, 2010).

Se os subgrafos forem suficientemente pequenos em grau, o número esperado de arestas para o modelo nulo pode ser menor que um, então mesmo a mais fraca conexão, ou seja, apenas uma aresta, é suficiente para manter as comunidades juntas. Na Figura 3.1, o grafo é formado por  $n_c$  cliques idênticos, com  $l$  vértices cada, conectados por uma única aresta. Espera-se que a partição com o maior valor de  $Q$  seja a que cada clique seja uma comunidade. No entanto, se  $n_c$  for maior que  $l^2$ , a modularidade será maior para partições em que as comunidades são grupos de cliques, como os pares indicados pela linha pontilhada.

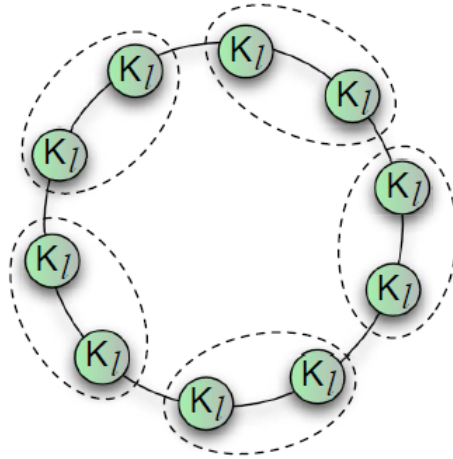


Figura 3.1 - Limite da Modularidade. A divisão natural em comunidades está representada pelos cliques  $K_l$ , mas o maior valor da modularidade é obtido se as comunidades forem definidas por pares de cliques, como indicado pelas linhas pontilhadas. Fonte: Figura extraída de Fortunato (2010).

### 3.1.1.4 Modularidade Dinâmica $Q_D$

Em (MUCHA et al., 2010) é apresentada uma função de qualidade baseada na modularidade (Seção 3.1.1.2), mas com aplicações em Redes Dinâmicas. A Figura 3.2 exemplifica uma rede com variação no tempo. As linhas sólidas são representadas por matrizes de adjacência  $A_{ijs}$ , onde o elemento  $a_{ijs}$  caracteriza a existência ou não de uma aresta entre os vértices  $i$  e  $j$  no *snapshot*  $s$ . Já as linhas pontilhadas são representadas por matrizes  $B_{jrs}$ , onde o elemento  $b_{jrs}$  caracteriza o acoplamento do vértice  $j$  entre os *snapshots*  $r$  e  $s$ .

A partir dessas duas matrizes a força de um vértice  $j$  no *snapshot*  $s$  pode ser calculada como  $\kappa_{js} = \sum_i A_{ijs} + \sum_r B_{jrs}$ , onde o primeiro somatório calcula a força do vértice no *snapshot* em questão, e o segundo somatório calcula a força do vértice nas conexões inter-*snapshots*. No caso de redes sem peso, a força do vértice é equivalente ao grau do vértice nas duas matrizes.

Na equação original da modularidade (Equação 3.7) o valor final do somatório é dividido por  $2m$ , onde  $m$  é o número total de arestas da rede. Na modularidade dinâmica esse valor é substituído por  $2m_D$ , onde  $2m_D = \sum_{js} \kappa_{js}$ , ou seja, a soma dos graus (no caso de redes sem peso) dos vértices tanto inter-*snapshot* como intra-*snapshot*.

A equação da modularidade dinâmica é escrita como (MUCHA et al., 2010):

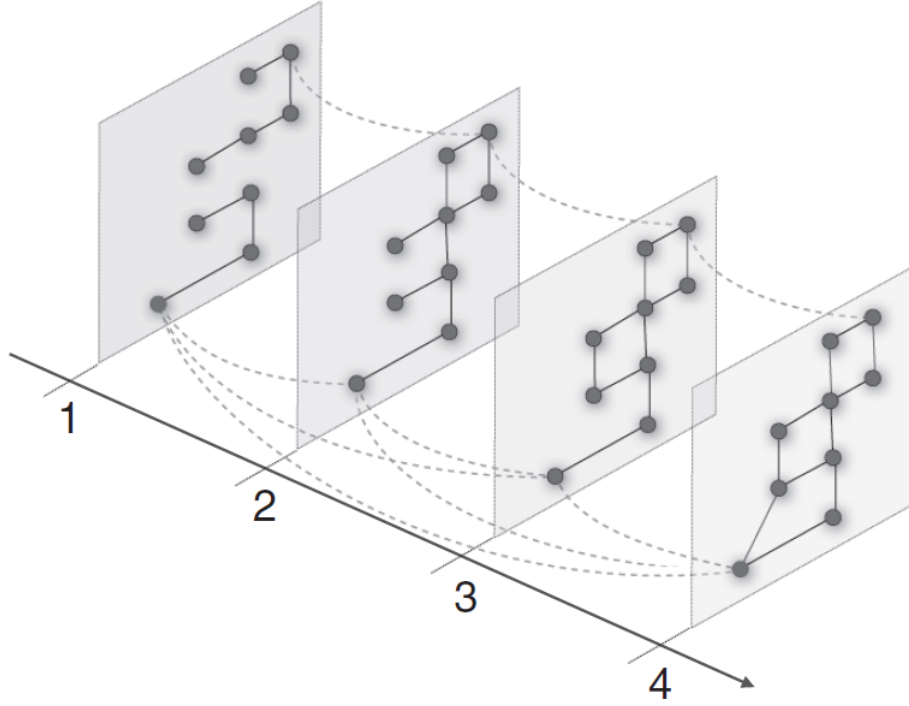


Figura 3.2 - Rede Dinâmica com 4 *snapshots*,  $s = \{1, 2, 3, 4\}$ , representados por matrizes  $A_{ijs}$  que armazenam conexões intra-*snapshots* (linhas sólidas). As conexões inter-*snapshots* (linhas pontilhadas) são armazenadas por matrizes  $B_{jrs}$ , que representam o acoplamento entre o vértice  $j$  nos *snapshots*  $r$  e  $s$ . Fonte: Figura extraída de Mucha et al. (2010).

$$Q_D = \frac{1}{2m_D} \sum_{ijrs} \left[ \left( A_{ijs} - \frac{k_{is}k_{js}}{2m_s} \right) \delta_{sr} + B_{jrs} \delta_{ij} \right] \delta(C_{is}, C_{jr}) \quad (3.8)$$

Onde,  $k_{is}$  e  $k_{js}$  são os graus dos vértices  $i$  e  $j$  no *snapshot*  $s$ . O valor  $m_s = \sum_j k_{js}$  é o número total de arestas dentro do *snapshot*  $s$ . As funções  $\delta$  são responsáveis por incorporar os termos do somatório ao valor final quando for o caso, ou seja,  $\delta_{sr} = 1$  se  $s = r$ , e  $\delta_{sr} = 0$  caso contrário,  $\delta_{ij} = 1$  se  $i = j$ , e  $\delta_{ij} = 0$  caso contrário, e, por fim,  $\delta(C_{is}, C_{jr}) = 1$  se a comunidade do vértice  $i$  no *snapshot*  $s$  for igual à comunidade do vértice  $j$  no *snapshot*  $r$ , e  $\delta(C_{is}, C_{jr}) = 0$  caso contrário.

### 3.1.1.5 Função Modularidade com Penalidade de Divisão $Q_{pd}$

Na Equação 3.7 a função  $\delta$  serve para que apenas pares de vértices de uma mesma comunidade contribuam para o somatório. Sendo assim, a equação da modularidade pode ser reescrita para que a somatória aja sobre as comunidades (CHEN et al., 2013):

$$Q = \sum_{i=1}^{n_c} \left[ \frac{l_{C_i}}{m} - \left( \frac{k_{C_i}}{2m} \right)^2 \right] \quad (3.9)$$

Na equação 3.9,  $n_c$  corresponde ao número de comunidades na rede,  $l_{C_i}$  é o número total de arestas dentro da comunidade  $C_i$  e  $k_{C_i}$  a soma dos graus de todos os vértices dentro da comunidade  $C_i$ . O grau da comunidade também pode ser expresso em função da quantidade de arestas,  $k_{C_i} = 2l_{C_i}^{in} + l_{C_i}^{out}$ , onde  $l_{C_i}^{in}$  é o número total de arestas dentro da comunidade e  $l_{C_i}^{out}$  é o número total de arestas que ligam a comunidade  $C_i$  a outras comunidades.

Em (CHEN et al., 2013) são apresentadas duas novas modificações da equação de modularidade. Nesta seção, a função que foi chamada de *Modularidade com Penalidade de Divisão* será apresentada, e a função chamada *Modularidade com Densidade* será apresentada na Seção 3.1.1.6. Segundo o autor, a inclusão da penalidade resolve algumas das limitações atribuídos a função da modularidade, mas acaba por deixar o problema do limite da modularidade ainda pior. Então é incluída na equação da modularidade com penalidade a densidade das comunidades, o que resolve por completo o problema do limite.

No artigo (CHEN et al., 2013), sete diferentes situações são apresentadas, em todas elas existem dois grupos de vértices bem conectados entre si, mas no primeiro exemplo não existem arestas conectando esses dois grupos, no segundo exemplo eles são fracamente conectados, e assim por diante, até a sétima situação onde as duas comunidades são fortemente conectadas se tornando uma comunidade só.

As três primeiras situações devem ter como resultado duas comunidades. Na quarta situação, o resultado é ambíguo, ou seja, tanto uma comunidade como duas comunidades podem ser resultados aceitáveis. E as três últimas situações devem ter como resultado uma comunidade apenas.

Quando usada apenas a modularidade (Equação 3.7), ela falha nas situações quatro, cinco e seis, pois resulta em um valor maior para duas comunidades na situação quatro, quando deveria dar o mesmo valor para os dois resultados, e nas situações cinco e seis, ela prioriza duas comunidades menores em vez de uma única maior.

O autor então propõe a adição de uma penalidade para as arestas entre comunidades diferentes. Ou seja,  $Q_{pd} = Q - PD$ . A modularidade mede o efeito positivo da existência de arestas entre grupos de nós. A penalidade por divisão irá subtrair da soma a fração de arestas que conectam vértices de comunidades diferentes, ou seja, mede o efeito negativo de ignorar arestas que ligam as comunidades. A função da *penalidade por divisão* é:

$$PD = \sum_{i=1}^{n_c} \left[ \sum_{\substack{j=1 \\ i \neq j}}^{n_c} \frac{l_{C_i, C_j}}{2m} \right] \quad (3.10)$$

Onde  $l_{C_i, C_j}$  é o número de arestas que ligam as comunidades  $C_i$  e  $C_j$ . Acrescentando então a penalidade na Equação 3.9, temos:

$$Q_{pd} = \sum_{i=1}^{n_c} \left[ \frac{l_{C_i}}{m} - \left( \frac{g_{C_i}}{2m} \right)^2 - \sum_{\substack{j=1 \\ i \neq j}}^{n_c} \frac{l_{C_i, C_j}}{2m} \right] \quad (3.11)$$

A Equação 3.11, assim como a função original da modularidade, é independente do número de vértices. Isso faz com que duas redes similares quanto ao número de arestas, mas onde uma delas possua um número bem menor de vértices, tenham modularidade similares. Quando a rede com menos vértices deveria ter uma modularidade maior já que sua maior densidade indica comunidades mais bem definidas.

Outro problema é relacionado ao limite da modularidade, apresentado na Seção 3.1.1.3. Quando aplicada a penalidade para esse caso, a variação  $\Delta Q_{C_i, C_j}$  é ainda maior, piorando ainda mais os resultados para casos como o da Figura 3.1.

### 3.1.1.6 Função Modularidade com Densidade $Q_{ds}$

Para resolver os dois problemas citados ao final da Seção 3.1.1.5, Chen et al. (2013) acrescentaram a densidade da comunidade na equação da modularidade com penalidade  $Q_{pd}$ . A nova métrica, chamada *Modularidade com Densidade*  $Q_{ds}$ , é testada em todas as setes situações já mencionadas na Seção 3.1.1.5, e mede corretamente a melhor partição para todas elas, inclusive na situação quatro, onde o resultado é ambíguo e os valores para uma e duas comunidades são similares.

Quando a modularidade com densidade é aplicada à situação da Figura 3.1, também obtém melhores resultados quando os cliques são considerados comunidades independentes. Ou seja, a nova métrica evita o problema do limite. E como introduz a densidade na métrica, o problema das duas redes com números diferentes de vértices é resolvido, com o valor sendo mais alto para comunidades mais densas, ou seja, mais bem definidas.

A equação final para a função de modularidade com densidade é:



$$Q_{ds} = \sum_{i=1}^{n_c} \left[ \frac{l_{C_i}}{m} \rho_{C_i} - \left( \frac{g_{C_i}}{2m} \rho_{C_i} \right)^2 - \sum_{\substack{j=1 \\ i \neq j}}^{n_c} \frac{l_{C_i, C_j}}{2m} \rho_{C_i, C_j} \right],$$

sendo

$$\rho_{C_i} = \frac{2l_{C_i}}{n_{C_i}(n_{C_i} - 1)} \quad (3.12)$$

e

$$\rho_{C_i, C_j} = \frac{l_{C_i, C_j}}{n_{C_i} n_{C_j}}$$

onde,  $\rho_{C_i}$  é a densidade interna da comunidade  $C_i$  e  $\rho_{C_i, C_j}$  é a densidade entre as comunidades  $C_i$  e  $C_j$ . O termo  $n_{C_i}$  corresponde ao número de vértices na comunidade  $C_i$ . Para mais detalhes sobre os resultados obtidos e para as provas de que a modularidade com densidade  $Q_{ds}$  supera os problemas existentes na modularidade  $Q$ , ver (CHEN et al., 2013).

### 3.1.2 Métodos com Critérios Externos

Nesta seção serão apresentadas medidas que utilizam uma estrutura pré-especificada dos dados para a validação. Essa estrutura reflete o que acredita-se ser o melhor resultado para o conjunto de dados. No caso das redes, o conhecimento prévio é saber quais vértices pertencem à mesma comunidade, ou seja, quais são os grupos de vértices que deveriam ser alocados juntos.

#### 3.1.2.1 Índice *Rand* Corrigido

O índice *Rand* Corrigido é uma normalização do índice *Rand* (HUBERT; ARABIE, 1985; RAND, 1971). O índice *Rand* determina a similaridade entre duas partições, pela concordância, positiva ou negativa, na associação de pares de objetos nos *clusters*. Ou seja, o índice penaliza as associações diferentes de pares de objetos nas duas partições. Para a utilização desse índice para validação, uma das partições deve ser previamente conhecida, e a outra deve ser a partição que está sendo avaliada.

Para cada par  $(v_i, v_j)$  de vértices no conjunto de dados, calcula-se o valor de quatro medidas, são elas:

- SS: se os dois vértices pertencem à mesma comunidade na partição sendo avaliada e na partição previamente conhecida.

- SD: se os dois vértices pertencem à mesma comunidade na partição sendo avaliada e a comunidades diferentes na partição previamente conhecida.
- DS: se os dois vértices pertencem a comunidades diferentes na partição sendo avaliada e à mesma comunidade na partição previamente conhecida.
- DD: se os dois vértices pertencem a comunidades diferentes na partição sendo avaliada e na partição previamente conhecida.

Sendo  $M = SS + SD + DS + DD = n(n-1)/2$ , onde  $n$  é o número de vértices no conjunto de dados, o índice Rand é calculado da seguinte forma:

$$R = \frac{(SS + DD)}{M} \quad (3.13)$$

O Rand Corrigido (RC) acrescenta na fórmula o valor esperado, normalizando o valor do índice. Dada uma rede  $G$  com  $n$  vértices e duas partições dessa rede em comunidades  $X = \{X_1, X_2, \dots, X_r\}$  e  $Y = \{Y_1, Y_2, \dots, Y_s\}$ , a sobreposição entre  $X$  e  $Y$  pode ser resumida numa tabela de contingência  $[n_{ij}]$  onde cada entrada  $n_{ij}$  denota o número de objetos comuns entre  $X_i$  e  $Y_j$ :  $n_{ij} = |X_i \cap Y_j|$ .

Sendo  $a_i = \sum_j n_{ij}$  e  $b_j = \sum_i n_{ij}$ , ou seja, as somas das linhas e das colunas da tabela de contingência, respectivamente, o valor esperado pode ser calculado por:

$$ER = \left[ \sum_i \binom{a_i}{2} \sum_j \binom{b_j}{2} \right] / \binom{n}{2} \quad (3.14)$$

Por fim, a fórmula para calcular o índice Rand Corrigido é expresso desta maneira (VINH et al., 2009):

$$RC = \frac{\sum_{ij} \binom{n_{ij}}{2} - ER}{\frac{1}{2} \left[ \sum_i \binom{a_i}{2} + \sum_j \binom{b_j}{2} \right] - ER} \quad (3.15)$$

O valor se aproxima de 0, podendo inclusive ser negativo, para partições aleatórias e do valor 1 quando as partições casam perfeitamente.

### 3.1.2.2 Informação Mútua Normalizada

Formalmente, a informação mútua ( $I$ ) de duas variáveis aleatórias discretas  $X$  e  $Y$  pode ser definida como:

$$I(X, Y) = \sum_{y \in Y} \sum_{x \in X} p(x, y) \log \left( \frac{p(x, y)}{p(x)p(y)} \right) \quad (3.16)$$

Informação mútua mede a informação que é compartilhada entre  $X$  e  $Y$ , ou seja, mede o quanto que conhecendo uma dessas variáveis reduz a incerteza sobre a outra. Se  $X$  e  $Y$  forem independentes, conhecer  $X$  não me dará nenhuma informação sobre  $Y$  e vice-versa, então o valor de  $I(X, Y)$  é zero. Em outro extremo, se  $X$  e  $Y$  são idênticas, então toda informação dada por  $X$  é compartilhada por  $Y$  e vice-versa. Como resultado, o valor de  $I(X, Y)$  será o valor da entropia de  $X$ , que terá o mesmo valor da entropia de  $Y$ .

Na teoria da informação, entropia é a medida da incerteza numa variável aleatória. Esse termo se refere geralmente à entropia de Shannon, que é a média da imprevisibilidade de uma variável aleatória. A entropia de Shannon fornece um limite absoluto da melhor compressão possível de qualquer mensagem, assumindo que a mensagem será representada como uma sequência independente e distribuída de forma idêntica de variáveis aleatórias.

A equação para calcular a informação mútua normalizada é (LANCICHINETTI; FORTUNATO, 2009b):

$$I_{norm}(X, Y) = \frac{2I(X, Y)}{H(X) + H(Y)} \quad (3.17)$$

onde,  $H(X)$  e  $H(Y)$  correspondem à entropia de  $X$  e  $Y$ , respectivamente. Para o uso em redes, uma das variáveis aleatórias deve ser a divisão em comunidades pré-estabelecidas e a outra deve ser a divisão em comunidades detectadas. Se a partição encontrada for idêntica à partição estipulada,  $I_{norm} = 1$ , caso elas forem totalmente independentes,  $I_{norm} = 0$  (DANNON et al., 2005). Essa medida é atualmente a mais utilizada para validação dos resultados obtidos por algoritmos de detecção de comunidades (FORTUNATO, 2010).

### 3.1.3 Métodos com Critérios Relativos

O método que será apresentado nesta seção compara diversos resultados de agrupamento para decidir qual é o melhor em algum aspecto. Pode ser usado para decidir entre vários algoritmos, ou para decidir o melhor conjunto de parâmetros de entrada para o mesmo

algoritmo.

### 3.1.3.1 Índice Dunn

O índice Dunn, originalmente, é calculado utilizando alguma combinação de distâncias inter-*cluster* e intra-*cluster* (DUNN, 1973). A equação pode ser generalizada da seguinte forma:

$$D = \min_{1 \leq i \leq n_c} \left\{ \min_{\substack{1 \leq j \leq n_c \\ j \neq i}} \left\{ \frac{d(C_i, C_j)}{\max_{1 \leq k \leq n_c} \Delta_k} \right\} \right\} \quad (3.18)$$

onde  $d(C_i, C_j)$  é a distância inter-*cluster* entre os *clusters*  $C_i$  e  $C_j$ ,  $\Delta_k$  é a distância intra-*cluster* do *cluster*  $C_k$ , e  $n_c$  é o número de *clusters*. Para que essa medida possa ser usada em redes, é preciso definir quais serão as medidas de distância inter-comunidade e intra-comunidade.

