DETECÇÃO DE COMUNIDADES EM GRAFOS MULTICAMADA MUITO GRANDES

FELIPE MENEZES MACHADO

DETECÇÃO DE COMUNIDADES EM GRAFOS MULTICAMADA MUITO GRANDES

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação do Instituto de Ciências Exatas da Universidade Federal de Minas Gerais como requisito parcial para a obtenção do grau de Mestre em Ciência da Computação.

Orientador: Virgílio Augusto Fernandes de Almeida

Belo Horizonte

Setembro de 2011

Ficha catalográfica elaborada pela Biblioteca do ICEx - UFMG

Machado, Felipe Menezes.

M149d Detecção de comunidades em grafos multicamada muito grandes. / Felipe Menezes Machado. – Belo Horizonte, 2016.
xxiv, 122 f.: il.; 29 cm.

Dissertação (mestrado) - Universidade Federal de Minas Gerais – Departamento de Ciência da Computação.

Orientador: Virgílio Augusto Fernandes Almeida.

1. Computação - Teses. 2. Teoria das redes complexas 3. Teoria dos grafos. I. Orientador. II. Título.

CDU 519.6*62 (043)

UNIVERSIDADE FEDERAL DE MINAS GERAIS
INSTITUTO DE CIÊNCIAS EXATAS
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

FOLHA DE APROVAÇÃO

Detecção de comunidades em grafos multicamada muito grandes

FELIPE MENEZES MACHADO

Dissertação defendida e aprovada pela banca examinadora constituída pelos Senhores:

PROF. VIRGÍLIO AUGUSTO FERNANDES ALMEIDA - Orientador
Departamento de Ciência da Computação - UFMG

PROF. WAGNER MEIRA JÚNIOR
Departamento de Ciência da Computação - UFMG

PROF. RENATO ANTÔNIO CELSO FERREIRA
Departamento de Ciência da Computação - UFMG

PROF. JAYME LUIZ SZWARCFITER
Núcleo de Computação Eletrônica - UFRJ

Belo Horizonte, 19 de setembro de 2011.

Dedico esse trabalho a meus pais, que tanto trabalharam para proporcionar uma educação digna aos seus filhos.

Agradecimentos

Agradeço primeiramente a Samara, minha companheira de todos momentos durante esse período.

Agradeço meus pais, que tanto me ensinaram e tanto trabalharam para que eu pudesse chegar até esse trabalho. Agradeço meus irmãos, meus melhores amigos e companheiros durante a vida.

Agradeço ao meus avós pelo grande exemplo de vida e por todo apoio.

Agradeço a Carmen e o Vicente, que me acolheram com muito carinho e sempre me ajudam quando necessário.

Agradeço toda minha família, que sempre me proporcionam diversão, carinho e amizade.

Agradeço especialmente aqueles que não estão mais entre nós: meu avô Augusto, que teve grande orgulho por ter um neto com curso superior, resultado de seu trabalho duro para constituir a família que me originou; e minha tia Cirlei, professora muito dedicada e trabalhadora, e uma tia muito carinhosa e dedicada, da qual muito me orgulho.

Agradeço meu orientador Virgílio, que originou esse trabalho e é responsável pelo meu interesse na pesquisa e pelas grandes oportunidades que tive enquanto aluno.

Agradeço meus companheiros da Maratona de Programação: Leonardo, Itamar e Thiago, que tiveram papel importantíssimo em uma das minhas maiores conquistas, a classificação para a final mundial do ICPC.

Agradeço todos meus amigos, amigos do DCC, amigos de infância, amigos da Sambatech, pelas horas de divertimento, desabafo, papo-furado, trabalho, etc.

Agradeço ao David e todo o pessoal do laboratório Mascotte no INRIA, França, onde parte desse trabalho foi desenvolvido.

"Todas verdades são fáceis de enteder uma vez descobertas;
o ponto é descobri-las."
(Galileo Galilei)

Resumo

Vários trabalhos foram desenvolvidos na última década no campo de comunidades em grafos, com uma vasta gama de definições, análises e técnicas desenvolvidas. No entanto, só recentemente foi desenvolvido o primeiro trabalho sobre comunidades em grafos multicamada (grafos com vários tipos de aresta). As técnicas utilizadas nesse trabalho não permitem a utilização em grafos muito grandes (da ordem de milhões de vértices e arestas), e portanto ainda não há um método que permita a detecção de comunidades em grandes grafos multicamada.

Nesse trabalho apresentamos um novo algoritmo para esse fim, ou seja, que faz o particionamento em comunidades de grafos multicamada muito grandes. Partindo de uma fórmula desenvolvida recentemente para a modularidade de grafos multicamada, multiresolução e dependentes do tempo, nós derivamos uma fórmula mais simples para grafos multicamada, com algumas simplificações. Então, baseando-se na estrutura da fórmula mais simples, nós modificamos um algoritmo guloso bem conhecido para detecção de comunidades baseado em modularidade, para chegar a um novo algoritmo que nos dá melhores resultados de modularidade multicamada quando aplicado a esse tipo de grafos. Resultados experimentais nos mostram que podemos atingir melhores resultados para modularidade multicamada com uma perda de eficiência da ordem da multiplicidade do grafo multicamada. Concluímos então que o algoritmo é viável para grafos muito grandes, e pode ser utilizado para analisar redes complexas existentes com melhores resultados que os algoritmos encontrados na literatura.