Como opção, a densidade pode ser usada. Quanto maior a densidade de uma comunidade, menos os vértices dessa comunidade estão distantes entre si, e o mesmo vale para a distância entre duas comunidades, quanto menor a densidade, mais distantes elas estão. Então o inverso da densidade deve indicar as distâncias inter-comunidade e intra-comunidade, já que são valores inversamente proporcionais. Assim sendo:

$$d(C_i, C_j) = \frac{1}{\rho_{C_i, C_j}}$$

e

$$\Delta_k = \frac{1}{\rho_{C_k}} \quad (3.19)$$

As fórmulas para calcular os valores de  $\rho_{C_i, C_j}$  e  $\rho_{C_k}$  podem ser encontradas na Equação 3.12. Essa formulação tem um problema peculiar. Se existir pelo menos uma comunidade que não esteja bem definida, mesmo que as outras estejam, irá afetar o valor final, já que o denominador da Equação 3.18 contém a função max em vez de uma média. Ou seja, se este indicador for usado, é preciso ter isso em mente na hora de analisar os dados, já que o valor do índice será baixo para o caso em que existir uma anomalia nos dados.

O objetivo do índice Dunn é calcular o quão compacto são os *clusters* e quão distantes estão estes *clusters* entre si. Para um conjunto de dados específico, quanto mais alto o

valor do índice, melhor está sua divisão.

## 3.2 Algoritmos de Detecção de Comunidades

Nesta seção serão mostrados alguns algoritmos que se propõe a detectar comunidades em redes complexas. Esses algoritmos, com exceção dos algoritmos dinâmicos apresentados na Seção 3.2.5, serão usados posteriormente nos experimentos para a exploração da metodologia que é proposta neste trabalho.

### 3.2.1 Algoritmo Fast Greedy

O algoritmo Fast Greedy (CLAUSET et al., 2004) é baseado na medida de modularidade (Seção 3.1.1.2). A modularidade é uma medida entre a rede e uma divisão em comunidades dessa rede, ou seja, ela calcula um valor para a divisão proposta, e quanto maior esse valor, melhor essa divisão é, considerando a estrutura da rede. A ideia do algoritmo é que se altos valores da modularidade correspondem a uma boa divisão em comunidades, então é possível achar boas divisões realizando-se uma busca dentre as possíveis divisões com altos valores de modularidade.

Mas achar o máximo global dessa busca não seria tarefa fácil considerando que o crescimento de possibilidades de divisão é exponencial ao número de vértices na rede, isso impossibilitaria que todas as combinações possíveis sejam testadas, portanto a utilização de uma heurística deve ser empregada. No caso do algoritmo Fast Greedy a heurística utilizada é a gulosa.

O algoritmo começa com cada vértice sendo o único membro de uma comunidade, e repetidamente, dois grupos de vértices vão sendo agregados. Os dois grupos de vértices são escolhidos de tal forma que a junção destes causem o maior aumento na modularidade. Esse processo se repete até que todos os vértices estejam reunidos numa única comunidade. Todo esse processo pode ser traduzido num dendrograma que é uma decomposição hierárquica dos passos do algoritmo. Esse dendrograma então é analisado, e o corte que tiver o maior valor de modularidade é escolhido como resultado final. A Figura 3.3 ilustra esse processo final.

O algoritmo apresentado em (CLAUSET et al., 2004) é um melhoramento do algoritmo apresentado em (NEWMAN, 2004). A diferença entre os dois está na forma como as estruturas de dados foi armazenada e na forma como a modularidade foi calculada. Basicamente, em vez da modularidade da rede ser recalculada a cada passo do algoritmo, ela é calculada a partir do resultado anterior no Fast Greedy, causando um ganho no tempo de execução do algoritmo.

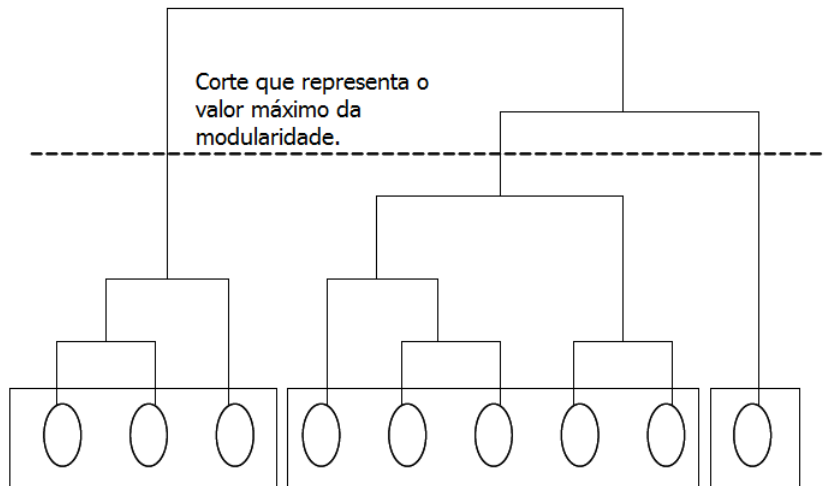


Figura 3.3 - Ilustração de um dendrograma com o corte que representa o valor máximo da modularidade. Fonte: Figura adaptada de Raghavan et al. (2007).

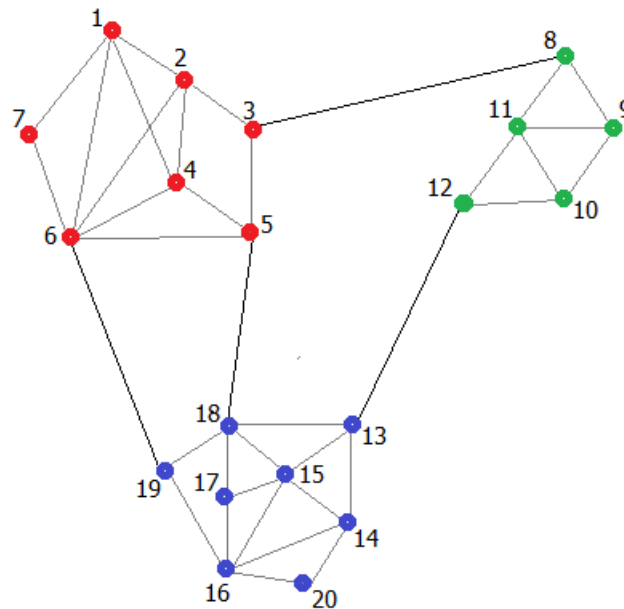
### 3.2.2 Algoritmo Infomap

O Infomap parte da atuação de um *random walker* na rede, e utilizando teoria dos códigos, especificamente o código de Huffman, propõe uma função objetivo a ser minimizada. O objetivo do algoritmo é encontrar uma partição para rede em que os vértices são agrupados em módulos, onde cada vértice pertence a um e somente um módulo. Essa partição deverá agrupar em um mesmo módulo vértices nos quais as informações, que são representadas pelo *random walker*, fluem rápida e facilmente (ROSVALL et al., 2009; ROSVALL; BERGSTROM, 2008).

O primeiro passo do algoritmo é calcular, a partir da atuação de um *random walker*  $q$ , uma variável aleatória  $P$  que é a distribuição da frequência de visitas que o elemento  $q$  faz aos vértices da rede num tempo infinito. Ou seja,  $P = \{p_i\}, 1 \leq i \leq n$ , sendo  $n$  o número de vértices da rede.

O código de Huffman tem por objetivo comprimir um código. Sendo esse código o caminho percorrido pelo elemento  $q$  dentro da rede, cada vértice deverá ter um código binário único associado, baseado na distribuição  $P$ . Vértices com maiores probabilidades de serem visitados deverão ter nomes menores. enquanto os poucos visitados terão nomes maiores. Além disso, é levado em consideração que num caminho percorrido pelo elemento  $q$ , os vértices que deverão ser agrupados num mesmo módulo aparecerão em sequência e por longos períodos. Isso porque um *random walker* tende a ficar “preso” num módulo e raramente pular de um módulo para outro. A Figura 3.4 mostra uma rede hipotética com 3

módulos que podem ser identificados pelas cores vermelha, verde e azul. A figura também mostra um possível caminho percorrido por um *random walker*, e o fenômeno discutido acima pode ser observado, pois os movimentos entre módulos são bem mais raros que movimentos dentro dos módulos.



Caminho (tamanho 50):

7 6 4 1 6 5 18 17 16 20 14 15 17 18 13 12 10 11 12 13 15 18 5 4 6 2 3 8 9 11 10 9  
11 12 13 15 17 16 20 14 15 18 5 4 1 2 6 19 16 20

Figura 3.4 - Uma rede qualquer, com 20 vértices divididos em 3 módulos, representadas pelas cores vermelha, verde e azul. Abaixo, um possível caminho percorrido por um *random walker* com 50 passos.

Para comprimir esse código, o Infomap usa uma descrição de dois níveis, assim é possível utilizar ainda menos *bits* para descrever o caminho percorrido pelo elemento  $q$ . Os dois níveis são representados da seguinte forma: num nível mais alto estão os módulos, que terão um código de Huffman próprio, e para cada módulo também existirá um código de Huffman para descrever seus elementos. Essa divisão em dois níveis possibilita que códigos sejam reutilizados em módulos diferentes e, portanto, que os nomes dos vértices tenham a possibilidade de serem menores mesmo que tenham uma probabilidade pequena de serem visitados. Na Tabela 3.1 estão descritos os códigos associados a cada uma dos vértices e módulos numa codificação de um nível que é a mais conhecida do código de

Tabela 3.1 - Códigos de cada um dos vértices e módulos para codificação de um nível e codificação de dois níveis. O número de visitas foi calculado a partir do caminho descrito na Figura 3.4.

Módulo/Vértice	Nº de visitas	Codificação 1 nível	Codificação 2 níveis
m1	16	-	10
v1	2	00010	100
v2	2	00011	101
v3	1	001000	0100
v4	3	0100	110
v5	3	0101	111
v6	4	0000	00
v7	1	111010	0101
s1	-	-	011
m2	11	-	11
v8	1	11011	000
v9	2	01110	010
v10	2	011111	011
v11	3	1010	10
v12	3	1011	11
s2	-	-	001
m3	23	-	0
v13	3	0110	010
v14	2	00101	001
v15	4	0011	110
v16	3	1000	011
v17	3	1001	100
v18	4	111	111
v19	1	001001	0000
v20	3	1100	101
s3	-	-	0001

Huffman, e a codificação para o caso da configuração em dois níveis. O número de visitas mostrado na coluna 2 é baseado no caminho descrito na Figura 3.4. É possível perceber ainda, que de fato existe a reutilização de códigos na configuração de dois níveis, e a ambiguidade é resolvida pelos códigos m1, m2 e m3, que identificam a qual módulo o vértice em questão pertence.

Associado a cada módulo, além do código de identificação, também existirá um código de saída. Ou seja, dada uma sequência de *bits* descrevendo um caminho percorrido por um *random walker*, o primeiro código identifica o módulo onde o elemento  $q$  inicia o percurso, em seguida os códigos identificam os vértices que o elemento visitou enquanto



ainda estava dentro do módulo inicial, até que o código de saída é identificado. Em seguida, o próximo código identificará para qual módulo o elemento seguiu em seu percurso, e novamente a sequência de códigos identificam os vértices visitados em sequência até que o código de saída desse segundo módulo seja identificado e o código do novo módulo identifique para onde o elemento  $q$  seguiu no percurso, e assim por diante. Na Tabela 3.1 é possível identificar esses códigos associados aos módulos e aos seus respectivos códigos de saída, estes representados pelos nomes de  $m_1$ ,  $m_2$ ,  $m_3$ ,  $s_1$ ,  $s_2$  e  $s_3$ . Na Figura 3.5 as configurações dos códigos para um nível e dois níveis do caminho que foi descrito na Figura 3.4 são explicitadas. Mesmo sendo acrescentados os códigos de entrada e saída dos módulos, existe uma economia no número necessário de *bits* para descrever o mesmo caminho.

Código de 1 nível (215 bits):

```
111010 0000 0100 00010 0000 0101 111 1001 1000 1100 00101 0011 1001 111
0110 1011 011111 1010 1011 0110 0011 111 0101 0100 0000 00011 001000 11011
01110 1010 011111 01110 1010 1011 0110 0011 1001 1000 1100 00101 0011 111
0101 0100 00010 00011 0000 001001 1000 1100
```

Código de 2 níveis (184 bits):

```
10 0101 00 110 100 00 111 011 0 111 100 011 101 001 110 100 111 010 0001 11
11 011 10 11 011 0 010 110 111 0001 10 111 110 00 101 0100 011 11 000 010 10
011 010 10 11 001 0 010 110 100 011 101 001 110 111 0001 10 111 110 100 101
00 011 0 0000 011 101
```

Figura 3.5 - Configuração dos códigos que descrevem o caminho descrito na Figura 3.4, utilizando os valores que estão na Tabela 3.1.

Mas para que essa sequência de *bits* que identificam o trajeto percorrido pelo *random walker* seja mínima, os módulos precisam ser apropriadamente identificados. Para isso, é introduzida uma função que calcula a quantidade média de *bits* necessários por passo para descrever o caminho do elemento  $q$  de acordo com uma partição  $M$  qualquer. Essa função, chamada *map equation* e denotada por  $L(M)$ , calcula os *bits* necessários para codificar cada um dos módulos e seus correspondentes códigos de saída, assim como os *bits* necessários para codificar cada um dos vértices dentro dos módulos. Tendo as probabilidades calculadas inicialmente e representadas pela variável aleatória  $P$ , o teorema de Shannon pode ser invocado. O teorema diz que quando usados  $x$  nomes para descrever  $x$  estados de uma variável aleatória  $X$ , o tamanho médio de compressão não pode ser menor que a entropia dessa variável:  $H(X) = -\sum_1^x p_i \log(p_i)$ . Assim sendo, a função  $L(M)$  é

limitada por baixo pela entropia  $H(P)$ , conforme ilustra a Equação 3.20.

$$L(M) = q_{\curvearrowright} H(Q) + \sum_{i=1}^m p_{\circlearrowleft}^i H(P^i) \quad (3.20)$$

A Equação 3.20 demonstra a *map equation*, e essa equação comprime dois termos: o primeiro é a entropia de movimento entre os módulos, e o segundo a entropia do movimento dentro dos módulos, incluindo o movimento de sair do módulo. Cada um dos módulos é associado a um peso que representa a frequência com que cada um desses movimentos ocorre de acordo com a partição  $M$ . Para detalhamento de cada um dos termos da função é recomendável a leitura do apêndice do artigo (ROSVALL; BERGSTROM, 2008).

A partir desse ponto o objetivo do algoritmo é minimizar a Equação 3.20. Para essa etapa, qualquer algoritmo de otimização da literatura poderia ser utilizado. No algoritmo Infomap a heurística escolhida foi a gulosa. No começo, cada vértice é associado a um módulo diferente, e dois a dois os módulos vão sendo fundidos de acordo com a junção que fará com que a  $L(M)$  seja diminuída ao máximo, e esse processo se mantém até que o limite seja atingido. No Infomap, além do algoritmo guloso, o resultado é refinado pelo *simulated annealing* (KIRKPATRICK et al., 1983). Nessa segunda fase, alguns vértices podem ser trocados de módulos caso haja uma redução no valor da *map equation*.

### 3.2.3 Algoritmo Label Propagation

A ideia principal do algoritmo Label Propagation (RAGHAVAN et al., 2007) é: supondo que um vértice  $v$  na rede tem vizinhos  $v_1, v_2, \dots, v_k$  e que cada vizinho carrega um rótulo que indica a qual comunidade ele pertence, então o vértice  $v$  irá decidir seu rótulo baseado nos rótulos dos seus vizinhos. Cada vértice escolhe participar da comunidade que é maioria entre seus vizinhos, e empates são resolvidos aleatoriamente. Cada nó é inicializado com rótulos únicos, e os rótulos são propagados dentro da rede. Grupos de nós mais densos alcançam um consenso quanto ao rótulo rapidamente, e o processo continua até que se estabeleça um equilíbrio na rede. Ao fim do processo, grupos de nós com o mesmo rótulo são agrupados numa mesma comunidade.

A cada passo do algoritmo uma ordem de visitação aos vértices é escolhida aleatoriamente, então, para cada vértice  $v$  na ordem escolhida, o rótulo é atualizado de acordo com os seus vizinhos. No começo do algoritmo, quando cada vértice possui seu próprio rótulo, pouco consenso entre os rótulos é estabelecido, mas conforme aumenta o número de iterações esses grupos vão ganhando maior influência, conseguindo agregar cada vez mais vértices. O processo se repete até que esse grupo alcance a borda de outro grupo,

quando estes começam a disputar vértices da fronteira entre eles.

Apesar da rede ser identificada como uma única comunidade ser um caso que possa satisfazer os critérios de parada do algoritmo, o processo de formação de consenso entre os grupos e a competição entre eles desencoraja esse resultado. No caso de grafos randômicos, com ligações homogêneas, o algoritmo identifica a rede toda como uma única comunidade.

O algoritmo pode enfrentar problemas na detecção de comunidades por sua dependência da ordem de visitação dos vértices escolhidos aleatoriamente no começo de cada iteração e também na quebra de empates quando o vértice tem duas ou mais opções de rótulos. A cada execução do algoritmo, o resultado é diferente, mesmo quando as condições iniciais são as mesmas.

### 3.2.4 Algoritmo Walktrap

O algoritmo Walktrap (PONS; LATAPY, 2006) é baseado na intuição de que *random walks* quando caminham dentro de uma rede tendem a ficar “presos” em componentes densamente conectados, que correspondem às comunidades. A partir dessa ideia, o algoritmo propõe uma medida que calcula uma distância entre vértices e/ou grupos de vértices baseado no caminho percorrido pelo *random walker* entre os dois componentes. Essa medida é então utilizada como cálculo de similaridade num algoritmo de *clustering* hierárquico.

Um *random walker* se move de um vértice a outro a cada passo. O vértice para o qual o *walker* se move é escolhido de forma uniforme e aleatória dentre os vizinhos do vértice em que ele se encontra. A sequência, ou caminho, de vértices percorridos pelo *walker* na rede é uma cadeia de Markov, sendo os estados da cadeia os vértices visitados. A cada passo, a probabilidade de transição de um vértice  $i$  para um vértice  $j$  é  $P_{ij} = \frac{A_{ij}}{d(i)}$ , sendo  $A$  a matriz de adjacência da rede e  $d(i)$  o grau do vértice  $i$ . A matriz  $P$  é definida como *matriz de transição*, e descreve o processo do *random walker*. Para calcular a probabilidade do *walker* ir do vértice  $i$  para o vértice  $j$  em  $t$  passos, basta calcular  $P^t$  e observar o elemento  $(P^t)_{ij}$ .

A medida proposta, que é chamada *distância  $r$* , é calculada a partir da matriz de transição  $P$ , descrita pela Equação 3.21. Para mais detalhes sobre, recomenda-se a leitura de (PONS; LATAPY, 2006).

$$r_{C_1 C_2} = \sqrt{\sum_{k=1}^n \frac{(P_{C_1 k}^t - P_{C_2 k}^t)^2}{d(k)}} \quad (3.21)$$

Onde  $C_1$  e  $C_2$  são os componentes que estão sendo medidos e  $P'_{ij}$  é uma simplificação de  $(P^t)_{ij}$ . Nota-se que  $C_x$  pode indicar inclusive vértices, se estes forem generalizados para uma comunidade de um único componente.

A partir da distância  $r$ , é usado um algoritmo de *clustering* hierárquico aglomerativo baseado no método Ward (WARD, 1963). Começando por uma partição  $\mathcal{P}_1 = \{\{v\}, v \in V\}$ , que considera que todo vértice  $v$  é uma comunidade, a distância  $r$  é calculada entre todas as comunidades que são adjacentes, ou seja, que tenham pelo menos uma aresta entre elas, e as duas que obtiverem a maior similaridade serão unidas numa única comunidade. Esse processo se repete até que todos os vértices estejam aglomerados numa única comunidade. Ao final do algoritmo é gerado um dendrograma, assim como no algoritmo Fast Greedy, e assim como este, um corte é feito na altura que obtiver maior valor da modularidade, como indicado pela Figura 3.3.

### 3.2.5 Detecção de Comunidades em Redes Complexas Dinâmicas

A maioria significativa de algoritmos para detecção de comunidades age sobre redes estáticas. Somente poucos e recentes trabalhos tratam de algoritmos para detecção em redes dinâmicas. Nesta seção serão apresentadas as abordagens mais comuns na detecção de comunidades quando a evolução no tempo é considerada, e alguns trabalhos serão citados.

Uma primeira abordagem, que será adotada para os experimentos neste trabalho (Capítulo 5), é chamada *aplicação longitudinal*. Trata-se de uma simples extensão de qualquer algoritmo de detecção em redes estáticas para detecção em redes dinâmicas. Consiste em aplicar o algoritmo de detecção em sucessivos *snapshots* da rede e fazer uma correlação entre os resultados obtidos a cada passo. Essa abordagem é utilizada em (PALLA et al., 2007), que utiliza o algoritmo proposto em (PALLA et al., 2005). É a técnica usada também em (ASUR et al., 2009), que utiliza o algoritmo desenvolvido em (DONGEN, 2000). E mais recentemente utilizada em (KIM; HAN, 2009), usando o algoritmo proposto em (XU et al., 2007).

Uma segunda abordagem, que pode ser considerada similar à descrita acima, faz a correlação entre sucessivos *snapshots* a partir dos vértices da rede. Ou seja, o primeiro passo é igual à abordagem anterior, em que um algoritmo para redes estáticas é aplicado em sucessivos *snapshots*, mas a correlação entre eles é feita considerando a evolução da pertinência de cada vértice na rede, além de considerar o surgimento ou desaparecimento destes. Essa abordagem é utilizada em (FENN et al., 2009), utilizando o algoritmo proposto em (REICHARDT; BORNHOLDT, 2006).

Em (WANG et al., 2008), essa abordagem é utilizada de forma diferente. A evolução da comunidade é descrita em termos de um pequeno conjunto de vértices centrais. Quando uma comunidade do tempo  $t + 1$  contém um vértice central de uma comunidade no tempo  $t$ , essa é considerada uma evolução desta. E assim, mudanças nas comunidades podem ser identificadas e analisadas.

Por fim, uma abordagem mais elaborada, que pode ser chamada *aplicação incremental*, na qual a detecção de comunidade do tempo  $t + 1$  é inicializada com informações derivadas da detecção no tempo  $t$ . As vantagens dessa abordagem é que há uma economia de processamento e leva a uma detecção mais consistente, já que resultados obtidos entre dois *snapshots* não são tão diferentes. Métodos de aplicação incremental ainda podem ser divididos em duas outras: uma que irá detectar as comunidades interativamente, sempre utilizando o resultado obtido anteriormente; e outra que só irá iniciar uma nova detecção caso um evento significativo ocorra na rede, como a inserção de um novo vértice.

Em (SUN et al., 2007), o algoritmo GraphScope é apresentado e a detecção de comunidades é feita visando minimizar a informação requerida para codificar o grafo e sua estrutura em comunidades, e cada nova detecção é inicializada a partir do resultado obtido na detecção anterior. Em (YANG et al., 2009) é usada uma técnica para estimar repulsão e atração entre os vértices, que acaba dividindo a rede em comunidades, e também utilizando como dados iniciais o resultado obtido no passo anterior. E em (LIN et al., 2008), o algoritmo FaceNet apresenta uma detecção de comunidades incremental baseada no *clustering evolucionário* (CHAKRABARTI et al., 2006). De acordo com esse método, a detecção é formulada como um problema de minimização de custo para armazenamento do grafo.

Em (FALKOWSKI et al., 2007) o conceito de vértices centrais é utilizado novamente, o algoritmo DenGraph só irá atualizar a detecção de comunidades caso uma atualização na estrutura da rede modifique o conjunto de vértices centrais. E em (FRANKE; GEYER-SCHULZ, 2009) é apresentado um algoritmo incremental baseado num *random walker*, e que também só inicia uma nova detecção a partir da percepção da mudança na estrutura da rede.

O algoritmo apresentado em (QUILES et al., 2013) pode ser interpretado como um algoritmo de aplicação incremental que mantém ao mesmo tempo uma detecção interativa e atualização a partir de um evento significativo na estrutura da rede. O algoritmo começa criando uma representação espacial de uma rede inicial, onde cada vértice é representado por um objeto no espaço chamado *partícula*. Na representação espacial, duas interações governam o movimento das partículas, a relacional e a espacial. A primeira é responsável pela atração entre as partículas, enquanto que a segunda é responsável pela repulsão.

Essas duas interações fazem com que partículas que representam vértices densamente conectados fiquem mais próximas no espaço. Enquanto que partículas que representam vértices que não têm uma relação próxima irão se afastar. Após algumas iterações, conglomerados de vértices irão se formar no espaço e podem então ser interpretados como as comunidades da rede.

A partir de uma rede, as partículas são aleatoriamente inseridas no espaço. Usando então as duas relações, as partículas são atraídas e repelidas por seus pares até que um estado de equilíbrio seja atingido. A partir desse estado, dada as posições das partículas no espaço, qualquer algoritmo de agrupamento pode ser utilizado, assinalando os vértices em comunidades. Esse processo é demonstrado nas Figuras 3.6(a)-(c).

Em um cenário dinâmico, qualquer mudança na estrutura da rede pode ser interpretada como uma perturbação do modelo, e um novo estado de equilíbrio será atingido após um período de tempo. Um experimento usando o algoritmo demonstra sua capacidade de lidar com essa situação. A partir de uma rede de com 128 vértices e 4 comunidades de 32 vértices cada (ver Seção 3.3.1), o algoritmo atinge o estado de equilíbrio. Após um tempo, 200 novas arestas são inseridas na rede, ligando duas comunidades distintas. Essas mudanças na rede fazem com que as duas comunidades afetadas possam ser interpretadas como uma só com 64 vértices. A partir dessa perturbação, o algoritmo passa a buscar um novo estado de equilíbrio que quando atingido aglomera os vértices das antigas comunidades num grupo único de partículas.

Todo esse processo pode ser acompanhado na Figura 3.6. Em (d) pode-se ver a nova rede com as 200 novas arestas ligando as comunidades representadas pelas cores azul claro e azul escuro. Em (e) o estado de equilíbrio atingido a partir da rede (a). Em (f)-(h) as relações de atração e repulsão movem as partículas no espaço. Por fim, em (i) o novo estado de equilíbrio, onde as partículas das comunidades azul escuro e azul claro são aglomeradas.

### 3.3 Avaliação de Algoritmos

Na Seção 3.2, foram apresentadas algumas técnicas para detecção de comunidades em redes complexas. Quando um novo algoritmo é projetado, é possível calcular sua complexidade. Contudo, para que a acurácia de detecção seja quantificada, testes comparativos precisam ser realizados.

Em computação, *benchmark* é o ato de executar um programa de computador ou um conjunto de programas, a fim de avaliar o desempenho relativo de um objeto, normalmente

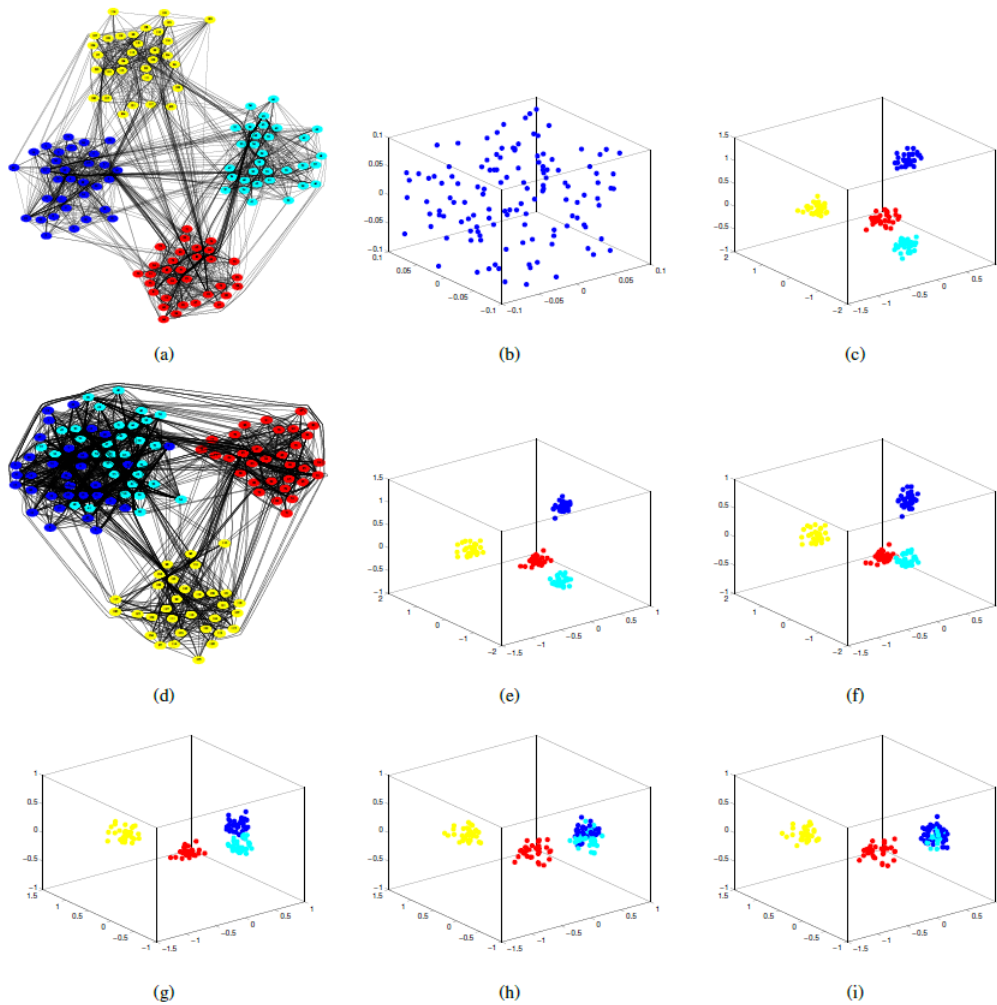


Figura 3.6 - Passos do algoritmo apresentado em (QUILES et al., 2013). (a) Rede inicial. (b) Inserção das partículas em posições aleatórias. (c) Estado de equilíbrio e interpretação dos conglomerados em comunidades. (d) Nova estrutura de rede, com 200 arestas adicionadas entre as comunidades representadas pelas cores azul claro e azul escuro. (e)-(h) Movimentações das partículas. (i) Novo estado de equilíbrio. Fonte: Figura extraída de Quiles et al. (2013).



executando uma série de testes padrões e ensaios de execução nele (WIKIPEDIA, 2013). Testar a performance de um algoritmo significa aplicá-lo a um problema específico cuja solução é bem conhecida e comparar tal solução de referência à fornecida pelo algoritmo. Em detecção de comunidades, um problema com uma solução bem definida é um grafo com uma clara estrutura em comunidades (FORTUNATO, 2010).

Nesta seção serão mostradas algumas das técnicas mais usadas de geração de *benchmarks* para detecção de comunidades em redes complexas. Essas técnicas são baseadas no modelo chamado *planted l-partition* (CONDON; KARP, 2001). Nesse modelo, uma partição de um grafo de tamanho  $n = n_{C_i}l$  é gerada, onde  $n_{C_i}$  é o número de vértices dentro de cada partição e  $l$  é o número de partições do grafo. Cada nó nesse modelo tem probabilidade  $p_{in}$  de ser conectado com outros nós do mesmo grupo, e probabilidade  $p_{out}$  de ser conectado com nós de outros grupos (LANCICHINETTI; FORTUNATO, 2009b).

Cada subgrafo desse modelo, correspondente a um dos grupos de nós, se comporta como um grafo randômico ER, descrito na Seção 2.2.1, onde  $p = p_{in}$ . O grau médio de um vértice é  $\langle k \rangle = p_{in}(n_{C_i} - 1) + p_{out}n_{C_i}(l - 1)$ , e enquanto  $p_{in} \geq p_{out}$  cada grupo será uma comunidade, pois a densidade de arestas dentro dos grupos será maior que a densidade do grafo.

Nas próximas seções serão apresentados vários *benchmarks* encontrados na literatura para detecção de comunidades. Nota-se que apesar de ser usado o termo “*benchmark*”, o que será descrito na verdade são técnicas de geração de classes de redes. Ou seja, não existe um pacote de redes pré-definidas a serem usadas como *benchmark*, mas metodologias para geração de classes específicas de redes que serão usadas na avaliação de desempenho dos algoritmos.

O objetivo é que o algoritmo consiga identificar corretamente as comunidades numa rede que tenha um comportamento bem conhecido, principalmente no que diz respeito ao grau dos vértices e tamanho das comunidades. Mas não necessariamente todas as redes que serão investigadas pelos algoritmos precisam ser exatamente iguais.

Primeiramente, será introduzido o *benchmark* mais famoso para detecção de comunidades, conhecido como *benchmark* GN, e suas variações. Logo depois, serão mostrados *benchmarks* que utilizam uma lei de potência na escolha dos graus dos vértices, dentre eles o *benchmark* LFR e suas variações.

É importante notar que as metodologias ditas acima geram sempre redes estáticas. Até a escrita dessa dissertação não havia trabalho publicado que propusesse uma metodologia



para a geração de redes dinâmicas com propósitos de *benchmark*. A metodologia que será apresentada no Capítulo 4 visa preencher essa lacuna, propondo funções que possam imitar o comportamento de uma rede dinâmica ao longo do tempo, e, assim, possam ser investigadas por algoritmos de detecção de comunidades para redes dinâmicas a fim de comparar suas performances.

### 3.3.1 *Benchmark* GN

O *benchmark* GN foi apresentado pela primeira vez em (GIRVAN; NEWMAN, 2002), e é chamado assim por causa das iniciais dos seus autores. Foi criado para poder testar um algoritmo de detecção que também foi apresentado nesse artigo. É baseado no modelo *planted l-partition*, onde  $n_{C_i}$  é fixado em 32, e  $l$  em 4, dando um total de 128 vértices. O grau médio dos vértices é  $\langle k \rangle = 16$ , dividido com uma probabilidade  $p_{in}$  de se conectar com vértices da mesma comunidade e probabilidade  $p_{out}$  de se conectar com vértices de outra comunidade.

É comum adotar a notação  $z_{in} = p_{in}(n_{C_i} - 1) = 31p_{in}$  e  $z_{out} = p_{out}n_{C_i}(l - 1) = 96p_{out}$ , que indicam o grau interno e externo esperado de um vértice, respectivamente (NEWMAN; GIRVAN, 2004). Espera-se que um algoritmo consiga identificar satisfatoriamente as comunidades enquanto  $z_{out} < 12$ . Caso  $z_{out} \geq 12$ , a rede já não tem mais uma estrutura em comunidades (LANCICHINETTI; FORTUNATO, 2009b).

Em (FAN et al., 2007), o *benchmark* GN é adaptado para casos de redes ponderadas. A rede é idêntica à do *benchmark* original, mas cada aresta tem um valor  $w$  associado, sendo  $w_{in}$  para arestas que ligam vértices de uma mesma comunidade e  $w_{out}$  para arestas que ligam comunidades diferentes. Nos experimentos realizados no artigo,  $w_{in}$  é fixado em 1, enquanto o valor de  $w_{out}$  é investigado.

Quando a rede tem  $z_{out} = 4$  os algoritmos conseguem identificar bem as comunidades, mesmo quando o valor de  $w_{out}$  é maior que 1. Isso porque apesar do peso maior, as comunidades estão bem separadas. Já quando a rede tem  $z_{out} = 8$ , o valor de  $w_{out}$  é crucial na identificação das comunidades, sendo que os algoritmos conseguem identificá-las bem até um valor aproximado  $w_{out} \approx 0.6$ . Quando assume valores maiores, o desempenho dos algoritmos começa a decair. Num terceiro experimento, o valor de  $w_{out}$  é fixado em 0.2 enquanto o valor  $z_{out}$  é variado, e o resultado é que os algoritmos têm um desempenho satisfatório até valores altos de  $z_{out}$ , aproximadamente 10, mostrando que os pesos nas arestas alteram significadamente o desempenho dos algoritmos.

Em (ARENAS et al., 2006), uma generalização do *benchmark* GN é proposta. O modelo

contém dois níveis hierárquicos. Ou seja, existe um primeiro nível de comunidades e um segundo nível de comunidades. No primeiro nível os vértices são divididos entre os grupos, e no segundo nível os grupos do primeiro nível são organizados em grupos maiores.

Em vez de 128 nós, como no *benchmark* original, este contém 256 nós, sendo que no primeiro nível eles são divididos em 16 grupos e no segundo nível em 4 grupos que contêm 4 dos grupos do primeiro nível. Cada vértice mantém um grau médio interno  $z_{in_1} + z_{in_2} + z_{out} = 18$ , onde  $z_{in_1}$  é o grau interno no primeiro nível e  $z_{in_2}$  é o grau interno nos segundo nível.

### 3.3.2 *Benchmark LFR*

O *benchmark* GN, apresentado na Seção 3.3.1, foi por muito tempo o mais utilizado para comparações e avaliações de algoritmos de detecção. No entanto, existem ressalvas a serem consideradas. Nesse *benchmark* todos os vértices têm aproximadamente o mesmo grau, e todas as comunidades têm o mesmo tamanho. Ou seja, as redes criadas com essa metodologia não podem ser um parâmetro para redes reais (LANCICHINETTI et al., 2008; DANON et al., 2006).

A maioria das redes reais são caracterizadas por terem distribuição heterogênea dos graus dos vértices, cuja curva decai como uma lei de potência. Da mesma forma, não é correto assumir que todas as comunidades têm o mesmo tamanho. A distribuição do tamanho das comunidades também pode ser aproximada por uma lei de potência em muitos casos (DANON et al., 2006). Experimentos com redes com comunidades de tamanhos diferentes são feitos em (DANON et al., 2006). A Figura 3.7 mostra um exemplo de uma rede com comunidades de diferentes tamanhos.

Em (BAGROW, 2008) é apresentado um algoritmo para criação de redes com distribuição de graus como uma lei de potência. A partir de um grafo construído sob as regras do modelo BA, descrito na Seção 2.2.1, os vértices são aleatoriamente divididos em dois ou mais grupos, os quais serão as comunidades ao final do algoritmo. Depois de divididos os vértices, uma técnica de troca é usada para que as arestas sejam rearrumadas e uma estrutura em comunidades surja na rede.

A técnica de troca consiste em escolher duas arestas,  $(i, j)$  e  $(k, l)$ , e substituí-las pelas arestas  $(i, k)$  e  $(j, l)$ . Isso faz com que os graus dos vértices  $i, j, k$  e  $l$  permaneçam os mesmos após a troca. Sendo assim, os vértices são escolhidos de forma que  $i$  e  $k$  pertençam ao mesmo grupo de nós, assim como  $j$  e  $l$ , garantindo com que as novas arestas sejam *intra-cluster*. Assim, a estrutura em comunidades é formada e a distribuição de graus

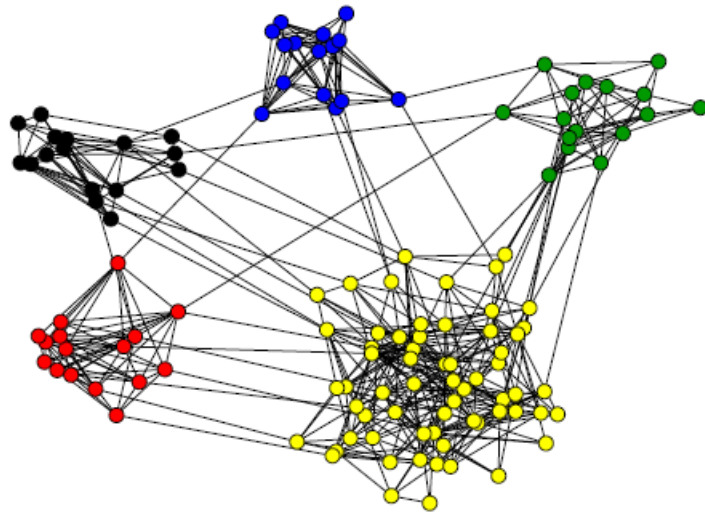


Figura 3.7 - Exemplo de rede com comunidades de diferentes tamanhos. Fonte: Figura extraída de Danon et al. (2006).

segue o modelo BA, ou seja, uma lei de potência.

Em 2008, uma metodologia que é capaz de gerar redes com comunidades de tamanhos diferentes e também com distribuição de graus que segue uma lei de potência foi apresentada (LANCICHINETTI et al., 2008). A metodologia, chamada *benchmark LFR*, assume que a distribuição de graus e a distribuição de tamanho das comunidades seguem uma lei de potência com expoentes  $\tau_1$  e  $\tau_2$ , respectivamente.

Os parâmetros para a geração de uma rede são: o número total de vértices na rede ( $n$ ), o grau médio desejado  $\langle k \rangle$  e, por conseguinte, os extremos  $k_{min}$  e  $k_{max}$ ; e um valor  $\mu$ , que controla a distribuição do grau dos vértices entre internos e externos. Ou seja, definido o valor de  $\mu$ , a fração aproximada de arestas de um vértice que se conectam com vértices de outras comunidades é  $\mu$ , e a fração aproximada de arestas de um vértice que se conectam com vértices da mesma comunidade é  $1 - \mu$ . Esse parâmetro é chamado *mixing parameter*. Por fim, o tamanho mínimo e máximo das comunidades,  $s_{min}$  e  $s_{max}$ , que deve obedecer as seguintes restrições:  $s_{min} > k_{min}$  e  $s_{max} > k_{max}$ . Isso garante que qualquer que seja o grau do vértice, ele poderá ser incluído em pelo menos uma comunidade. A soma dos tamanhos de todas as comunidades deve ser igual a  $n$  e devem obedecer uma lei de potência com expoente  $\tau_2$  (LANCICHINETTI; FORTUNATO, 2009b).