Palavras-chave: Redes Complexas, Grafos multicamada, Comunidades em grafos.

Abstract

Many studies have been developed in the last decade in the field of communities in graphs, with a great variety of definitions, analysis and techniques developed. However, only recently the first work about communities in multiplex graphs (graphs with many types of edges) has been published. The techniques used in this study do not allow for use with very large graphs (in the order of millions of vertexes and edges), and therefore there isn't yet a method that allows for detection of communities in very large multiplex networks.

In this work we present such an algorithm, one that partitions large multiplex graphs in communities. Starting from a recently developed modularity formula for multiplex, multi-resolution and time-dependent graphs, we come up with a simpler formula for multiplex networks, with some simplifications. Then, based on the structure of the simpler formula, we modify a well-known greedy algorithm for community detection based on modularity, to come up with a new algorithm that gives us better results of multiplex modularity when applied to this type of graphs. Experimental results shows us that we can achieve better results of multiplex modularity with a loss of performance in the order of the multiplex graph multiplicity. We then conclude that the algorithm is viable for very large graphs, and can be used to analyze existing complex networks with better results than with the algorithms found in the literature.

Keywords: Complex Networks, Multiplex Graphs, Communities in networks.

Lista de Figuras

Lista de Tabelas

Sumário

Capítulo 1

Introdução

Detecção de comunidades em grafos é um tema amplamente abordado, com vários estudos que variam tanto em definições quanto em técnicas Fortunato [2010]. Comunidades em grafos tem várias definições, desde definições locais (baseadas nos pequenos grupos de vértices) até definições globais (baseadas no grafo como um todo) e definições baseadas em similaridade de vértices. Da mesma maneira existem também várias técnicas, cada qual baseada em uma determinada definição de comunidade. Exemplos de técnicas tradicionais são o particionamento de grafos e os vários tipos de agrupamento (hierárquica, espectral, etc).

Comunidades, também chamadas de *clusters* ou agrupamentos, são grupos de vértices que tem grande probabilidade de compartilhar propriedades comuns ou tem papéis semelhantes no grafo. Podemos ter estruturas de comunidades que permitem a cada nó do grafo pertencer a uma só comunidade, o que seria um particionamento do conjunto de vértices, ou permitindo a cada nó pertencer a várias comunidades. Temos vários exemplos de organizações em grupos em grafos no mundo real, variando desde o âmbito social, até os campos da Web, Ciência da Computação, Biologia, Economia e outros. Comunidades podem também ter várias aplicações concretas, como por exemplo: balanceamento de carga da Web (Krishnamurthy & Wang [2000]), recomendações de produtos a clientes (Reddy et al. [2002]), redes *ad-hoc* (Perkins [2001]).

O objetivo da detecção de comunidades em grafos é identificar os grupos utilizando somente a informação contida na topologia do grafo. Esse problema já é antigo e os primeiros trabalhos sobre comunidades em grafos datam de mais de 50 anos atrás. Um trabalho proposto por Girvan & Newman [2002], onde eles apresentam um novo algoritmo de detecção de comunidades, despertou uma grande atividade na área, e vários novos métodos foram desenvolvidos recentemente, com várias contribuições dos campos da Física, Ciência da Computação, Sociologia e Matemática discreta.

A função de modularidade Newman & Girvan [2004] ganhou bastante popularidade nessa área por resumir vários aspectos das estruturas de comunidades em uma função baseada na comparação com um grafo aleatório (chamado modelo nulo). Depois de ser amplamente adotada, a métrica de modularidade foi utilizada como base de vários algoritmos de otimização para detecção de comunidades em grafos, desde otimizações gulosas até otimizações espectrais. Foram desenvolvidos também alguns poucos algoritmos baseados na função de modularidade para grafos muito grandes (Newman [2004a]; Clauset et al. [2004]; Blondel et al. [2008]; Shah & Zaman [2010]). Veremos mais detalhes sobre a função de modularidade na seção 1.3

O primeiro algoritmo proposto de detecção de comunidades em grafos muito grandes foi o apresentado por Newman [2004a]. Trata-se de um algoritmo guloso que forma comunidades de forma aglomerativa, agrupando comunidades até obter um bom particionamento. O trabalho por Clauset et al. [2004] apresenta uma reformulação nas estruturas de dados do algoritmo de Newman para otimizar a complexidade e permitir o uso do algoritmo em grafos ainda maiores. Várias modificações foram propostas com base nesses algoritmos, com o intuito de melhorar a qualidade da modularidade encontrada e deixar o algoritmo mais eficiente. Veremos mais detalhes sobre essa família de algoritmos na seção 1.4.

Outro algoritmo proposto para detecção de comunidades em grandes grafos é o desenvolvido por Blondel et al. [2008]. Trata-se também de um algoritmo guloso, para aplicação em grafos com pesos nas arestas. O primeiro passo consiste de uma varredura por todos vértices; dado um vértice i, calcula-se o ganho em modularidade com pesos (Newman [2004b]) obtido ao se colocar o vértice i na comunidade de seu vizinho j, e escolhe-se a comunidade do vizinho que dá o maior aumento de modularidade (se positivo). As novas comunidades são trocadas por super-vértices, onde conexões entre esses vértices são arestas com peso igual à soma dos pesos nas arestas entre vértices dessas comunidades. Então esses passos são repetidos até que o valor de modularidade não possa mais ser melhorado, e então o algoritmo para. De acordo com os experimentos do trabalho, os valores de modularidade encontrados são melhores que os dos algoritmos por Clauset et al. (Clauset et al. [2004]) e Wakita e Tsurumi (Wakita & Tsurumi [2007]), e o algoritmo é altamente eficiente, sendo limitado somente pelo uso de memória, da ordem do número de arestas do grafo.

Somente recentemente foi proposta uma função de modularidade para grafos multicamada (Mucha et al. [2010]), que permite (através de algoritmos de otimização) a detecção de comunidades nesse tipo de grafos. No entanto, como os algoritmos genéricos de otimização (que aceitam qualquer função linear para ser otimizada) são relativamente lentos, as técnicas propostas não podem ser utilizadas para grafos muito

grandes (da ordem de milhões de vértices e arestas). No trabalho citado só são analisados grafos de menos de 2 mil vértices. Veremos mais detalhes sobre grafos multicamada na seção 1.5

1.1 Objetivos

A proposta desse trabalho é desenvolver um algoritmo eficiente para a detecção de comunidades em grafos multicamada muito grandes. Pesquisando na literatura, não foi encontrado nenhum algoritmo semelhante, e acreditamos que um algoritmo simples possa dar resultados satisfatórios e permita vários trabalhos futuros. Com base em um algoritmo de detecção de comunidades em grafos comuns já existente, e utilizando ideias sobre comunidades em grafos multicamada de um trabalho recente, desenvolvemos um algoritmo cujo objetivo é melhorar a qualidade das estruturas de comunidades encontradas em grafos reais e iniciar o desenvolvimento e interesse nesse tipo de grafo que pode ser facilmente encontrado no mundo real.

1.2 Motivação

Ainda não existem na literatura algoritmos para detecção de comunidades em grandes grafos multicamada. Dada a grande quantidade de redes complexas existentes atualmente, muitas das quais podem ser combinadas em grafos multicamada, é de grande interesse um algoritmo de detecção de comunidades nesse tipo de grafo, uma vez que fazendo-se uso das informação extras contidas nos grafos multicamada podemos atingir melhores resultados (melhores comunidades).

1.3 Modularidade

A métrica de modularidade (Newman & Girvan [2004]) define uma função com base na estrutura de comunidades (um mapeamento de vértices para comunidades), no grafo e em um modelo de grafo nulo (possivelmente dependente do grafo original). O modelo de grafo nulo define como seriam as arestas em um grafo aleatório. O modelo nulo deve manter o máximo de coesão com o grafo original, mantendo os vértices e possivelmente os graus dos vértices, somente alterando a ordem das arestas. Assim temos um grafo com a mesma densidade de arestas mas sem uma estrutura de comunidades (pois as arestas são aleatórias). Uma vez tendo definido o modelo nulo, a modularidade é definida como:

Definição A modularidade de um grafo, dado um modelo nulo P, é definida como:

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

Onde A_{ij} é um valor que indica a existência ou não (valor 1 ou 0) de um aresta entre os vértices i e j do grafo; P_{ij} é um valor equivalente, só que indica a existência da aresta no modelo nulo; m é o número de arestas no grafo; e C_i é a comunidade do vértice i. A função δ é o delta de Kronecker, que vale 1 caso as variáveis (no caso C_i e C_j) sejam iguais, ou 0 caso contrário.