Uma vez que os parâmetros foram estabelecidos, é possível gerar uma rede. Para isso, é criada uma rede com  $n$  vértices, sendo que a distribuição de graus foi retirada de uma lei

de potência com expoente  $\tau_1$ . No começo, todos os vértices estão sem uma comunidade definida. Então, enquanto existir um vértice sem comunidade, este é atribuído aleatoriamente a alguma comunidade. Caso o tamanho da comunidade escolhida seja menor que o grau do vértice, este permanece sem comunidade e outro vértice é escolhido. Caso a comunidade escolhida já esteja totalmente preenchida, ou seja, a quantidade de vértices atribuída a essa comunidade exceda o seu tamanho, um outro vértice da comunidade é escolhido aleatoriamente e se torna novamente sem comunidade. Esse procedimento é repetido até que todos os vértices estejam atribuídos a alguma comunidade (LANCICHINETTI et al., 2008).

Depois de todos os vértices estarem atribuídos a uma comunidade, as arestas devem ser rearrumadas para que o valor do *mixing parameter*  $\mu$  seja respeitado. Cada vértice  $i$  tem um grau  $k_i$  atribuído, esse valor é a soma do grau interno  $k_i^{in} = (1 - \mu)k_i$  e do grau externo  $k_i^{out} = \mu k_i$  do vértice, que a princípio não retrata a realidade, já que as comunidades foram definidas após a criação da rede. Então, a técnica de troca é usada nesse passo para que as arestas dos vértices sejam corretamente divididas entre arestas internas e externas à comunidade. A Figura 3.3.2 ilustra um exemplo de rede gerada pelo *benchmark* LFR.

Em 2009, os autores publicaram uma extensão ao *benchmark* LFR (LANCICHINETTI; FORTUNATO, 2009a). Essa extensão adiciona opções de se criar redes com comunidades sobrepostas, ou seja, existe a possibilidade de alguns vértices pertencerem simultaneamente a mais de uma comunidade, além de redes direcionadas e ponderadas. Todos os pacotes com as diferentes possibilidades de redes do *benchmark* LFR podem ser obtidos para testes e uso em (FORTUNATO, 2013).

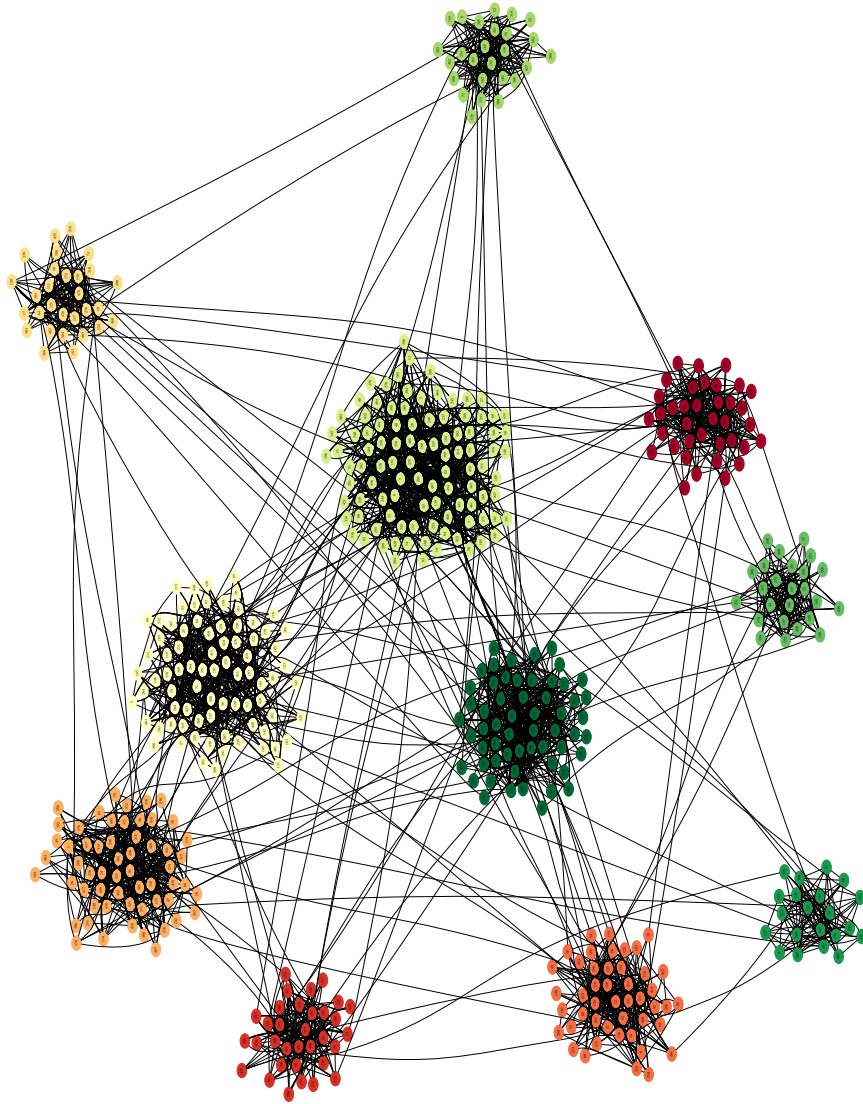


Figura 3.8 - Figura criada a partir da rede gerada utilizando o código fonte do *benchmark LFR* disponível em (FORTUNATO, 2013).



## 4 METODOLOGIA

Neste capítulo será apresentada a metodologia desenvolvida neste trabalho. O objetivo é possibilitar a criação de uma estrutura que simule o comportamento de uma rede complexa dinâmica. Para que essa simulação seja bem sucedida é preciso ter definido a característica que se deseja explorar. No caso deste trabalho, criar redes dinâmicas que possam ser utilizadas como *benchmark* para algoritmos de detecção de comunidades em redes complexas é o objetivo. Ou seja, a rede a ser criada precisa ter uma estrutura em comunidades e conseguir manter essa estrutura apesar das alterações que a rede possa sofrer ao longo do tempo.

Alguns estudos da área das redes sociais investigaram o comportamento das comunidades em redes dinâmicas. Em (LIN et al., 2007), *blogs e bloggers* são acompanhados e a evolução das comunidades formadas são analisadas através do tempo. Em (SCHLITTER; FALKOWSKI, 2009), as evoluções das comunidades são investigadas num aplicativo de música (Last.fm<sup>1</sup>). Em (PALLA et al., 2007), análises de redes de publicações científicas e ligações entre telefones móveis também são analisadas.

Os três trabalhos citados acima observam a formação de novas comunidades, as modificações nas comunidades existentes, assim como as movimentações mais complexas (PAPADOPOULOS et al., 2012). Comparando os resultados entre esses três trabalhos, percebe-se a formação de um consenso a respeito das possíveis transformações que as comunidades podem sofrer. Apesar de nominadas de forma diferente, há uma concordância quanto ao número de possíveis transformações assim como à forma como elas acontecem.

Basicamente há três tipos de transformações, ilustradas na Figura 4.1: (a) um-para-um, que envolve o crescimento ou contração da comunidade; (b) um-para-muitos e muitos-para-um, que envolve uma comunidade se dividindo em outras, ou duas ou mais comunidades unindo-se em uma; (c) um-para-zero e zero-para-um, que envolve o nascimento ou extinção de uma comunidade. Nesse trabalho, essas seis possíveis transformações serão chamadas de *born*, *extinction*, *growth*, *contraction*, *merge* e *split*.

A função *born* tratará do nascimento de uma nova comunidade. A função *extinction* tratará da extinção de uma comunidade existente na rede. A função *growth* irá provocar o crescimento de uma comunidade já existente na rede. A função *contraction* irá provocar a contração de uma determinada comunidade. Por fim, a função *merge* irá lidar com a união de duas ou mais comunidades em uma, enquanto que a função *split* irá tratar da divisão de uma comunidade em duas ou mais comunidades.

---

<sup>1</sup><http://www.lastfm.com.br/>

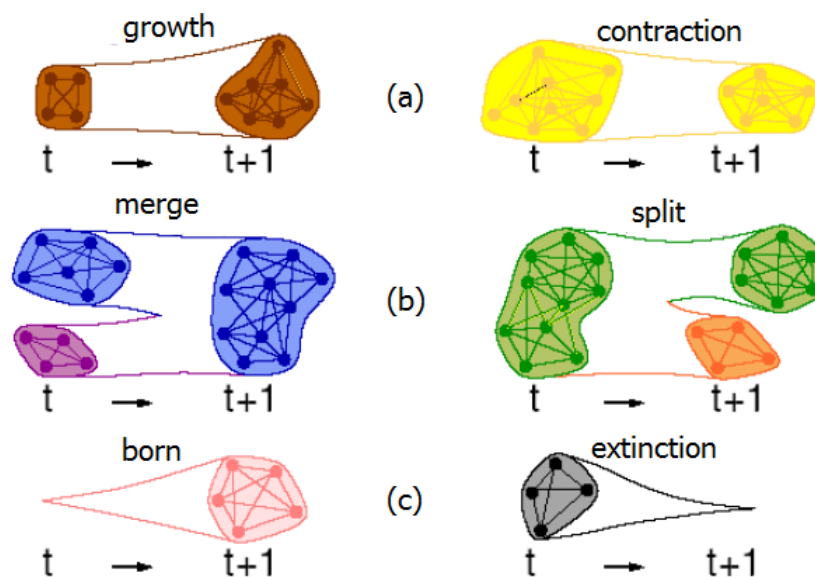


Figura 4.1 - Possíveis transformações que uma comunidade pode sofrer numa rede complexa dinâmica. Fonte: Figura extraída de Palla et al. (2007).

Identificadas as possíveis transformações que uma comunidade pode sofrer durante sua evolução numa rede dinâmica, a metodologia propõe seis algoritmos que simulam esses comportamentos. Partindo de uma rede inicial, construída utilizando o algoritmo do *benchmark* LFR, apresentada na Seção 3.3.2, as transformações podem ser aplicadas sequencialmente, sem sobreposição, formando uma lista de grafos (*snapshots*) que representam a evolução da rede ao longo do tempo. A cada ação das funções uma cópia do estado atual do grafo é armazenada numa lista chamada *grafos* que serve como uma “linha do tempo” das transformações sofridas pela rede e será o parâmetro de saída de todas as funções.

Nas próximas seções, os algoritmos serão detalhados, os parâmetros de entrada serão explicados, um pseudo-código de cada um deles será apresentado e algumas imagens da função em funcionamento serão mostradas. A metodologia foi construída utilizando a biblioteca *igraph* (CSARDI; NEPUSZ, 2006), nas linguagens R (R Core Team, 2012) e C (RITCHIE, 1993).

#### 4.1 Função Born

O objetivo da função *born* é, dada uma rede inicial, fazer surgir uma nova comunidade na rede. Os parâmetros de entrada são um grafo inicial  $g$ , o tamanho mínimo de comunidade  $nmin$ , que é opcional, e caso não seja definido assume o valor  $minsizecomu$  usado no algoritmo LFR. O tamanho máximo de comunidade  $nmax$ , também opcional, e assume



o valor *maxsizecomu* usado no algoritmo LFR caso não seja definido. E, por último, o *mixing parameter*  $\mu$ , que caso não seja definido, pois é um parâmetro opcional, assume o valor *mixing* usado no algoritmo LFR.

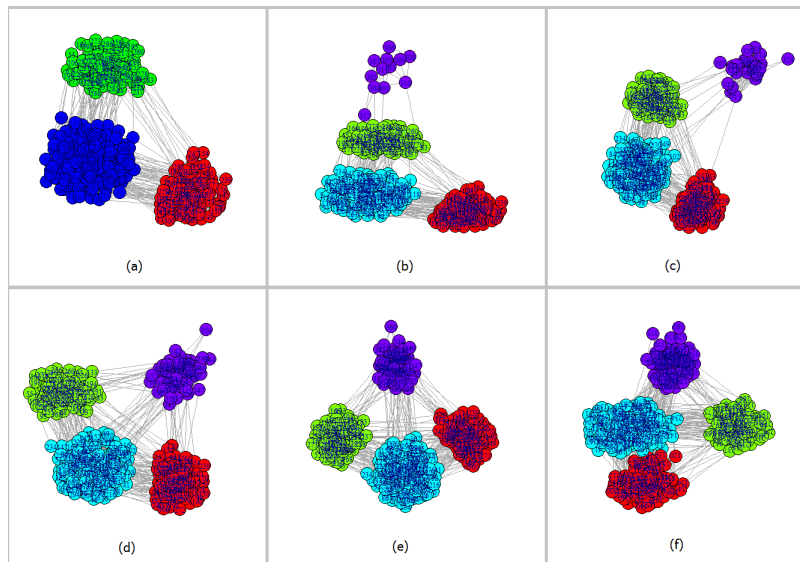


Figura 4.2 - Passos da função *born*. A comunidade sofrendo a transformação é a que está colorida de roxo.

O Algoritmo 4.1.1 mostra o pseudo-código da função *born*. A primeira ação é definir o tamanho e o índice da nova comunidade. Feito isso, cada novo vértice é adicionado ao grafo, e a cada vértice novas arestas são adicionadas de acordo com o grau escolhido. A cada aresta adicionada, há duas opções: *in* ou *out*, *in* para arestas internas e *out* para arestas externas. O parâmetro  $\mu$  é responsável pela distribuição entre internas e externas. Finalmente, depois de todos os vértices adicionados, arestas são adicionadas internamente à comunidade com o objetivo de aumentar a densidade da comunidade até que ela se iguale à densidade média das outras comunidades da rede<sup>1</sup>.

A Figura 4.2 mostra alguns passos da função agindo sobre uma rede. Em (b) é possível observar os passos iniciais, quando os vértices ainda não estão bem conectados, inclusive um dos vértices está mais próximo da comunidade que está representada pela cor verde. Em (c) e (d) constata-se os momentos iniciais de um novo vértice na comunidade, quando há poucas arestas o ligando. Por fim, em (f) os momentos finais da função quando arestas estão sendo adicionadas com o objetivo de aumentar a densidade da nova comunidade.

<sup>1</sup>A função *densidade* do Algoritmo 4.1.1, assim como dos outros algoritmos apresentados neste capítulo, é uma função definida localmente que calcula a média das densidades das comunidades da rede, e não a densidade do grafo como um todo.

## 4.2 Função Extinction

O objetivo da função *extinction*, dada uma rede inicial, é extinguir uma das comunidades da rede. Os parâmetros de entrada são um grafo inicial  $g$  e um índice opcional  $comu$ , que indica qual a comunidade deverá ser extinta. Caso esse valor seja zero, uma comunidade será escolhida aleatoriamente.

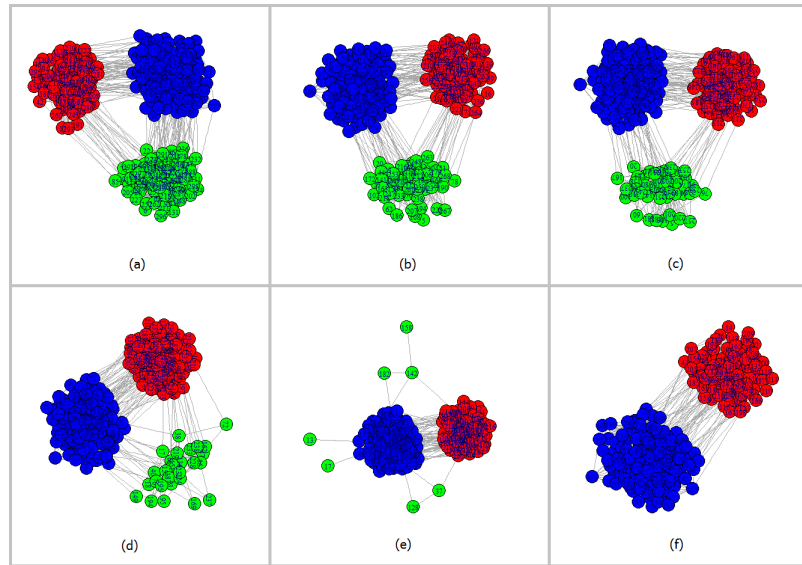


Figura 4.3 - Passos da função *extinction*. A comunidade sofrendo a transformação é a que está colorida de verde.

O Algoritmo 4.2.1 mostra o pseudo-código da função *extinction*. A primeira ação é definir qual a comunidade que irá ser extinta de acordo com o parâmetro  $comu$ . Feito isso, cada vértice da comunidade é deletado da rede numa ordem aleatória, até que não exista mais nenhum vértice pertencente à comunidade  $idcomu$ .

Na Figura 4.3 é possível observar o comportamento da rede durante a transformação. Em (b), (c) e (d) já é perceptível uma menor densidade da comunidade de cor verde, que é a que está sofrendo a ação. Em (e) verifica-se uma divisão da comunidade em partes, devido à exclusão das arestas que os ligam, e em alguns casos os vértices podem até se tornar isolados. Por fim, em (f) nota-se a exclusão completa da comunidade que sofreu a transformação.

## 4.3 Função Growth

O objetivo da função *growth*, dada uma rede inicial, é provocar o crescimento de uma das comunidades da rede. Os parâmetros de entrada são um grafo inicial  $g$ , um índice

opcional *comu*, que indica qual a comunidade que deverá sofrer a transformação. Caso esse valor seja zero, uma comunidade será escolhida aleatoriamente. O *mixing parameter*  $\mu$ , caso não seja definido, pois é um parâmetro opcional, assume o valor *mixing* usado no algoritmo LFR. E, por último, o tamanho máximo de comunidade *nmax*, também opcional e assume o valor *maxsizecomu* usado no algoritmo LFR, caso não seja definido.

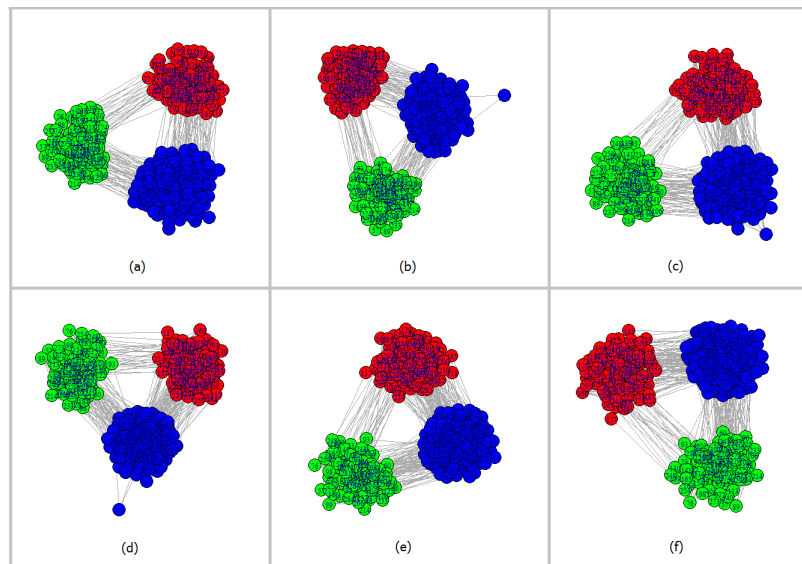


Figura 4.4 - Passos da função *growth*. A comunidade sofrendo a transformação é a que está colorida de azul.

O Algoritmo 4.3.1 mostra o pseudo-código da função *growth*. A primeira ação é definir qual a comunidade que irá sofrer a transformação de acordo com a o parâmetro *comu*. Feito isso, um novo tamanho para a comunidade é escolhido. Novos vértices são adicionados à comunidade até que o novo tamanho seja atingido. A cada novo vértice, arestas também são adicionadas de acordo com o grau escolhido para o vértice. A cada aresta adicionada, há duas opções: *in* ou *out*, *in* para arestas internas e *out* para arestas externas, sendo  $\mu$  o parâmetro responsável pela distribuição entre internas e externas. Finalmente, depois de todos os novos vértices serem adicionados, arestas vão sendo criadas internamente à comunidade com o objetivo de aumentar a densidade da comunidade até que ela se iguale à densidade média das outras comunidades da rede.

A Figura 4.4 mostra alguns *snapshots* da transformação agindo sobre a rede. Em (b), (c) e (d) os momentos iniciais de novos vértices na rede são vistos. Em (e) e (f) o aumento de arestas na rede com o objetivo de aumentar a densidade é mostrado.

#### 4.4 Função Contraction

O objetivo da função *contraction*, dada uma rede inicial, é provocar a contração de uma das comunidades da rede. Os parâmetros de entrada são um grafo inicial  $g$ , um índice opcional *comu*, que indica qual a comunidade que deverá sofrer a transformação, caso o valor seja zero, uma comunidade será escolhida aleatoriamente. E, por último, o tamanho mínimo de comunidade *nmin*, que é opcional, e caso não seja definido assume o valor *minsizecomu* usado no algoritmo LFR.

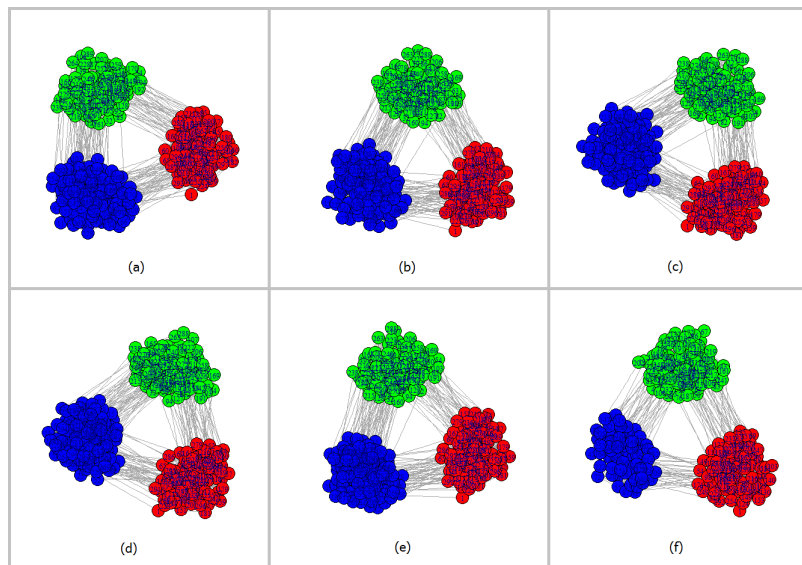


Figura 4.5 - Passos da função *contraction*. A comunidade sofrendo a transformação é a que está colorida de azul.

O Algoritmo 4.4.1 mostra o pseudo-código da função *contraction*. A primeira ação é definir qual comunidade sofrerá a transformação de acordo com o parâmetro *comu*. Feito isso, um novo tamanho para a comunidade é escolhido. Vértices são removidos da rede até que o novo tamanho seja atingido. A cada vértice removido, arestas são adicionadas internamente à comunidade com o objetivo de manter a densidade da comunidade igual à densidade média das outras comunidades da rede.

Na Figura 4.5, os passos da transformação são exibidos, onde é possível notar uma diferença na densidade da comunidade azul entre os passos (a) e (f). Há também uma diminuição no diâmetro da representação da comunidade, expondo a diminuição de fato no número de vértices da comunidade.

## 4.5 Função Merge

O objetivo da função *merge*, dada uma rede inicial, é unir duas ou mais comunidades em apenas uma. Os parâmetros de entrada são um grafo inicial  $g$ , um vetor opcional  $v$  de comunidades, que indica quais comunidade deverão ser unidas, caso o vetor esteja vazio, a quantidade de comunidades e quais comunidade serão unidas serão escolhidos aleatoriamente. Por fim, duas probabilidades  $probR$  e  $probA$ , opcionais e que a soma deve ser igual a 1. As probabilidades estão relacionadas com as ações possíveis para que as comunidades sejam unidas. É possível adicionar novas arestas à comunidade, ou arestas internas às antigas comunidades podem ser redirecionadas para serem uma ligação entre elas. Caso os valores das probabilidades não sejam definidos, eles assumem o valor 0.5.

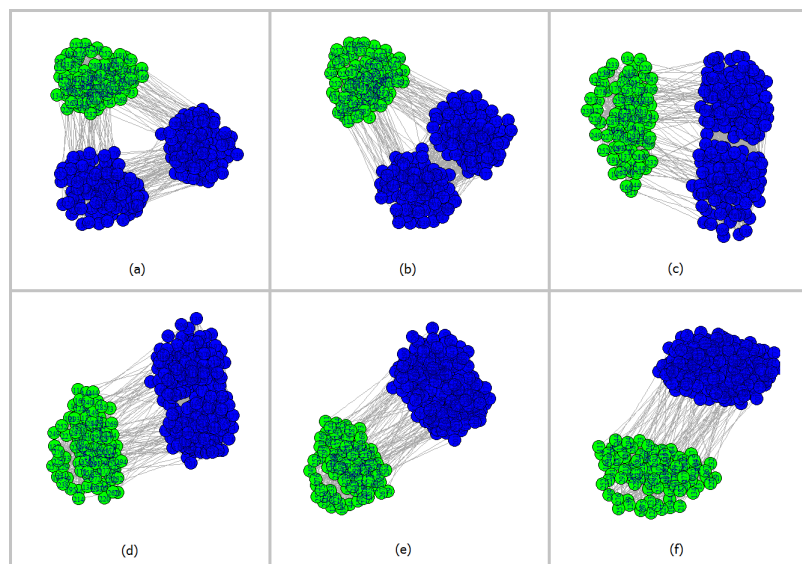


Figura 4.6 - Passos da função *merge*. As comunidades sofrendo a transformação são as que estão coloridas de azul.

O algoritmo 4.5.1 mostra o pseudo-código da função *merge*. A primeira ação é definir a quantidade e quais comunidades serão unidas de acordo com o vetor  $v$ . Depois, um novo índice de comunidade escolhido aleatoriamente dentre os índices do vetor  $v_{comu}$  é atribuído a todos os vértices das antigas comunidades. Então uma condição de parada é definida,  $R_t$  é o total de possíveis arestas que podem ser redirecionadas e  $D_t$  é o valor médio da densidade das outras comunidades da rede, os dois valores são influenciados pelos valores de  $probR$  e  $probA$ . Enquanto a condição de parada não for satisfeita, um tipo de ação, redirecionar ou adicionar, é escolhido. Caso o tipo escolhido seja adicionar, uma aresta interna é adicionada. Caso o tipo escolhido seja redirecionar, a técnica de troca, explicada na Seção 3.3.2, é usada.

Alguns passos da função podem ser observados na Figura 4.6. Em (a) apesar da comunidade colorida de azul ser considerada única, é claro para o observador que há na verdade dois grupos de vértices. Em (b), (c) e (d), arestas estão sendo adicionadas entre os grupos dispersos da comunidade, e a aproximação destes pode ser verificada. Por fim, em (e) e (f) a união da comunidade é atingida.

#### 4.6 Função Split

O objetivo da função *split*, dada uma rede inicial, é dividir uma das comunidades em duas ou mais comunidades. Os parâmetros de entrada são um grafo inicial  $g$ , um índice opcional *comu*, que indica qual a comunidade que será dividida, caso o valor seja zero, uma comunidade será escolhida aleatoriamente. Um valor  $x$  que indica em quantas partes a comunidade será dividida, esse valor é opcional e caso não seja definido é escolhido aleatoriamente. Duas probabilidades *probR* e *probD*, opcionais, e que a soma deve ser igual a 1. As probabilidades estão relacionadas com as possíveis ações para que a comunidade seja dividida. É possível deletar arestas entre as novas comunidades ou redirecionar arestas que estão entre as comunidades para se tornarem arestas internas as novas comunidades. Caso os valores das probabilidades não sejam definidos, eles assumem o valor 0.5. E, por fim, o *mixing parameter*  $\mu$ , que caso não seja definido, pois é um parâmetro opcional, assume o valor *mixing* usado no algoritmo LFR.

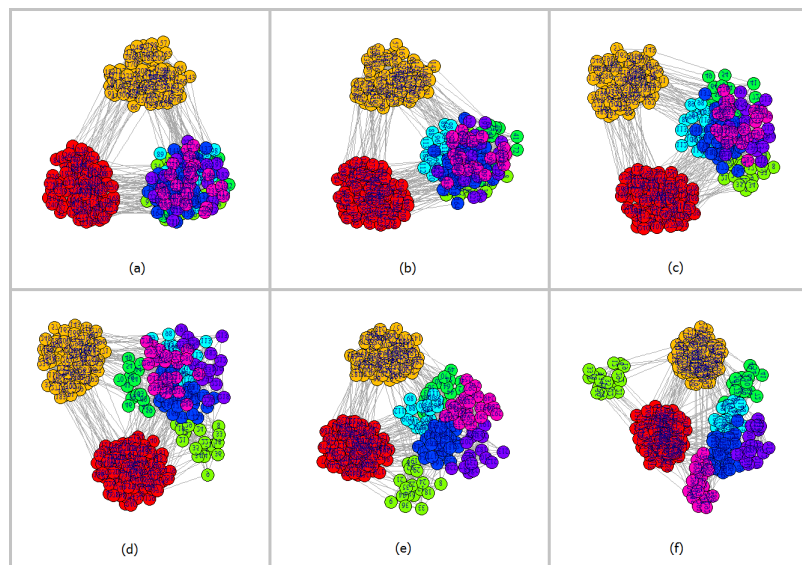


Figura 4.7 - Passos da função *split*. As comunidades sofrendo a transformação são as que estão coloridas de verde claro, verde, roxo, rosa, azul e azul claro.

O Algoritmo 4.6.1 mostra o pseudo-código da função *split*. A primeira ação é definir

qual a comunidade que irá sofrer a divisão de acordo com o parâmetro *comu*. Depois, é escolhido em quantos pedaços a comunidade irá se dividir de acordo com o parâmetro  $x$ . Feito isso, novos índices são criados, e os vértices são atribuídos as  $x$  novas comunidades de forma aleatória. A condição de parada é então definida, esta compara a proporção de arestas inter-comunidade com o valor do *mixing parameter*  $\mu$ . Enquanto a condição de parada não é atingida, um tipo de ação, deletar ou redirecionar, é escolhido. Caso o tipo escolhido seja deletar, uma aresta inter-comunidade é deletada. Caso o tipo escolhido seja redirecionar, a técnica de troca, explicada na Seção 3.3.2, é usada.

A Figura 4.7 mostra alguns passos da transformação agindo sobre a rede. Em (a) é possível perceber como os vértices são separados em pedaços de forma aleatória. Em (b), (c), (d) e (e) testemunha-se o processo de separação dos pedaços, que ocorre de forma lenta e pontual. Em (e) já é verificável a separação completa das partes, mas ainda muito ligados por arestas inter-comunidades. Por fim, em (f), as novas comunidades estão definidas.

---

**Algoritmo 4.1.1:** Função Born

---

**Input** : graph  $g$ **Output:** a list of graphs

```
1 begin
2    $tamcomu$  = valor entre  $nmin$  e  $nmax$ ;
3    $idcomu$  = valor do índice da nova comunidade;
4    $grafos$  = {};
5   for  $i \leftarrow 1$  to  $tamcomu$  do
6     if  $i == 1$  then
7       Adicionar novo vértice  $v1$  ao grafo  $g$ ;
8       Atribuir vértice  $v1$  a comunidade  $idcomu$ ;
9        $grafos = grafos + g$ ;
10      Escolher vértice  $v2$  qualquer do grafo  $g$ ;
11      Adicionar aresta  $(v1, v2)$ ;
12       $grafos = grafos + g$ ;
13    else
14      Adicionar novo vértice  $v1$  ao grafo  $g$ ;
15      Atribuir vértice  $v1$  a comunidade  $idcomu$ ;
16       $grafos = grafos + g$ ;
17       $grau$  = valor entre 2 e  $i$ ;
18      for  $j \leftarrow 1$  to  $grau$  do
19         $conexao$  = in ou out, de acordo com o valor de  $\mu$ ;
20        if  $conexao == out$  then
21          Escolher um vértice  $v2$  qualquer que não pertence a comunidade
22           $idcomu$ ;
23          Adicionar aresta  $(v1, v2)$  ao grafo  $g$ ;
24           $grafos = grafos + g$ ;
25        else
26          if  $conexao == in$  then
27            Escolher um vértice  $v2$  qualquer que pertence a comunidade
28             $idcomu$ ;
29            Adicionar aresta  $(v1, v2)$  ao grafo  $g$ ;
30             $grafos = grafos + g$ ;
31      while  $densidade(comunidade\ idcomu) < densidade(g)$  do
32        Escolher dois vértices  $v1$  e  $v2$  pertencentes a comunidade  $idcomu$ ;
33        Adicionar aresta  $(v1, v2)$  ao grafo  $g$ ;
34         $grafos = grafos + g$ ;
35  return ( $grafos$ );
```

---



---

**Algoritmo 4.2.1:** Função Extinction

---

**Input** : grafo inicial  $g$   
comunidade  $comu = 0$

**Output:** lista de grafos

```
1 begin
2    $grafos = \{\}$ ;
3   if  $comu == 0$  then
4     |  $idcomu =$  comunidade aleatória do grafo  $g$ ;
5   else
6     |  $idcomu = comu$ ;
7    $tamcomu =$  tamanho da comunidade  $idcomu$ ;
8   while  $tamcomu > 0$  do
9     | Escolher vértice  $v$  aleatório dentro da comunidade  $idcomu$ ;
10    | Deletar vértice  $v$ ;
11    |  $grafos = grafos + g$ ;
12    |  $tamcomu =$  novo tamanho da comunidade  $idcomu$ ;
13  return ( $grafos$ );
```

---

---

**Algoritmo 4.3.1:** Função Growth

---

**Input** : grafo inicial  $g$   
comunidade  $comu = 0$   
mixing parameter  $\mu = mixing$   
tamanho máximo de comunidade  $nmax = maxsizecomu$

**Output:** lista de grafos

```
1 begin
2    $grafos = \{\}$ ;
3   if  $comu == 0$  then
4     |  $idcomu =$  comunidade aleatória do grafo  $g$ ;
5   else
6     |  $idcomu = comu$ ;
7    $tamcomuinicial =$  tamanho da comunidade  $idcomu$ ;
8    $tamcomufinal =$  valor aleatório entre  $tamcomuinicial$  e  $nmax$ ;
9    $qntnovosvertices = tamcomufinal - tamcomuinicial$ ;
10  for  $i \leftarrow 1$  to  $qntnovosvertices$  do
11    | Adicionar novo vértice  $v1$  ao grafo  $g$ ;
12    | Atribuir  $v1$  a comunidade  $idcomu$ ;
13    |  $grafos = grafos + g$ ;
14    |  $grau =$  valor aleatório entre 2 e  $tamcomuinicial + i$ ;
15    | for  $j \leftarrow 1$  to  $grau$  do
16      |  $tipo = in$  ou  $out$  de acordo com  $\mu$ ;
17      | if  $conexao == out$  then
18        | Escolher vértice aleatório  $v2$  que não pertence a comunidade  $idcomu$ ;
19        | Adicionar aresta  $(v1, v2)$  ao grafo  $g$ ;
20        |  $grafos = grafos + g$ ;
21      | else
22        | if  $conexao == in$  then
23          | Escolher vértice aleatório  $v2$  que pertence a comunidade  $idcomu$ ;
24          | Adicionar aresta  $(v1, v2)$  ao grafo  $g$ ;
25          |  $grafos = grafos + g$ ;
26    | while  $densidade(comunidade\ idcomu) < densidade(g)$  do
27      | Escolher dois vértices  $v1$  e  $v2$  pertencentes a comunidade  $idcomu$ ;
28      | Adicionar aresta  $(v1, v2)$  ao grafo  $g$ ;
29      |  $grafos = grafos + g$ ;
30  return  $(grafos)$ ;
```

---

---

**Algoritmo 4.4.1:** Função Contraction

---

**Input** : grafo inicial  $g$ comunidade  $comu = 0$ tamanho mínimo de comunidade  $nmin = minsizecomu$ **Output:** lista de grafos

```
1 begin
2    $grafos = \{\}$ ;
3   if  $comu == 0$  then
4     |  $idcomu =$  comunidade aleatória do grafo  $g$ ;
5   else
6     |  $idcomu = comu$ ;
7    $tamcomuinicial =$  tamanho da comunidade  $idcomu$ ;
8    $tamcomufinal =$  valor aleatório entre  $nmin$  e  $tamcomuinicial$ ;
9    $qntvelhosvertices = tamcomuinicial = tamcomufinal$ ;
10  for  $i \leftarrow 1$  to  $qntvelhosvertices$  do
11    | Escolher vértice  $v$  aleatório dentro da comunidade  $idcomu$ ;
12    | Deletar vértice  $v$ ;
13    |  $grafos = grafos + g$ ;
14    | while  $densidade(comunidade\ idcomu) < densidade(g)$  do
15      | Escolher dois vértices  $v1$  e  $v2$  pertencentes a comunidade  $idcomu$ ;
16      | Adicionar aresta  $(v1, v2)$  ao grafo  $g$ ;
17      |  $grafos = grafos + g$ ;
18  return  $(grafos)$ ;
```

---

---

**Algoritmo 4.5.1: Função Merge**

---

**Input** : grafo inicial  $g$ vetor de comunidades  $v = \{\}$ probabilidade de redirecionar  $probR = 0.5$ probabilidade de adicionar  $probA = 0.5$ // A soma das probabilidades deve ser 1,  $probR + probA = 1$ **Output:** lista de grafos

```
1 begin
2    $grafos = \{\}$ ;
3   if  $v$  for vazio then
4      $ncomu =$  número de comunidades a serem unidas, escolhido aleatoriamente;
5      $vcomu = ncomu$  comunidades escolhidas aleatoriamente;
6   else
7      $ncomu =$  tamanho de  $v$ ;
8      $vcomu = v$ ;
9    $idcomu =$  valor escolhido aleatoriamente dentre os valores de  $vcomu$ ;
10  Todos os vértices pertencentes as comunidades de  $vcomu$  são atribuídas a
    comunidade  $idcomu$ ;
11   $R_t = probR * (\text{número total de arestas dentro da comunidade } idcomu)$ ;
12   $D_t = probA * \text{densidade}(g)$ ;
13   $r = 0$ ;
14   $d = \text{densidade}(\text{comunidade } idcomu)$ ;
15   $parada = (r \leq R_t) \text{ and } (d \leq D_t)$ ;
16  while  $\neg parada$  do
17     $tipo =$  add ou red, de acordo com  $probR$  e  $probA$ ;
18    if  $tipo == add$  then
19      Escolher dois vértices  $v1$  e  $v2$  pertencentes a comunidade  $idcomu$ ;
20      Adicionar aresta  $(v1, v2)$  ao grafo  $g$ ;
21       $grafos = grafos + g$ ;
22       $d = \text{densidade}(\text{comunidade } idcomu)$ ;
23    else
24      if  $tipo == red$  then
25        Escolher duas arestas  $(v1, v2)$  e  $(v3, v4)$  dentro da comunidade  $idcomu$ ;
        // Aplicar técnica de troca;
26        Deletar as arestas  $(v1, v2)$  e  $(v3, v4)$ ;
27        Adicionar as arestas  $(v1, v3)$  e  $(v2, v4)$ ;
28         $grafos = grafos + g$ ;
29         $r = r + 1$ ;
30     $parada = (r \leq R_t) \text{ and } (d \leq D_t)$ ;
31  return  $(grafos)$ ;
```

---

---

**Algoritmo 4.6.1:** Função Split

---

**Input** : grafo inicial  $g$ comunidade  $comu = 0$ pedaços  $x = 0$ probabilidade de redirecionar  $probR = 0.5$ probabilidade de deletar  $probD = 0.5$ mixing parameter  $\mu = mixing$ // A soma das probabilidades deve ser 1,  $probR + probD = 1$ **Output:** lista de grafos

```
1 begin
2    $grafos = \{\}$ ;
3   if  $comu == 0$  then
4     |  $idcomu =$  comunidade aleatória do grafo  $g$ ;
5   else
6     |  $idcomu = comu$ ;
7   if  $x < 2$  then
8     |  $xcomu =$  valor entre 2 e o valor máximo tal que todas as novas comunidades
9     | tenham pelo menos 3 vértices;
10  else
11    |  $xcomu = x$ ;
12   $A_t =$  número total de arestas da comunidade  $idcomu$ ;
13  Criar novos índices  $\{idcomu_1, idcomu_2, \dots, idcomu_{xcomu}\}$  que representam as novas
14  comunidades;
15  Cada vértice da comunidade  $idcomu$  é atribuído a uma nova comunidade de forma
16  aleatória;
17   $A_c =$  número total de arestas que ligam comunidades diferentes;
18   $parada = A_c/A_t < \mu$ ;
19  while  $!parada$  do
20    |  $tipo =$  red ou del, de acordo com  $probD$  e  $probR$ ;
21    | if  $tipo == del$  then
22      | Escolher duas comunidades  $idcomu_x$  e  $idcomu_y$ ;
23      | Deletar uma aresta aleatória que liga as comunidades  $idcomu_x$  e  $idcomu_y$ ;
24      |  $grafos = grafos + g$ ;
25    | else
26      | if  $tipo == red$  then
27        | Escolher duas comunidades  $idcomu_x$  e  $idcomu_y$ ;
28        | // Aplicar técnica de troca;
29        | Deletar duas arestas que ligam as  $idcomu_x$  e  $idcomu_y$ ,  $(v1_x, v1_y)$  e
30        |  $(v2_x, v2_y)$ ;
31        | Adicionar duas arestas  $(v1_x, v2_x)$  e  $(v1_y, v2_y)$  ao grafo  $g$ ;
32        |  $grafos = grafos + g$ ;
33    | Calcular novo valor de  $A_c$ ;
34    |  $parada = A_c/A_t < \mu$ ;
35  return  $(grafos)$ ;
```

---



## 5 EXPERIMENTOS E RESULTADOS

Neste capítulo serão mostrados os experimentos realizados com a metodologia proposta. Especificamente, cinco tipos de experimentos foram realizados. No primeiro, cada uma das funções foi analisada individualmente, e nos outros quatro as funções foram aplicadas sucessivamente. Parâmetros distintos foram utilizados em cada conjunto de simulações com o objetivo de investigar suas influências no resultado final da detecção. O objetivo dos experimentos é verificar como as diferentes transformações influenciam a detecção de comunidades.