Se assumirmos um modelo nulo padrão, onde os vértices mantêm seus graus, e a distribuição das arestas é uniforme, então podemos utilizar $P_{ij} = \frac{k_i k_j}{2m}$, onde k_i é o grau do vértice i no grafo original. Assim teremos a fórmula da modularidade como sendo:

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

Como a função delta só será diferente de zero quando ambos vértices estiverem na mesma comunidade, podemos reescrever a fórmula como uma soma sobre as comunidades, onde o somatório das arestas será sobre as arestas da comunidade. Assim temos:

$$Q = \frac{1}{2m} \sum_{c=1}^{n_c} \sum_{i,j \in C_c} \left(A_{ij} - \frac{k_i k_j}{2m} \right)$$

Onde n_c é o número de comunidades. Se chamarmos de l_c o número de arestas entre vértices da comunidade C_c (ou seja, $l_c = \frac{1}{2} \sum_{i,j \in C_c} A_{ij}$) e d_c o somatório dos graus dos vértices pertencentes a comunidade C_c (ou seja, $d_c = \sum_{i \in C_c} k_i$), podemos fazer a seguinte reformulação:

$$Q = \sum_{c=1}^{n_c} \left(\frac{l_c}{m} - \left(\frac{d_c}{2m} \right)^2 \right)$$

Assim temos uma fórmula alternativa da modularidade de uma estrutura de comunidades em um grafo. Aqui temos que $\frac{l_c}{m}$ é a fração de arestas no grafo que conectam vértices dentro do grupo c, e $\frac{d_c}{2m}$ é a fração de extremidades de arestas ligadas a comunidade c. Definiremos esses valores da seguinte maneira:

Definição e_{ij} é a fração de arestas no grafo que conectam vértices na comunidade i a

aqueles na comunidade j:

$$e_{ij} = \frac{1}{2m} \sum_{vw} A_{vw} \delta(c_v, i) \delta(c_w, j)$$

Definição a_i é a fração de extremidades de arestas que estão ligadas a vértices na comunidade i:

$$a_i = \frac{1}{2m} \sum_{v} k_v \delta(c_v, i)$$

De onde tiramos o seguinte corolário:

Corolário 1.3.1

$$a_i = \sum_j e_{ij}$$

Prova

$$\begin{aligned}
\sum_j e_{ij} &= \sum_j \frac{1}{2m} \sum_{vw} A_{vw} \delta(c_v, i) \delta(c_w, j) = && (1.1)\\
&= \frac{1}{2m} \sum_j \sum_{vw} A_{vw} \delta(c_v, i) \delta(c_w, j) = && (1.2)\\
&= \frac{1}{2m} \sum_{vw} \sum_j A_{vw} \delta(c_v, i) \delta(c_w, j) = && (1.3)\\
&= \frac{1}{2m} \sum_{vw} A_{vw} \delta(c_v, i) = \frac{1}{2m} \sum_v k_v \delta(c_v, i) = a_i && (1.4)
\end{aligned}$$

Substituindo esse valores na fórmula alternativa da modularidade, teremos então que:

$$Q = \sum_i \left(e_{ii} - a_i^2 \right)$$

E assim temos uma fórmula simplificada e compacta da modularidade. Outra maneira de chegarmos a essa fórmula, utilizando as definições de e_{ij} e a_i apresentadas acima, é fazendo $\delta(c_v, c_w) = \sum_i \delta(c_v, i) \delta(c_w, i)$, de onde teremos:

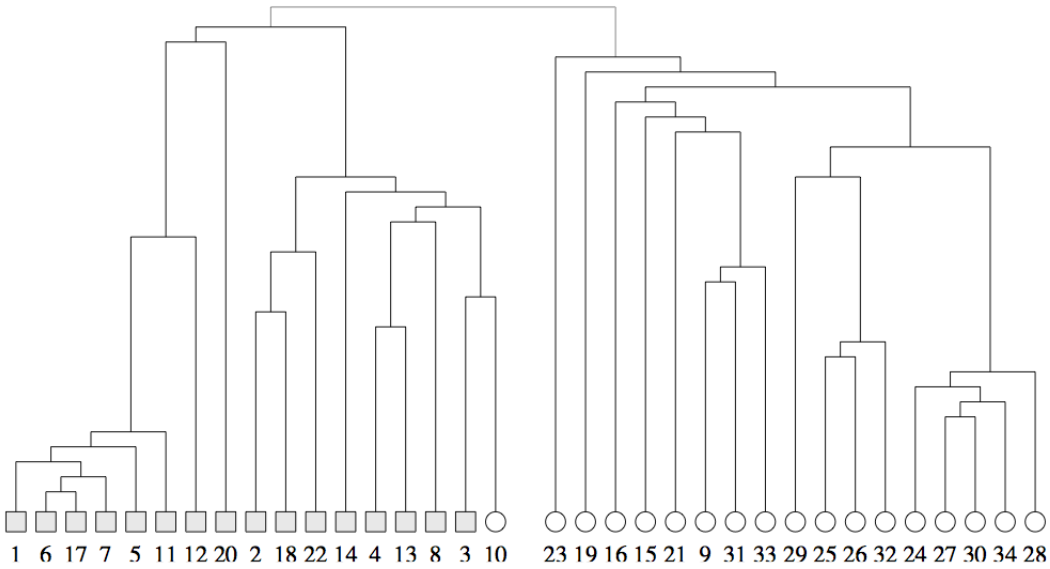


Figura 1.1: Dendrograma

$$Q = \frac{1}{2m}\sum_{vw}\left[A_{vw} - \frac{k_v k_w}{2m}\right]\sum_i \delta(c_v, i)\delta(c_w, i) = \tag{1.5}$$

$$= \sum_i \left[\frac{1}{2m}\sum_{vw} A_{vw}\delta(c_v, i)\delta(c_w, i)\right. \tag{1.6}$$

$$\left. -\frac{1}{2m}\sum_v k_v\delta(c_v, i)\frac{1}{2m}\sum_w k_w\delta(c_w, i)\right] \tag{1.7}$$

$$= \sum_i \left(e_{ii} - a_i^2\right) \tag{1.8}$$

1.4 Detecção de comunidades em grafos grandes

O primeiro algoritmo para particionamento de grafos muito grandes em comunidades foi proposto por Newman [2004a]. É um algoritmo guloso classificado como um método de agrupamento aglomerativo hierárquico (*hierarchical agglomerative clustering*). A ideia é basicamente começar com um particionamento onde cada vértice pertence a sua própria comunidade, e ir aglomerando pares de comunidades até chegar a uma só comunidade contendo todos os nós do grafo. Assim formamos um dendrograma, como o demonstrado na figura 1.1. Durante o processo de aglomeração, o algoritmo calcula também a modularidade da estrutura de comunidades atual, e escolhe-se como o melhor particionamento aquele que tem o maior valor de modularidade.

O melhor particionamento encontrado durante esse processo é a estrutura de comunidades determinada pelo algoritmo. O algoritmo é guloso pois a escolha do par de comunidades a serem unidades é uma escolha gulosa, onde se escolhe o par que ao ser unido dará o maior aumento de modularidade naquele passo.

A análise de complexidade do algoritmo nos dá uma complexidade de $O((m+n)n)$ onde n é o número de vértices e m é o número de arestas, ou seja, uma complexidade quadrática no tamanho do grafo, se considerarmos o número de arestas limitado pelo número de vértices ($m = O(n)$). Resultados experimentais mostram que o algoritmo seria viável (ainda que lento) para grafos até da ordem de centenas de milhares de vértices.

O trabalho feito por Clauset et al. [2004] é baseado no algoritmo anterior, e consiste da modificação das estruturas de dados utilizadas para alcançar um tempo de execução melhor para o algoritmo. Ao utilizar árvores binárias e filas de prioridade, é alcançada uma complexidade de $O(md \log n)$, onde d seria a altura do dendrograma. O autor argumenta que grafos do mundo real normalmente possuem $d \sim \log n$ e $m \sim n$, e portanto teríamos uma complexidade de aproximadamente $O(n \log^2 n)$, o que permitiria a aplicação em grafos da ordem de milhões de vértices. Resultados experimentais mostram que o algoritmo é bastante eficiente e pode ser aplicado facilmente a grafos com centenas de milhares de nós e alguns milhões de arestas.

A técnica consiste em manter uma matriz, chamada de matriz ΔQ, que nos dá, para cada par de comunidades, o incremento na modularidade que teríamos ao juntar esse par de comunidades em uma só comunidade. O algoritmo otimizado trabalha com a matriz sendo um vetor de árvores binárias, onde pra cada comunidade C temos uma árvore com cada nó da árvore sendo uma comunidade X e o incremento de modularidade ΔQ_X que temos ao juntar C e X. Dessa forma, podemos consultar, alterar e remover uma comunidade da lista de comunidades ligadas a C em tempo $O(logn)$. Outra estrutura mantida pelo algoritmo é o vetor a, com os valores a_i de cada comunidade.

Inicializamos essas estruturas da seguinte forma:

- Matriz ΔQ_{ij}

$$\Delta Q_{ij} = \begin{cases} \frac{1}{2m} - \frac{k_i k_j}{(2m)^2} & \text{se } i,j \text{ estão conectados,} \\ 0 & \text{caso contrário.} \end{cases}$$

- Vetor a_i

$$a_i = \frac{k_i}{2m}$$

Uma vez inicializadas as estruturas, vem a etapa de aglomeração, onde iremos em cada passo escolher um par de comunidades para serem unidas, e fazer esse agrupamento, atualizando as estruturas de acordo com o par escolhido. As atualizações são feitas de acordo com as seguintes regras:

- Matriz ΔQ_{ij}

$$\Delta Q'_{jk} = \begin{cases} \Delta Q_{ik} + \Delta Q_{jk} & \text{se } k \text{ está conectado a } i \text{ e } j, \\ \Delta Q_{ik} - 2a_j a_k & \text{se } k \text{ está conectado somente a } i, \\ \Delta Q_{jk} - 2a_i a_k & \text{se } k \text{ está conectado somente a } j. \end{cases}$$

- Vetor a_i

$$a'_j = a_j + a_i$$

Uma vez determinada a inicialização e atualização da matriz ΔQ e do vetor a, falta determinar como é feita a escolha do par de comunidades a serem unidas em cada passo. Para tanto, é utilizado um *heap* para cada comunidade e um *heap* global. O *heap* de cada comunidade C é ordenado de forma decrescente com o valor de incremento de modularidade para cada comunidade X conectada a comunidade C. Assim o topo dessa estrutura terá a comunidade que ao se juntar com C daria o maior aumento de modularidade. Uma vez tendo o melhor par para cada comunidade C, o *heap* global é montado com o topo do *heap* de cada comunidade, ou seja, contém os melhores pares para cada comunidade. Assim, o topo do *heap* global terá o par de comunidades que dará o maior incremento de modularidade naquele passo.