Como dito no Capítulo 4, o parâmetro de saída das funções é uma lista de grafos que forma uma linha do tempo das transformações ocorridas na rede. Cada item dessa lista é um *snapshot* do estado da rede naquele intervalo de tempo. O tamanho dessa lista ao final da função indica a quantidade de passos necessários para que cada uma das transformações seja concluída.

Para avaliar a influência das transformações na detecção de comunidades, foi usada a abordagem de aplicação longitudinal em sucessivos *snapshots*, explicada na Seção 3.2.5. Os algoritmos utilizados foram o Fast Greedy (Seção 3.2.1), Infomap (Seção 3.2.2), Label Propagation (Seção 3.2.3) e Walktrap (Seção 3.2.4). A Informação Mútua Normalizada (Seção 3.1.2.2) é calculada entre as comunidades na rede, e as comunidades encontradas pelos algoritmos. Adicionalmente, gráficos que relacionam o valor da Informação Mútua Normalizada e os passos da transformação são apresentados.

Os parâmetros de entrada, que são utilizadas tanto no algoritmo LFR para a criação do grafo inicial como em algumas funções da metodologia, são: número inicial de vértices  $n$ , grau médio  $\langle k \rangle$ , grau máximo  $k_{max}$ , *mixing parameter*, os expoentes  $\tau_1$  e  $\tau_2$  que controlam a distribuição de graus e a distribuição do tamanho das comunidades, respectivamente. Por fim, os tamanhos máximo e mínimo de comunidade, *minsizecomu* e *maxsizecomu*.

Nas próximas seções os experimentos serão explicados com mais detalhes. Os parâmetros de entrada utilizados serão indicados e gráficos de desempenho dos algoritmos serão mostrados.

### 5.1 Experimentos Individuais

Nos experimentos desta seção, cada função da metodologia foi analisada individualmente. Os parâmetros de entrada, mostrados na Tabela 5.1, foram utilizados por todos os experimentos. Tais parâmetros foram empiricamente ajustados para que as redes geradas fossem pequenas e com uma estrutura de comunidades bem definida, assim o foco dos resultados

estaria na influência das transformações. As próximas figuras mostram os gráficos gerados para cada função. O tamanho do eixo  $x$  de cada gráfico depende da quantidade de passos necessários para executar a transformação. As legendas indicam qual cor representa cada algoritmo usado. O verde representa o Fast Greedy, o vermelho representa o Infomap, o azul claro representa o Label Propagation e o roxo representa o Walktrap. Em alguns gráficos não é possível ver uma ou mais linhas, pois os valores se sobrepõem em alguns casos.

Tabela 5.1 - Parâmetros de entrada usados nos Experimentos Individuais.

<b>Parâmetro</b>	<b>Valor</b>
$n$	300
$\langle k \rangle$	30
$k_{max}$	60
$mixing$	0.05
$\tau_1$	2
$\tau_2$	1
$minsizecomu$	50
$maxsizecomu$	100

Na Figura 5.1 os valores da informação mútua se mantêm altos para todos os algoritmos. As pequenas quedas observadas ocorrem quando um novo vértice é adicionado à rede, e os algoritmos o consideram pertencente a uma nova comunidade, mas logo depois o novo vértice é classificado corretamente conforme as arestas vão sendo adicionadas. Como apresentado na Figura 4.2(c) do capítulo anterior, é possível ver um caso onde um vértice pode ser mal-interpretado, pois ainda não está adequadamente conectado com a sua comunidade.

Na Figura 5.2, vemos o gráfico gerado para a função *extinction*. A detecção funciona perfeitamente do começo até quase o final quando a detecção tem algumas falhas. Isso ocorre pois conforme os vértices vão sendo removidos, a comunidade pode acabar se dividindo em vértices isolados ou com poucas ligações com outros vértices. Na Figura 4.3(e) é possível verificar esse evento.

Na Figura 5.3, é possível observar o gráfico gerado para a função *growth*. Os algoritmos Walktrap e Infomap têm desempenho impecável durante a transformação, enquanto o Fast Greedy e o Label Propagation têm pequenas falhas, que ocorrem quando um novo vértice é adicionado, assim como na função *born*.



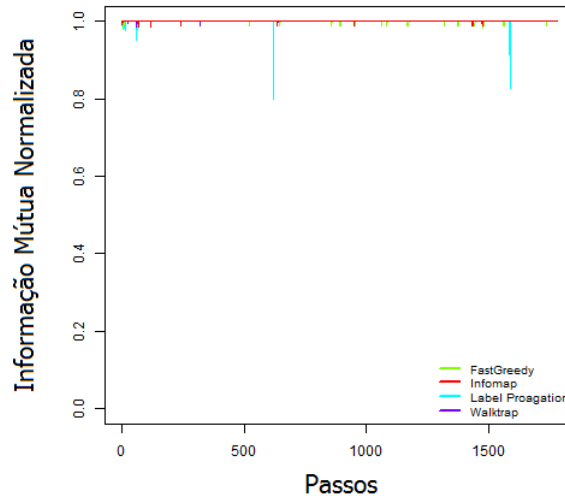


Figura 5.1 - Gráfico gerado com os resultados obtidos nos Experimentos Individuais da função *born*.

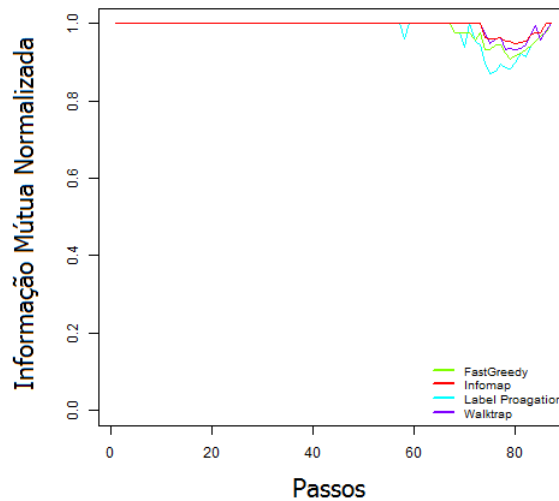


Figura 5.2 - Gráfico gerado com os resultados obtidos nos Experimentos Individuais da função *extinction*.

O gráfico gerado para a função *contraction* é observado na Figura 5.4. Excluindo-se o algoritmo Fast Greedy nas iterações iniciais, o valor da informação mútua se mantém máximo para o resto dos passos. Verificando o porquê do Fast Greedy ter mal-detectado alguns vértices, percebe-se que isso ocorre quando algum vértice com um grau muito alto é excluído, a densidade da comunidade cai um pouco e o algoritmo considera haver outra comunidade. Contudo, ao passo que novas arestas vão sendo adicionadas, o algoritmo volta a detectar as comunidades perfeitamente.

O resultado do processo de detecção associado a função *merge* é apresentado na Figura

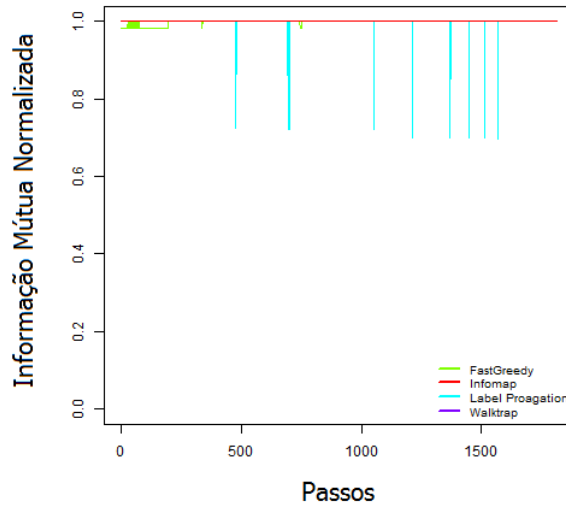


Figura 5.3 - Gráfico gerado com os resultados obtidos nos Experimentos Individuais da função *growth*.

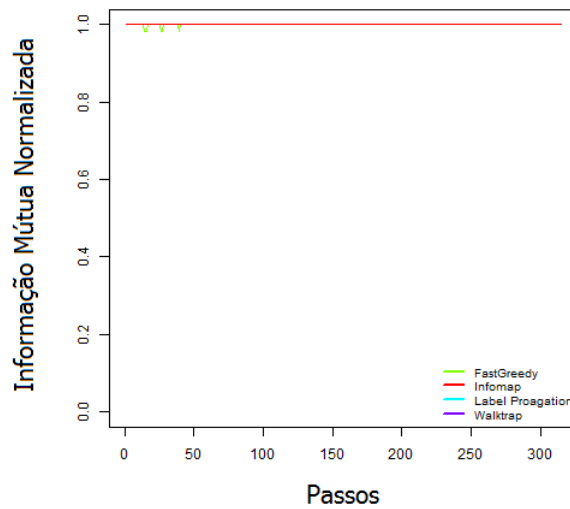


Figura 5.4 - Gráfico gerado com os resultados obtidos nos Experimentos Individuais da função *contraction*.

5.5. Somente os algoritmos Label Propagation e Infomap detectam as comunidade perfeitamente, mas a diferença entre o desempenho desses dois algoritmos é notável. Enquanto o Infomap é preciso nos resultados, o Label Propagation é bastante inconstante, só mantendo o resultado já nos últimos passos. Tanto o Walktrap, quanto o Fast Greedy não identificaram a junção das comunidades. O ponto exato em que o Infomap considerou a junção das comunidades está ilustrada na Figura 4.6(d).

Por fim, na Figura 5.6 pode-se conferir o gráfico gerado com os resultados obtidos pela função *split*. Os algoritmos depois de algum tempo conseguem identificar as comunida-

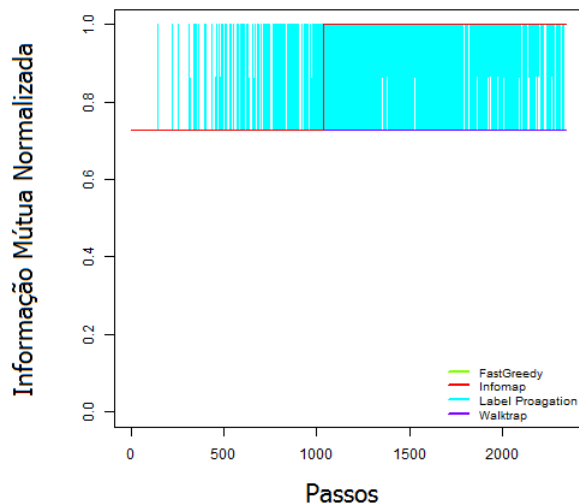


Figura 5.5 - Gráfico gerado com os resultados obtidos nos Experimentos Individuais da função *merge*.

des corretamente. Mas somente nos últimos passos, precisamente na iteração 1834, que pode ser observada na Figura 4.7(f), é que todos os algoritmos conseguem identificar as comunidades perfeitamente.

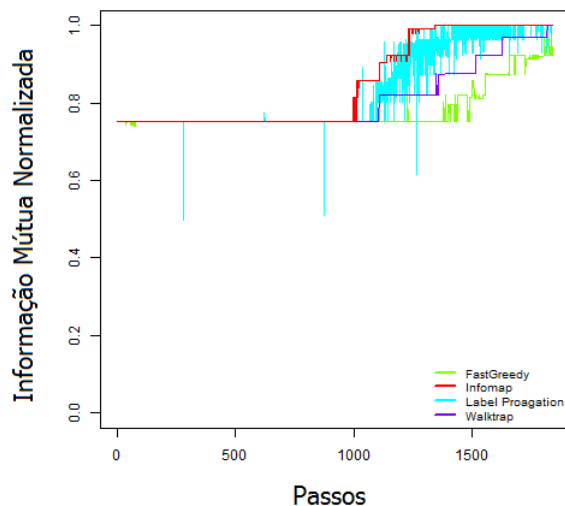


Figura 5.6 - Gráfico gerado com os resultados obtidos nos Experimentos Individuais da função *split*.

Esse experimento mostra que as funções da metodologia preservam a estrutura em comunidades apesar das transformações. Somente na função *merge* os algoritmos não identificaram todas as comunidades perfeitamente, mas será visto na próxima seção que isso não ocorre para todos os casos.

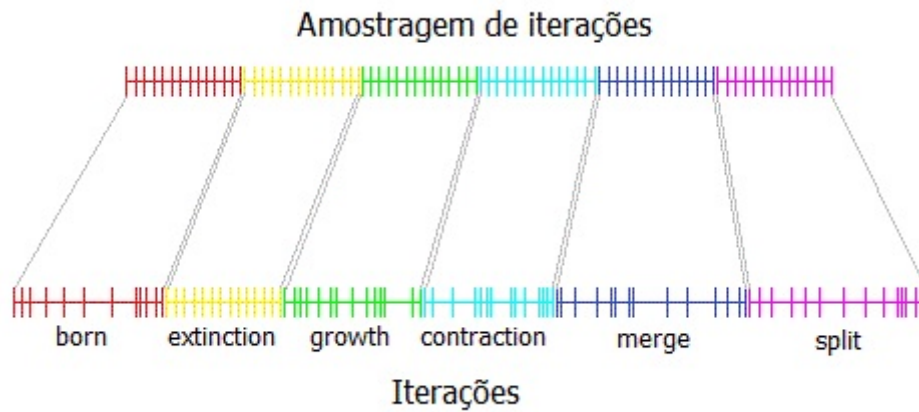


Figura 5.7 - Exemplo de uma amostragem de iterações.

## 5.2 Experimentos Completos

Nos experimentos dessa seção, as funções da metodologia são aplicadas sucessivamente na seguinte ordem: *born*, *extinction*, *growth*, *contraction*, *merge* e *split*. As próximas figuras irão ter divisão de “setores” por cor, que indicam qual a função que está atuando em cada passo. A cor vermelha indica a função *born*, a cor amarela indica a função *extinction*, a cor verde indica a função *growth*, a cor azul claro indica a função *contraction*, a cor azul indica a função *merge*, e por último, a cor magenta indica a função *split*.

Como visto na Seção 5.1, cada transformação precisa de uma quantidade diferente de passos para concluir suas transformações. Nos Experimentos Completos uma quantidade fixa de passos foi escolhida dentre todas as iterações. Para cada um dos experimentos a função que precisou de menos passos foi verificada, e essa mesma quantidade de passos foi escolhida entre os passos das outras funções, incluindo o primeiro e o último passo. A Figura 5.7 exemplifica como seria essa amostragem de passos, ou iterações.

Quatro experimentos completos são realizados, a diferença entre eles são os valores dos parâmetros de entrada. Dos oito possíveis, como indicados na Tabela 5.1, seis permaneceram fixos, com os valores indicados por essa mesma tabela. São eles:  $\tau_1$ ,  $\tau_2$ ,  $\langle k \rangle$ ,  $k_{max}$ ,  $minsizecomu$  e  $maxsizecomu$ . Os valores dos expoentes não foram alterados porque estudos quanto a influência desse dois parâmetros já foram realizados em (LANCICHINETTI et al., 2008), e os valores 1 e 2 foram usados respectivamente em (LANCICHINETTI; FORTUNATO, 2009b). Os valores de tamanho mínimo e máximo de comunidade foram mantidos fixos para que uma rede com poucas comunidades grandes ou uma rede com muitas comunidades pequenas fosse influenciado somente pelo valor do parâmetro  $n$ . Por fim, os

valores de  $\langle k \rangle$  e  $k_{max}$  foram escolhidos de forma que estivessem adequados para o tamanho das comunidades.

### 5.2.1 Experimento Completo 1

Os parâmetros de entrada usados nesse experimento foram  $n = 300$  e  $mixing = 0.05$ . A intenção é que a rede formada tivesse poucas comunidades mas bem separadas. Por isso valores pequenos para  $n$  e para o  $mixing$ .

A Figura 5.8 mostra os gráficos gerados no Experimento Completo 1. Todos os algoritmos mantiveram valores altos de informação mútua, porém somente os algoritmos Infomap e Label Propagation conseguiram atingir o valor máximo ao fim de cada uma das transformações. A transformação com maior impacto no valor da informação mútua foi a *merge*, indicada pela cor azul, em todos os algoritmos.

O desempenho entre os algoritmos pode ser considerado parecido. O algoritmo com a maior falha foi o Label Propagation durante a função *merge*, e o algoritmo que se recuperou mais rápido na função *merge* foi o Fast Greedy, contrariando os resultados obtidos nos Experimentos Individuais, indicados pela Figura 5.5.

### 5.2.2 Experimento Completo 2

Os parâmetros de entrada usados nesse experimento foram  $n = 300$  e  $mixing = 0.2$ . A intenção é que a rede formada tivesse poucas comunidades mas não tão bem separadas. Por isso, um valor pequeno para  $n$  e um valor maior para o  $mixing$ .

A Figura 5.9 mostra os gráficos gerados no Experimento Completo 2. Esse experimento mostra como o valor do  $mixing$  influencia bastante os resultados obtidos. É possível perceber também o quanto o algoritmos Label Propagation é inconstante, chegando em alguns passos a obter valor mínimo. Novamente as funções *merge* e *split* são as que provocam as maiores variações no valor da informação mútua, considerando que são as transformações mais complexas.

O algoritmos com melhor desempenho nesse experimento foi o Infomap, que obtém os maiores valores durante as transformações. No entanto, o algoritmo não consegue obter valor máximo na última transformação. A função *split* tem como condição de parada uma regra definida a partir do  $mixing$ , como pode ser observado no Algoritmo 4.6.1, e um valor alto influencia diretamente no desempenho da função, causando a má detecção das comunidades ao fim da transformação.

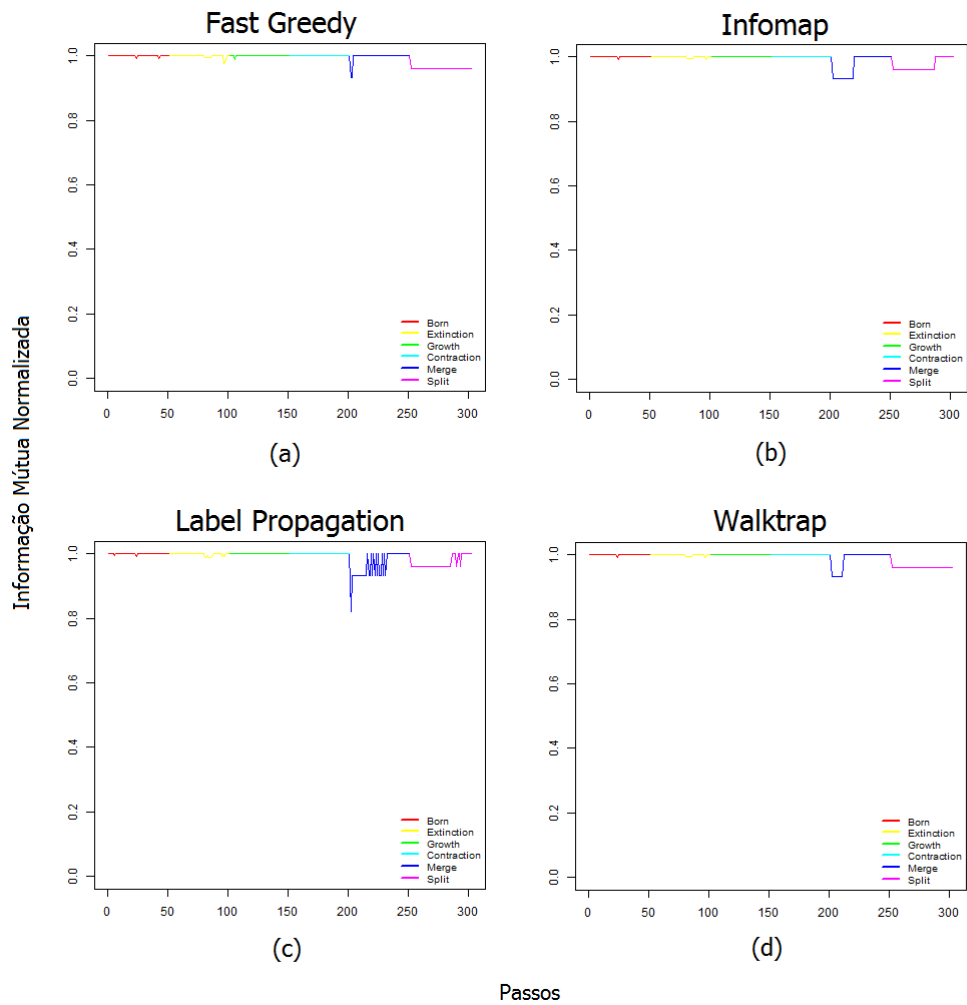


Figura 5.8 - Gráficos gerados com os resultados obtidos pelo Experimento Completo 1.