Para atualizar essas estruturas, basta que, a cada vez que uma entrada na matriz ΔQ é atualizada, a entrada no *heap* correspondente seja atualizada, e caso a entrada atualizada chegue ao topo desse *heap*, atualizamos também a entrada no *heap* global. Assim, o topo do *heap* global terá sempre o par de comunidades que levam ao maior incremento na modularidade se unidas.

1.4.1 Algoritmo

Aqui temos o algoritmo guloso em pseudocódigo.

1.4.2 Modificações

Várias modificações para o algoritmo guloso foram propostas. De acordo com Wakita & Tsurumi [2007], o algoritmo por Clauset et al. [2004] tende a dar preferência a

{Entrada: Grafo g, com n vértices e m arestas}
{Variáveis: matriz ΔQ, vetor a_i, modularidade atual Q, estrutura de comunidades atual C, modularidade máxima Q_{max}, estrutura de comunidades com modularidade máxima C_{max}}
{Inicialização}
para $v = 1$ a n **faça**
  para cada vizinho w de v **faça**
    $\Delta Q_{vw} \leftarrow \frac{1}{2m} - \frac{k_v k_w}{(2m)^2}$
  fim para
  $a_v \leftarrow \frac{k_v}{2m}$
  $Q \leftarrow Q + (-a_v^2)$
  $C_v \leftarrow v$
fim para
$Q_{max} \leftarrow Q$
$C_{max} \leftarrow C$
{Aglomeração}
para *contador* $= 1$ a $n - 1$ **faça**
  $i, j \leftarrow$ comunidades com maior ΔQ
  $Q \leftarrow Q + \Delta Q_{ij}$
  $C_i \leftarrow j$
  para cada comunidade k conectada a i e j **faça**
    $\Delta Q_{jk} \leftarrow \Delta Q_{ik} + \Delta Q_{jk}$
    $\Delta Q_{ik} \leftarrow 0$ {Pode remover da estrutura de dados}
  fim para
  para cada comunidade k conectada somente a i **faça**
    $\Delta Q_{jk} \leftarrow \Delta Q_{ik} - 2a_j a_k$
    $\Delta Q_{ik} \leftarrow 0$ {Pode remover da estrutura de dados}
  fim para
  para cada comunidade k conectada somente a j **faça**
    $\Delta Q_{jk} \leftarrow \Delta Q_{jk} - 2a_i a_k$
  fim para
  $a_j \leftarrow a_j + a_i$
  $a_i \leftarrow 0$ {Pode remover da estrutura de dados}
  se $Q > Q_{max}$ **então**
    $Q_{max} \leftarrow Q$
    $C_{max} \leftarrow C$
  fim se
fim para
retorna C_{max}

Algoritmo 1: Algoritmo guloso para particionamento de grafos em comunidades

grandes comunidades ao escolher o par de maior aumento de modularidade. Dessa forma, o dendrograma formado pela união das comunidades pode acabar com uma altura $d \sim n$, o que deixa o algoritmo com uma complexidade $O(mn \log n)$, o que torna o algoritmo inviável para grafos com milhões de vértices e arestas. Resultados experimentais (Wakita & Tsurumi [2007]) com grafos variados mostram que isso de fato acontece na maioria dos casos.

Uma possível solução para o fato do algoritmo dar preferência a grandes comunidades é a normalização pelo tamanho das comunidades no cálculo do incremento de modularidade ΔQ. O trabalho de Wakita & Tsurumi [2007] propõe a normalização por um fator dependente do tamanho de ambas comunidades em uma determinada entrada da matriz ΔQ. Esse fator é definido da seguinte forma:

$$f(c_i, c_j) = min(|c_i|/|c_j|, |c_j|/|c_i|)$$

Onde c_i é a comunidade i, e $|c_i|$ é o tamanho da comunidade i em quantidade de vértices (número de vértices pertencentes aquela comunidade). Podemos ver que a função retorna valores no intervalo $(0, 1]$, onde o valor máximo 1 é atingido quando as comunidades tem o mesmo tamanho. Essa normalização deixa o algoritmo mais eficiente, e frequentemente resulta em melhores valores de modularidade do que os encontrados pelo algoritmo de Clauset et al. [2004]. O algoritmo com as modificações de Wakita & Tsurumi [2007] é o utilizado como base para o algoritmo proposto nesse trabalho.