### 5.2.3 Experimento Completo 3

Os parâmetros de entrada usados nesse experimento foram  $n = 500$  e  $mixing = 0.05$ . A intenção nesse experimento é que a rede formada tivesse mais comunidades do que nos Experimentos Completos 1 e 2 e bem separadas. Por isso, um valor maior para  $n$  e um valor pequeno para o  $mixing$ .

A Figura 5.10 mostra os gráficos gerados no Experimento Completo 3. Os algoritmos voltam a ter um bom desempenho na detecção das comunidades, apesar de existirem mais comunidade, considerando o valor maior de  $n$ . Novamente, somente os algoritmos Infomap e Label Propagation conseguiram chegar ao valor máximo ao final das transformações.

Todos os algoritmos tiveram desempenhos bastante parecidos, assim como no Experimento Completo 1, a diferença foi que o Fast Greedy se recuperou mais rápido na função

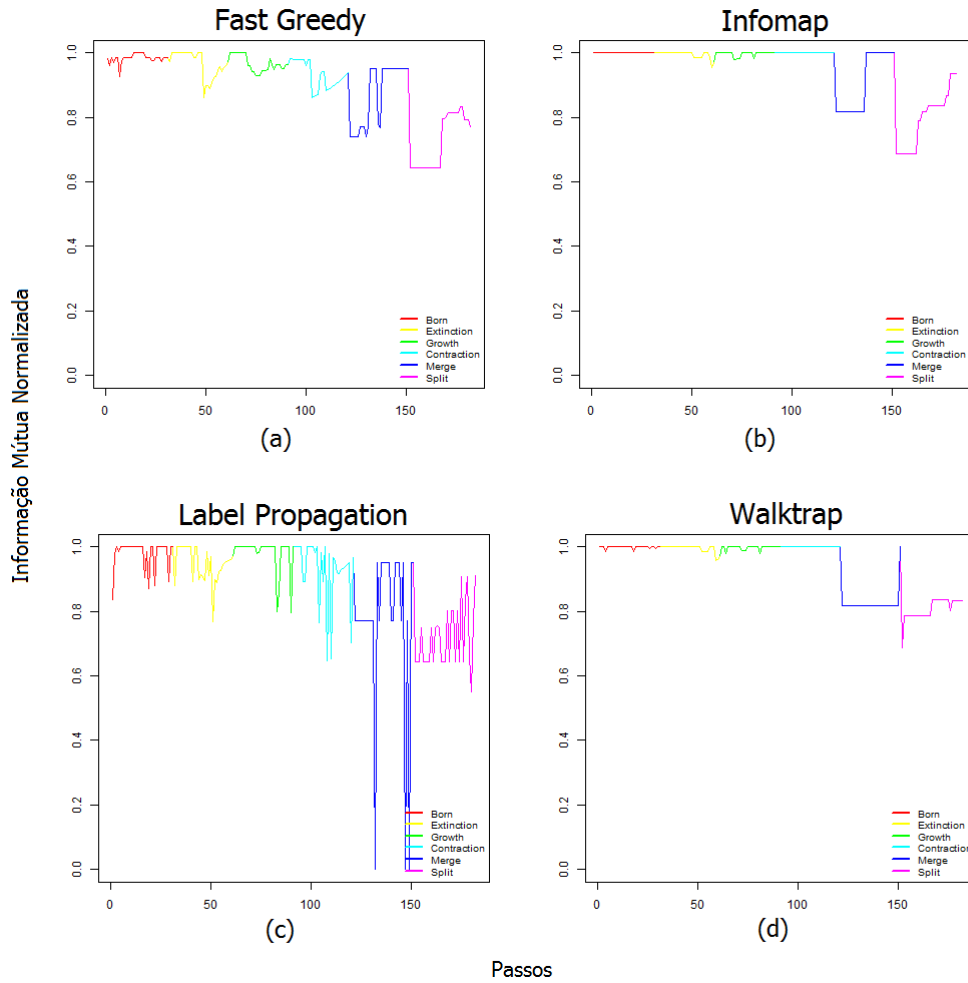


Figura 5.9 - Gráficos gerados com os resultados obtidos pelo Experimento Completo 2.

*merge*, contrariando novamente o resultado obtido nos Experimentos Individuais, indicado na Figura 5.5.

#### 5.2.4 Experimento Completo 4

Os parâmetros de entrada usados nesse experimento foram  $n = 500$  e  $mixing = 0.2$ . A intenção nesse experimento é que a rede formada tivesse mais comunidades que nos Experimentos Completos 1 e 2, porém com uma maior proporção de ligações inter-comunidades. Por isso um valor maior para  $n$  e para o *mixing*.

A Figura 5.11 mostra os gráficos gerados no Experimento Completo 4. Os algoritmos têm um bom desempenho na detecção das comunidades, apesar do valor maior do *mixing*, isso porque como existem mais comunidades nessa rede, um má interpretação de alguns vértices não influencia tanto no valor da informação mútua.

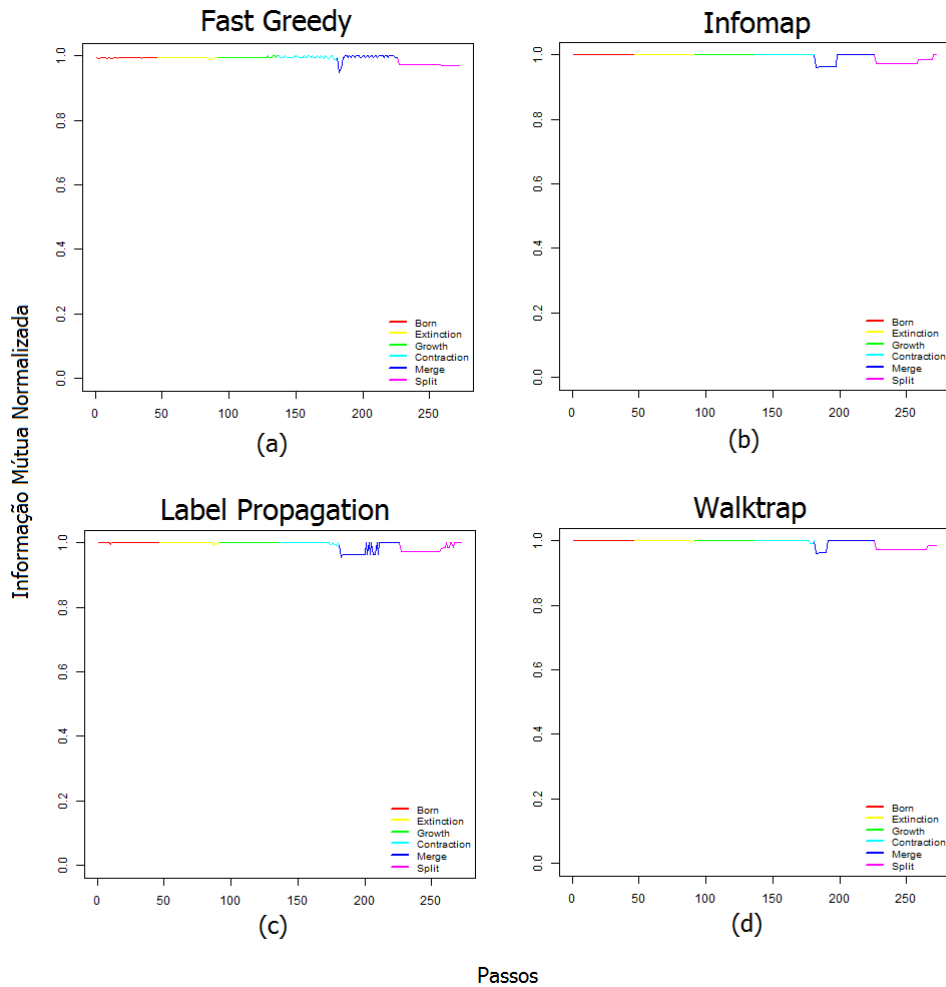


Figura 5.10 - Gráficos gerados com os resultados obtidos pelo Experimento Completo 3.

Nenhum algoritmo consegue atingir o valor máximo da informação mútua depois das transformações *merge* e *split*. O algoritmo Label Propagation ainda consegue atingir o valor máximo durante a função *merge*, mas não mantém o resultado ao final da transformação. Os algoritmos que conseguem atingir os maiores valores são o Label Propagation e Walktrap ao final de todas as transformações.

### 5.3 Conclusões dos Experimentos

Dentre todos os experimentos, individuais e completos, somente em alguns casos isolados a detecção não teve performance satisfatória. Dentre os algoritmos, o que manteve melhor desempenho médio geral foi o Infomap. O algoritmo Label Propagation também obteve bons resultados, porém o algoritmo apresenta uma maior inconsistência, ou seja, nem sempre o resultado obtido é o melhor possível. Os algoritmos Fast Greedy e Walktrap não tiveram desempenhos tão bons quanto o Infomap, mas não necessariamente tiveram



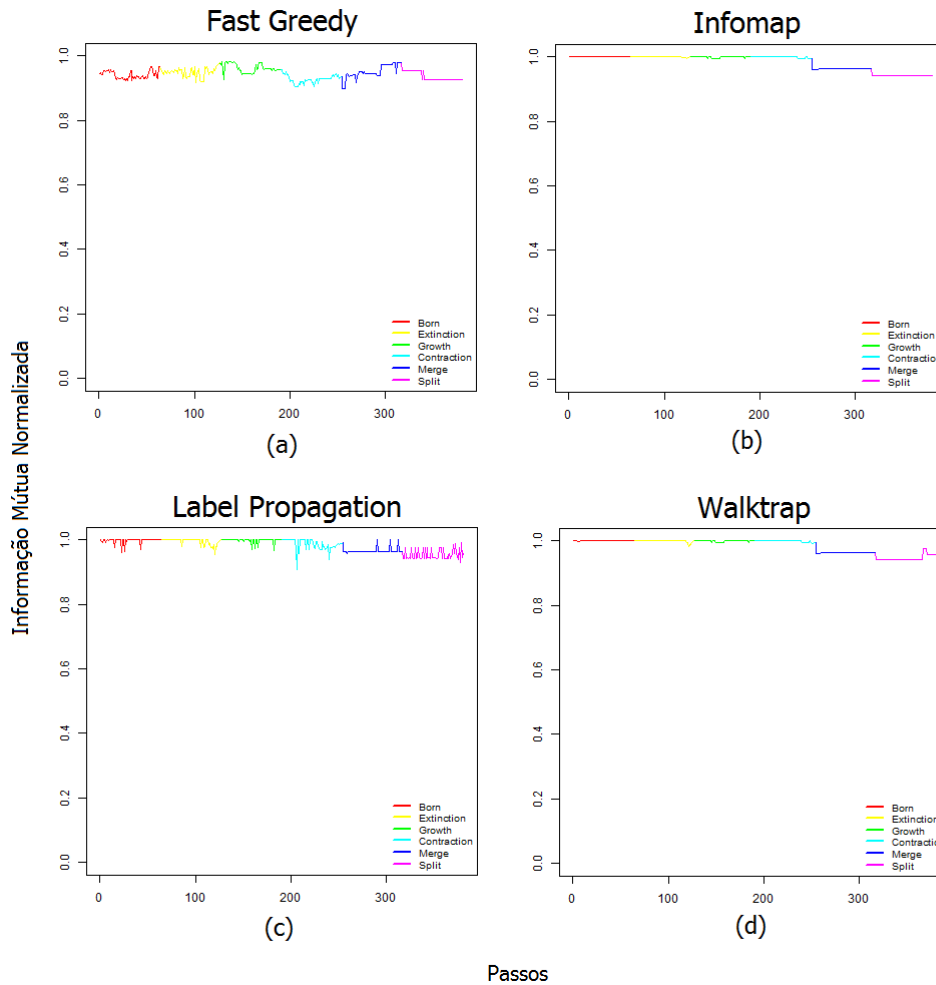


Figura 5.11 - Gráficos gerados com os resultados obtidos pelo Experimento Completo 4.

resultados ruins. Eles poderiam ser usados de forma satisfatória para a detecção.

As funções mais complexas e portanto mais difíceis de serem acompanhadas pelos algoritmos são a *merge* e a *split*. As transformações causadas por essas funções são profundas na rede, tanto que elas também precisam das maiores quantidades de passos para serem concluídas.

As Tabelas 5.2 e 5.3 mostram algumas estatísticas do desempenho dos algoritmos. Na Tabela 5.2 a média dos valores da informação mútua de cada algoritmos para cada experimento é calculada. O algoritmo Infomap tem a média maior em todos os experimentos individuais, mas no Experimento Completo 1, o Fast Greedy têm valor médio maior, e nos Experimentos Completos 3 e 4, a maior média é a do algoritmo Walktrap. Mesmo assim, todos os algoritmos mantêm valores muito parecidos, ou seja, a metodologia consegue manter uma estrutura de comunidade comprovada por quatro algoritmos com abordagens

Tabela 5.2 - Valores médios dos desempenhos dos algoritmos de acordo com os experimentos.

<b>Experimento/Algoritmo</b>	<b>Fast Greedy</b>	<b>Infomap</b>	<b>Label Propagation</b>	<b>Walktrap</b>
EI Born	0.9995623	0.9999320	0.9996172	0.9999205
EI Extinction	0.9891155	0.9946571	0.9845096	0.9935877
EI Growth	0.9983162	1.0000000	0.9985566	1.0000000
EI Contraction	0.9996723	1.0000000	1.0000000	1.0000000
EI Merge	0.7270617	0.8794663	0.8426301	0.7271089
EI Split	0.7804254	0.8506009	0.8330535	0.8084530
Completo 1	0.9925355	0.9912771	0.9894647	0.9909344
Completo 2	0.9069037	0.9484890	0.8853985	0.9366107
Completo 3	0.9919887	0.9943170	0.9933527	0.9946506
Completo 4	0.9449553	0.9843080	0.9828750	0.9848928

Tabela 5.3 - Porcentagem de vezes que um algoritmo atingiu o valor máximo nos experimentos.

<b>Experimento/Algoritmo</b>	<b>Fast Greedy</b>	<b>Infomap</b>	<b>Label Propagation</b>	<b>Walktrap</b>
EI Born	88%	91%	88%	78%
EI Extinction	70%	69%	71%	70%
EI Growth	87%	92%	96%	95%
EI Contraction	97%	99%	100%	98%
EI Merge	0%	55%	42%	0%
EI Split	0%	27%	15%	1%
Completo 1	57%	69%	60%	60%
Completo 2	14%	57%	32%	45%
Completo 3	12%	60%	59%	57%
Completo 4	0%	48%	40%	47%

diferentes.

Na Tabela 5.3, a porcentagem de vezes que os algoritmos conseguem atingir o valor máximo da informação mútua é contabilizada. Os algoritmos Fast Greedy e Walktrap em quatro situações não atingem o valor máximo nenhuma vez, mas como mostrado na Tabela 5.2, não significa que as comunidades não foram detectadas satisfatoriamente. Novamente os valores mais baixos são obtidos com as funções *merge* e *split*, ratificando que essas são as funções mais complexas na rede.

As tabelas mostram que a função aparentemente mais simples é a *contraction* seguida de perto pela função *growth*. Essa constatação faz sentido pois essas funções são as que não mudam o número de comunidades dentro da rede, apenas modificam a estrutura de uma delas.

Os gráficos gerados, assim como as tabelas desta seção mostram que a metodologia funciona, pois consegue atingir o objetivo principal, manter a estrutura em comunidades apesar das transformações aplicadas à rede. Assim sendo, a metodologia pode ser usada para simular eventos em redes dinâmicas, e gerar *benchmarks* para avaliar o desempenho de algoritmos de detecção de comunidades.



## 6 CONCLUSÕES E TRABALHOS FUTUROS

A metodologia desenvolvida, descrita no Capítulo 4, consegue simular o comportamento e evolução de comunidades em Redes Complexas Dinâmicas. Além disso, a metodologia mantém a estrutura em comunidades da rede apesar das transformações sofridas. Com isso, o objetivo desse trabalho foi alcançado.

A metodologia conta com seis funções que são responsáveis por criar ou excluir vértices e arestas de tal forma que cada um dos propósitos das funções seja cumprido. A partir dos Experimentos individuais chega-se a conclusão que as transformações mais profundas ocorrem quando é preciso juntar duas ou mais comunidades em uma, ou dividir uma comunidade em um ou mais partes. Dentre as seis funções, são as que precisam de mais passos para se completarem, e as que mais afetaram o desempenho dos algoritmos. As outras quatro funções afetam muito pouco o desempenho dos algoritmos, somente em alguns momentos há uma distorção da detecção, como ao fim da função *extinction*, ou em alguns pontos das funções *born* e *growth*.

Todos os experimentos partiram de uma rede inicial gerada com o *benchmark* LFR (ver Seção 3.3.2). Em cada experimento, redes iniciais distintas foram geradas. Em outros testes realizados, percebe-se que o resultado das funções é bastante dependente das condições iniciais, ou seja, da rede LFR gerada, assim como da escolha das comunidades que serão transformadas pela seção.

Por exemplo, no Experimento Individual *extinction* foram necessários apenas 89 passos para a exclusão completa da comunidade, isso porque a comunidade escolhida para sofrer a ação tinha 89 vértices, e cada iteração excluía um dos vértices. Qualquer outra comunidade escolhida teria influenciado diretamente na quantidade de passos para que a função fosse completada. Além disso, a ordem com que os vértices vão sendo excluídos na comunidade afetam diretamente o desempenho dos algoritmos de detecção. Caso existisse uma regra onde os vértices seriam excluídos a partir de uma ordem crescente dos graus, por exemplo, o desempenho do algoritmo seria um, já se a ordem fosse decrescente, perceberíamos logo nos primeiros passos que a comunidade acabaria se dividindo em vértices isolados, já que os vértices que tem função central na comunidade seriam apagados logo no começo.

Em todas as outras funções, observa-se a influência das condições iniciais. O tamanho da nova comunidade influencia no número de passos da função *born*. A quantidade de novos vértices e o tamanho da comunidade escolhida na função *growth* determinam se será necessário a criação de muitas arestas para que a comunidade fique densa o suficiente. Na

função *contraction*, o número de iterações necessárias também é diretamente dependente da quantidade de vértices a serem excluídos e das arestas que terão que ser adicionadas para manter a densidade.

Assim como nas funções *merge* e *split*, o tamanho das comunidades escolhidas, e a ordem das arestas que serão redirecionadas, adicionadas ou excluídas dimensionarão o número de iterações e o desempenho dos algoritmos de detecção. Em função das diversas questões levantadas acima e com o objetivo de comparar os algoritmos em um mesmo cenário, todas as simulações expostas no Capítulo 5 foram feitas sobre uma única execução das funções sobre uma rede. Embora seja importante mencionar que resultados obtidos com outras redes e outras condições iniciais são qualitativamente semelhantes.

Sabe-se a partir dos Experimentos Completos, Seção 5.2, que mesmo utilizando diferentes parâmetros de entrada, a metodologia consegue manter a estrutura em comunidades da rede. Mas será que a metodologia também mantém as medidas intrínsecas à rede LFR inicial? Tais medidas são o grau médio  $\langle k \rangle$ , o grau máximo  $k_{max}$ , o *mixing parameter*  $\mu$  e os tamanhos mínimo e máximo de comunidade, *minsizecomu* e *maxsizecomu*, respectivamente. Para os trabalhos futuros pretende-se investigar se tais medidas são mantidas pela rede. Em caso negativo, as funções deverão ser adaptadas para assegurar que a rede mantenha as características iniciais.