Outra modificação proposta com o intuito de evitar a formação de grandes comunidades é a do trabalho por Danon et al. [2006], que sugere normalizar a variação de modularidade ΔQ produzida pela união de duas comunidades pela fração de arestas incidentes a uma das comunidades, para favorecer pequenas comunidades. De acordo com o autor isso leva a melhores valores de modularidade do que o algoritmo por Newman [2004a]. Mais uma modificação proposta para evitar grandes comunidades é a por Schuetz & Caflisch [2008a,b], que consiste em permitir a união de vários pares de comunidades em cada passo do algoritmo, ao invés de somente um par. Isso gera vários "centros"ao redor dos quais as comunidades são formadas, crescendo simultaneamente e tornando improvável a formação de poucas comunidades muito grandes. O método final tem a mesma complexidade que o algoritmo de Clauset et al. [2004], mas retorna maiores valores de modularidade.

De acordo com vários trabalhos, uma modificação que pode melhorar bastante a otimização gulosa é a utilização de outra configuração inicial de estrutura de comunidades, ao invés de uma comunidade por vértice (Du et al. [2007]; Pujol et al. [2006]; Xiang

et al. [2009]; Ye et al. [2008]). A configuração inicial pode ser obtida unido-se os vértices isolados em subgrafos maiores, de acordo com uma medida de similaridade topológica entre subgrafos. Outra configuração inicial possível é uma tal que nenhum vértice pode ser movido de sua comunidade para outra sem diminuir o valor da modularidade.

1.5 Grafos multicamada

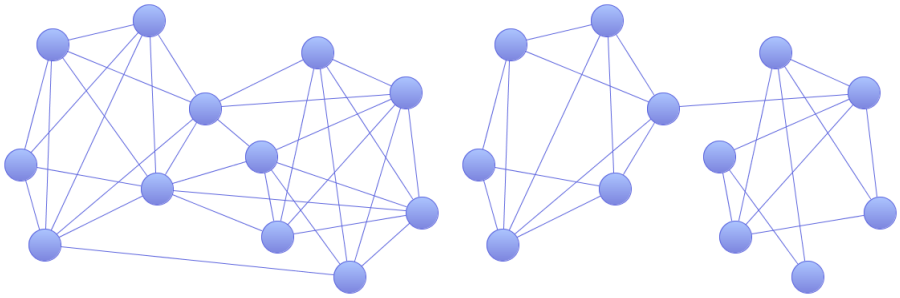
Um grafo multicamada é basicamente um conjunto de grafos. Cada grafo pode representar diferentes estruturas de conexão (ou seja, pode ter arestas com diferentes significados), diferentes instantes de tempo (ou seja, cada grafo é uma representação em um dado momento de um grafo dinâmico) ou outras características distintas. Normalmente, os grafos não precisam ter o mesmo conjunto de vértices, mas podemos simplificar e assumir que sempre terão o mesmo conjunto de vértices, onde esse conjunto seria a união dos conjuntos de vértices de todos os grafos.

Dessa forma, teremos que um grafo multicamada é um grupo de grafos baseados no mesmo conjunto de vértices, onde cada grafo pode ter um tipo diferente de aresta (amizade, colaboração, interação, disputa) ou podem ter o mesmo tipo de aresta mas existentes em diferentes instantes de tempo. Cada grafo do grafo multicamada é chamado de grafo componente.

Definição Seja V um conjunto de vértices e E_1, E_2, E_3, ..., E_k alguns conjuntos de arestas. Um grafo multicamada G é o conjunto de grafos $G = \{G_1(V, E_1), G_2(V, E_2), ..., G_k(V, E_k)\}$. Os grafos G_i são chamados grafos componentes. O número de grafos componentes k é chamado de multiplicidade do grafo.

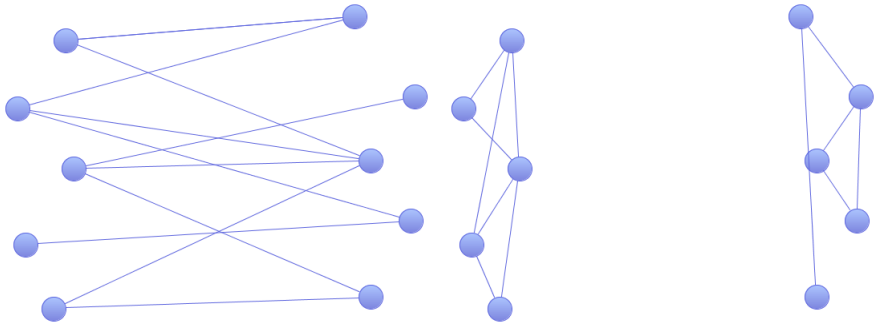
Na figura 1.2 temos um exemplo de um grafo multicamada composto de 2 grafos: um grafo de amizade (figura 1.2a) e um grafo de interação (figura 1.2b), onde o grafo de interações nesse caso é um subgrafo do grafo de amizade, indicando por exemplo que interações só podem ser feitas entre pessoas que tem uma relação de amizade de acordo com o grafo. Esse exemplo demonstra um possível mapeamento entre as redes sociais que temos hoje em dia para um grafo multicamada.