Também planeja-se utilizar a metodologia para a geração de um *benchmark* para testar algoritmos de detecção para Redes Dinâmicas, como os descritos na Seção 3.2.5. Nesse trabalho não foram utilizados tais algoritmos pois não houve tempo hábil para a aquisição dos códigos fontes ou mesmo a implementação destes. Os algoritmos Fast Greedy, Infomap, Label Propagation e Walktrap já se encontravam disponíveis na biblioteca *igraph* (CSARDI; NEPUSZ, 2006).

Por fim, propõe-se a criação de uma biblioteca nas linguagens C (RITCHIE, 1993) e R (R Core Team, 2012) para distribuição e utilização da metodologia por outros pesquisadores. Possibilitando inclusive o melhoramento dos códigos fontes e a expansão da biblioteca para abranger também redes direcionadas e ponderadas, redes com comunidades sobrepostas e redes com estruturas hierárquicas.

## REFERÊNCIAS BIBLIOGRÁFICAS

- ALBERT, R.; BARABÁSI, A.-L. Statistical mechanics of complex networks. **Rev. Mod. Phys.**, American Physical Society, v. 74, n. 1, p. 47–97, jan. 2002. Disponível em: <<http://link.aps.org/doi/10.1103/RevModPhys.74.47>>. **xiii, 5, 11**
- ARENAS, A.; DÍAZ-GUILERA, A.; PÉREZ-VICENTE, C. J. Synchronization reveals topological scales in complex networks. **Phys. Rev. Lett.**, v. 96, p. 114102, 2006. **41**
- ASUR, S.; PARTHASARATHY, S.; UCAR, D. An event-based framework for characterizing the evolutionary behavior of interaction graphs. **TKDD**, v. 3, n. 4, 2009. **36**
- BAGROW, J. P. Evaluating local community methods in networks. **Journal of Statistical Mechanics: Theory and Experiment**, v. 2008, n. 05, p. P05001, 2008. Disponível em: <<http://stacks.iop.org/1742-5468/2008/i=05/a=P05001>>. **42**
- BANSAL, S.; BHOWMICK, S.; PAYMAL, P. Fast community detection for dynamic complex networks. **Communications in Computer and Information Science**, Springer Berlin Heidelberg, v. 116, p. 196–207, 2011. **1**
- BARABÁSI, A. L. Scale-free networks: a decade and beyond. **Science**, v. 325, p. 412, 2009. **xiii, 13**
- BARABÁSI, A. L.; ALBERT, R. Emergence of scaling in random networks. **Science**, v. 286, p. 509–512, 1999. **11, 12**
- BARABÁSI, A. L.; ALBERT, R.; JEONG, H. Mean-field theory for scale-free random networks. **Physica A: Statistical Mechanics and its Applications**, v. 272, n. 1-2, p. 173–187, out. 1999. Disponível em: <<http://www.sciencedirect.com/science/article/B6TVG-3XK0NJ2-D/1/c157dd1ab1bc62dba60f7e060184833a>>. **11**
- BARABÁSI, A. L.; ALBERT, R.; JEONG, H. Scale-free characteristics of random networks: the topology of the world wide web. **Physica A**, v. 272, p. 173–187, 2000. **12**
- BOLLOBÁS, B. **Modern graph theory**. [S.l.]: Springer, 1998. (Graduate texts in mathematics). **5**
- CHAKRABARTI, D.; KUMAR, R.; TOMKINS, A. Evolutionary clustering. In: 12TH ACM SIGKDD INTERNATIONAL CONFERENCE ON KNOWLEDGE DISCOVERY AND DATA MINING. **Proceedings...** New York, NY, USA: ACM, 2006. (KDD '06), p. 554–560. ISBN 1-59593-339-5. Disponível em: <<http://doi.acm.org/10.1145/1150402.1150467>>. **37**

- CHEN, M.; NGUYEN, T.; SRYMANSKI, B. K. A new metric for quality of network community structure. **ASE Human Jornal**, v. 1(4), p. 226–240, 2013. 22, 23, 24, 25
- CLAUSET, A.; NEWMAN, M. E. J.; MOORE, C. Finding community structure in very large networks. **Physical Review E**, v. 70, p. 066111, 2004. Disponível em: <<http://www.citebase.org/abstract?id=oai:arXiv.org:cond-mat/0408187>>. 29
- CONDON, A.; KARP, R. M. Algorithms for graph partitioning on the planted partition model. **Random Struct. Algorithms**, v. 18, n. 2, p. 116–140, 2001. 40
- COSTA, L.; RODRIGUES, F.; TRAVIESO, G.; BOAS, P. Characterization of complex networks: a survey of measurements. **Advances in Physics**, v. 56, n. 1, p. 167–242, 2007. 10
- COSTA, L. F.; Oliveira Jr., O. N.; TRAVIESO, G.; RODRIGUES, F. A.; Villas Boas, P. R.; ANTIQUEIRA, L.; VIANA, M. P.; ROCHA, L. E. C. Analyzing and modeling real-world phenomena with complex networks: a survey of applications. **Advances in Physics**, v. 60, n. 3, p. 329–412, 2011. 1
- CSARDI, G.; NEPUSZ, T. The igraph software package for complex network research. **InterJournal**, Complex Systems, p. 1695, 2006. Disponível em: <<http://igraph.sf.net>>. 48, 78
- DANON, L.; DÍAZ-GUILERA, A.; ARENAS, A. The effect of size heterogeneity on community identification in complex networks. **Journal of Statistical Mechanics: Theory and Experiment**, v. 2006, n. 11, p. P11010, 2006. Disponível em: <<http://stacks.iop.org/1742-5468/2006/i=11/a=P11010>>. xiv, 42, 43
- DANON, L.; DUCH, J.; ARENAS, A.; DÍAZ-GUILERA, A. Comparing community structure identification. **Journal of Statistical Mechanics: Theory and Experiment**, v. 9008, p. 09008, 2005. 27
- DIESTEL, R. **Graph theory**. 3. ed. [S.l.]: Springer-Verlag Heidelberg, 2005. 5, 6
- DONGEN, S. van. **Graph clustering by flow simulation**. Tese (Doutorado) — University of Utrecht, 2000. 36
- DUNN, J. C. A fuzzy relative of the isodata process and its use in detecting compact well-separated clusters. **Journal of Cybernetics**, v. 3, n. 3, p. 32–57, 1973. Disponível em: <<http://www.tandfonline.com/doi/abs/10.1080/01969727308546046>>. 28



ERDOS, P.; RÉNYI, A. On random graphs i. **Publ. Math. (Debrecen)**, v. 6, p. 290, 1959. 10

\_\_\_\_\_. On the evolution of random graphs. **Publ. Math. Inst. Hung. Acad. Sci.**, v. 5, p. 17, 1960. 10

EULER, L. Solutio problematis ad geometriam situs pertinentis. **Commentarii Acadamiae Petropolitanae**, v. 8, p. 128, 1736. 5

FACELI, K. **Um framework para análise de agrupamento baseado na combinação multi-objetivo de algoritmos de agrupamento**. Tese (Doutorado) — USP - São Carlos, 2007. 17

FACELI, K.; CARVALHO, A. de; SOUTO, M. C. P. de. Validação de algoritmos de agrupamento. **Relatórios Técnicos do ICMC**, n. 254, 2005. 17

FALKOWSKI, T.; BARTH, A.; SPILIOPOULOU, M. Densgraph: A density-based community detection algorithm. In: IEEE/WIC/ACM INTERNATIONAL CONFERENCE ON WEB INTELLIGENCE. **Proceedings...** Washington, DC, USA: IEEE Computer Society, 2007. (WI '07), p. 112–115. ISBN 0-7695-3026-5. Disponível em: <<http://dx.doi.org/10.1109/WI.2007.43>>. 37

FAN, Y.; LI, M.; ZHANG, P.; WU, J.; DI, Z. Accuracy and precision of methods for community identification in weighted networks. **Physica A: Statistical Mechanics and its Applications**, v. 377, n. 1, p. 363–372, abr. 2007. Disponível em: <<http://www.sciencedirect.com/science/article/B6TVG-4MHP94W-4/1/00988db3212bd4667da5ee3cf99401ff>>. 41

FENN, D.; PORTER, M.; MCDONALD, M.; WILLIAMS, S.; JOHNSON, N.; JONES, N. Dynamic communities in multichannel data: an application to the foreign exchange market during the 2007 and 2008 credit crisis. 2009. 36

FEOFILOFF, P.; KOHAYAKAWA, Y.; WAKABAYASHI, Y. **Uma Introdução Sucinta à Teoria dos Grafos**. [S.l.: s.n.], 2011. 5, 6

FORTUNATO, S. Community detection in graphs. **Physics Reports**, v. 486, p. 75 – 174, 2010. xiii, 1, 5, 6, 8, 10, 15, 16, 18, 19, 20, 21, 27, 40

\_\_\_\_\_. **Benchmark graphs to test community detection algorithms**. 2013. Acessado em 30/11/2013. Disponível em: <<https://sites.google.com/site/santofortunato/inthepress2>>. xiv, 44, 45

- FORTUNATO, S.; BARTHÉLEMY, M. Resolution limit in community detection. **National Academy of Sciences**, v. 104, n. 1, p. 36, 2007. 20
- FRANKE, M.; GEYER-SCHULZ, A. An update algorithm for restricted random walk clustering for dynamic data sets. **Adv. Data Analysis and Classification**, v. 3, n. 1, p. 63–92, 2009. 37
- FREEMAN, L. A set of measures of centrality based on betweenness. **Sociometry**, v. 40, p. 35–41, 1977. 9
- GIRVAN, M.; NEWMAN, M. E. J. Community structure in social and biological networks. **PNAS**, v. 99, n. 12, p. 7821–7826, June 2002. 41
- HUBERT, L.; ARABIE, P. Comparing partitions. **Journal of Classification**, v. 2, p. 193–218, 1985. 25
- JAIN, A. K.; DUBES., R. **Algorithms for clustering data**. [S.l.]: Prentice Hall, 1988. 17
- KIM, M.-S.; HAN, J. A particle-and-density based evolutionary clustering method for dynamic networks. **Proc. VLDB Endow.**, VLDB Endowment, v. 2, n. 1, p. 622–633, ago. 2009. Disponível em:  
<<http://dl.acm.org/citation.cfm?id=1687627.1687698>>. 36
- KIRKPATRICK, S.; GELATT, C.; VECCHI, M. Optimization by simulated annealing. **Science**, American Association for the Advancement of Science, v. 220, n. 4598, p. 671, 1983. 34
- LANCICHINETTI, A.; FORTUNATO, S. Benchmarks for testing community detection algorithms on directed and weighted graphs with overlapping communities. **Phys. Rev. E**, v. 80, n. 1, p. 016118, jul. 2009. Disponível em:  
<<http://pre.aps.org/abstract/PRE/v80/i1/e016118>>. 44
- \_\_\_\_\_. Community detection algorithms: A comparative analysis. **Phys. Rev. E**, American Physical Society, v. 80, p. 056117, Nov 2009. Disponível em:  
<<http://link.aps.org/doi/10.1103/PhysRevE.80.056117>>. 15, 27, 40, 41, 43, 68
- LANCICHINETTI, A.; FORTUNATO, S.; RADICCHI, F. Benchmark graphs for testing community detection algorithms. **Physical Review E**, v. 78, n. 4, p. 046110, 2008. 42, 43, 44, 68
- LIN, Y.-R.; CHI, Y.; ZHU, S.; SUNDARAM, H.; TSENG, B. L. Facetnet: a framework for analyzing communities and their evolutions in dynamic networks. In: 17TH

- INTERNATIONAL CONFERENCE ON WORLD WIDE WEB. **Proceedings...** New York, NY, USA: ACM, 2008. p. 685–694. ISBN 978-1-60558-085-2. 37
- LIN, Y.-R.; SUNDARAM, H.; CHI, Y.; TATEMURA, J.; TSENG, B. L. Blog community discovery and evolution based on mutual awareness expansion. In: **WEB INTELLIGENCE. Proceedings...** [S.l.]: IEEE Computer Society, 2007. p. 48–56. ISBN 0-7695-3026-5. 47
- MAITY, S. K.; MANOJ, T. V.; MUKHERJEE, A. Opinion formation in time-varying social networks: The case of naming game. **CoRR**, abs/1204.1160, 2012. 14
- MILGRAM, S. The small world problem. **Psychology Today**, v. 2, p. 60–67, 1967. 11
- MUCHA, P. J.; RICHARDSON, T.; MACON, K.; PORTER, M. A.; ONNELA, J.-P. Community structure in time-dependent, multiscale, and multiplex networks. **Science**, v. 328, n. 5980, p. 876–878, 2010. xiii, 21, 22
- NEWMAN, M. E. J. The structure and function of complex networks. **Siam Review**, v. 45, p. 167–256, 2003. 8, 10
- \_\_\_\_\_. Fast algorithm for detecting community structure in networks. **Physical Review**, E 69, n. 066133, 2004. 29
- \_\_\_\_\_. Communities, modules and large-scale structure in networks. **Nature Physics**, Nature Publishing Group, v. 8, n. 1, p. 25–31, dez. 2011. Disponível em: <<http://dx.doi.org/10.1038/nphys2162>>. 1
- NEWMAN, M. E. J.; GIRVAN, M. Finding and evaluating community structure in networks. **Phys. Rev. E**, v. 69, n. 2, p. 026113, fev. 2004. Disponível em: <<http://link.aps.org/doi/10.1103/PhysRevE.69.026113>>. 15, 18, 19, 41
- NICOSIA, V.; TANG, J.; MUSOLESI, M.; RUSSO, G.; MASCOLO, C.; LATORA, V. Components in time-varying graphs. **CoRR**, abs/1106.2134, 2012. 13, 14
- ORMAN, G. K.; LABATUT, V.; CHERIFI, H. Qualitative comparison of community detection algorithms. **CoRR**, abs/1207.3603, 2012. 15
- PALLA, G.; BARABASI, A.-L.; VICSEK, T. Quantifying social group evolution. **Nature**, v. 446, n. 7136, p. 664–667, April 2007. Disponível em: <<http://dx.doi.org/10.1038/nature05670>>. xiv, 36, 47, 48
- PALLA, G.; DERENYI, I.; FARKAS, I.; VICSEK, T. Uncovering the overlapping community structure of complex networks in nature and society. **Nature**, v. 435, n. 7043,

p. 814–818, jun. 2005. Disponível em:

<<http://dx.doi.org/10.1038/nature03607>>. 36

PAPADOPOULOS, S.; KOMPATSIARIS, Y.; VAKALI, A.; SPYRIDONOS, P. Community detection in social media - performance and application considerations.

**Data Min. Knowl. Discov.**, v. 24, n. 3, p. 515–554, 2012. 1, 47

PONS, P.; LATAPY, M. Computing communities in large networks using random walks.

**J. Graph Algorithms Appl.**, v. 10, n. 2, p. 191–218, 2006. 35

QUILES, M. G.; ZORZAL, E. R.; MACAU, E. E. N. A dynamic model for community detection in complex networks. **The International Joint Conference on Neural**

**Networks (IJCNN)**, v.1, p. p 1–8, 2013. To be published. xiv, 37, 39

R Core Team. **R: a language and environment for statistical computing**. Vienna,

Austria, 2012. ISBN 3-900051-07-0. Disponível em: <<http://www.R-project.org/>>. 48, 78

RAGHAVAN, U. N.; ALBERT, R.; KUMARA, S. Near linear time algorithm to detect community structures in large-scale networks. **Phys. Rev. E**, v. 76, n. arXiv:0709.2938,

p. 036106. 12 p, Sep 2007. xiii, 30, 34

RAND, W. Objective criteria for the evaluation of clustering methods. **Journal of the American Statistical Association**, v. 66, n. 336, p. 846–850, 1971. 25

REICHARDT, J.; BORNHOLDT, S. Statistical mechanics of community detection.

**Physical Review E**, v. 74, p. 016110, 2006. Disponível em:

<<http://www.citebase.org/abstract?id=oai:arXiv.org:cond-mat/0603718>>. 36

RITCHIE, D. M. **The development of the C language**. New York: Association for Computing Machinery, 1993. 201–208 p. (Digital Library). 48, 78

ROCHA, L. H. C. da. **Redes acopladas: Estrutura e dinâmica**. Dissertação (Mestrado)

— Instituto de Física de São Paulo, Universidade de São Paulo, São Paulo, 2007. 5, 6, 10

RODRIGUES, F. A. **Caracterização, classificação e análise de redes complexas**. Tese (Doutorado) — Instituto de Física de São Paulo, Universidade de São Paulo, 2007. 5, 6,

8, 10, 11

ROSVALL, M.; AXELSSON, D.; BERGSTROM, C. T. The map equation. **The**

**European Physical Journal Special Topics**, v. 178, n. 1, p. 13–23, 2009. Disponível em: <<http://dx.doi.org/10.1140/epjst/e2010-01179-1>>. 30

ROSVALL, M.; BERGSTROM, C. T. Maps of random walks on complex networks reveal community structure. **National Academy of Sciences**, v. 105, n. 4, p. 1118–1123, 2008. 30, 34

SCHLITTER, N.; FALKOWSKI, T. Mining the dynamics of music preferences from a social networking site. In: 2009 INTERNATIONAL CONFERENCE ON ADVANCES IN SOCIAL NETWORK ANALYSIS AND MINING. **Proceedings...** Washington, DC, USA: IEEE Computer Society, 2009. (ASONAM '09), p. 243–248. ISBN 978-0-7695-3689-7. Disponível em:  
<<http://dx.doi.org/10.1109/ASONAM.2009.26>>. 47

SUN, J.; FALOUTSOS, C.; PAPADIMITRIOU, S.; YU, P. S. Graphscope: Parameter-free mining of large time-evolving graphs. In: 13TH ACM SIGKDD INTERNATIONAL CONFERENCE ON KNOWLEDGE DISCOVERY AND DATA MINING. **Proceedings...** New York, NY, USA: ACM, 2007. (KDD '07), p. 687–696. ISBN 978-1-59593-609-7. Disponível em:  
<<http://doi.acm.org/10.1145/1281192.1281266>>. 37

TRUDEAU, R. **Introduction to graph theory**. Dover Pub., 1993. (Dover Books on Mathematics Series). ISBN 9780486678702. Disponível em:  
<<http://books.google.com.br/books?id=8nYH5OYEW24C>>. 6

VIANA, M. P. **A metodologia das redes complexas para caracterização do sistema de havers**. Tese (Doutorado) — Universidade de São Paulo, São Paulo, 2007. xiii, 10, 12

VINH, N. X.; EPPS, J.; BAILEY, J. Information theoretic measures for clusterings comparison: is a correction for chance necessary? In: 26TH ANNUAL INTERNATIONAL CONFERENCE ON MACHINE LEARNING. **Proceedings...** New York, NY, USA: ACM, 2009. p. 1073–1080. ISBN 978-1-60558-516-1. 26

WANG, Y.; WU, B.; DU, N. **Community evolution of social network: feature, algorithm and model**. 2008. Eprint, acessado em 05/12/2013. Disponível em:  
<<http://arxiv.org/abs/0804.4356>>. 37

WARD, J. H. Hierarchical grouping to optimize an objective function. **Journal of the American Statistical Association**, v. 58, n. 301, p. 236–244, 1963. Disponível em:  
<<http://www.tandfonline.com/doi/abs/10.1080/01621459.1963.10500845>>. 36

WATTS, D. J.; STROGATZ, S. H. Collective dynamics of small-world networks. **Nature**, v. 393, n. 6684, p. 440–442, June 1998. 11

WIKIPEDIA. **Benchmark (computing)**. 2013. Acessado em 29/11/2013. Disponível em: <[http://en.wikipedia.org/wiki/Benchmark\\_\(computing\)](http://en.wikipedia.org/wiki/Benchmark_(computing))>. 40

XU, X.; YURUK, N.; FENG, Z.; SCHWEIGER, T. A. J. Scan: A structural clustering algorithm for networks. In: 13TH ACM SIGKDD INTERNATIONAL CONFERENCE ON KNOWLEDGE DISCOVERY AND DATA MINING. **Proceedings...** New York, NY, USA: ACM, 2007. (KDD '07), p. 824–833. ISBN 978-1-59593-609-7. Disponível em: <<http://doi.acm.org/10.1145/1281192.1281280>>. 36

YANG, S.; WANG, B.; ZHAO, H.; WU, B. Efficient dense structure mining using mapreduce. In: SAYGIN, Y.; YU, J. X.; KARGUPTA, H.; 0010, W. W.; RANKA, S.; YU, P. S.; WU, X. (Ed.). **Proceedings...** [S.l.]: IEEE Computer Society, 2009. p. 332–337. ISBN 978-0-7695-3902-7. 37