Na figura 1.3 temos outro exemplo de grafo multicamada. Nesse exemplo, temos dois grupos de vértices, onde os nós só são amigos de outros nós do mesmo grupo, e cada nó só faz disputas com nós do grupo adversário. A figura 1.3a mostra o grafo de disputas, e a figura 1.3b mostra o grafo de amizade.



(a) Grafo de amizade (b) Grafo de interação

Figura 1.2: Exemplo de grafo multicamada 1



(a) Grafo de disputa (b) Grafo de amizade

Figura 1.3: Exemplo de grafo multicamada 2

1.5.1 Modularidade em grafos multicamada

Recentemente foi desenvolvida uma função de qualidade baseada na modularidade original, para grafos multi-slice, multicamada ou multi-scale (Mucha et al. [2010]). A função utilizada é genérica e permite uso não só em grafos multicamada, como em grafos multislice genéricos, com conjuntos de nós distintos, com vários graus de acoplamento entre grafos (função de acoplamento que define a conexão que o nó tem com ele mesmo em outros grafos), fator de resolução (para normalizar pela resolução do grafo) e diferentes particionamentos para cada grafo envolvido.

$$Q_m = \frac{1}{2\mu} \sum_{ijrs} \left[\left(A_{ijs} - \gamma_s \frac{k_{is} k_{js}}{2m_s} \delta_{sr} \right) + \delta_{ij} C_{jsr} \right] \delta(g_{is}, g_{jr})$$

Onde temos:

- A_{ijs} - valor que indica existência de aresta; tem valor 1 ou 0 se existe ou não um aresta entre i e j no grafo componente s.
- k_{is} - grau do vértice i no componente s.

- m_s - número de arestas no componente s.
- γ_s - fator de resolução do componente s.
- C_{jsr} - acoplamento do vértice j entre os componentes s e r.
- g_{is} - comunidade do vértice i no componente s.
- μ - fator de normalização: $2\mu = \sum_{is} (k_{is} + c_{is})$, onde $c_{is} = \sum rC_{isr}$, ou seja, a soma dos acoplamentos do vértice i no componente s com os outros componentes.

O particionamento utilizado no trabalho citado é feito em vários níveis assim como o grafo, ou seja, para cada grafo componente do grafo multicamada, há um particionamento diferente, significando que dependendo do tipo de conexão, diferentes comunidades podem existir, mesmo se tratando dos mesmos vértices. Esse tratamento é esperado, visto que diferentes tipos de grafos, mesmo se tratando dos mesmos vértices, podem gerar diferentes tipos de comunidades. Por exemplo, se tivermos um grafo de amizade, teremos um particionamento em comunidades, mas se tivermos um grafo de trabalho (arestas entre nós que já trabalharam juntos), podemos ter outro particionamento muito diferentes.

Uma vez definida a fórmula de modularidade multicamada, pode-se usar algoritmos tradicionais de otimização para achar estruturas de comunidades que maximizam essa fórmula. No entanto, esses algoritmos não permitem a utilização em grafos com muitos nós pois tem alta complexidade (exponencial em muitos casos). No trabalho em que foi desenvolvida a fórmula, esses algoritmos são utilizados para grafos com menos de 2 mil vértices somente, um grafo com mais de 100 grafos componentes e outro grafo com 4 grafos componentes.

Assim, não existem métodos ainda para detecção de comunidade em grafos multicamada que possam ser utilizados com grafos muito grandes, da ordem de milhões de nós. Esse trabalho propõe um novo algoritmo para esse fim, que pode ajudar na análise desse tipo de grafo de grande escala.

Capítulo 2

Desenvolvimento

Este capítulo mostra como simplificamos a fórmula de modularidade multicamada para obter um maior entendimento de como ela funciona, e como isso levou ao desenvolvimento do novo algoritmo para particionamento em comunidades de grafos multicamada. Ao utilizar tipos de grafos multicamada mais simples, a fórmula de modularidade multicamada pode ser reduzida e dividida em partes que resultam em componentes similares aos utilizados na fórmula de modularidade original (como e_{ii} e a_i). Assim, o novo algoritmo é formulado com base a otimizar os componentes como utilizados na nova fórmula, a fim de maximizar a modularidade multicamada ao invés da modularidade comum.

O algoritmo final tem uma complexidade de tempo de execução com um aumento da ordem de k (a multiplicidade do grafo multicamada) em relação ao algoritmo guloso original, um aumento esperado e aceitável em se tratando de grafos multicamada. Poucas modificações são necessárias no algoritmo original, o que torna a implementação simples, e o algoritmo eficiente, pois já havia sido otimizado através das várias técnicas já explicadas.

2.1 Modularidade em grafos multicamada

A fórmula de modularidade multicamada, definida para tipos genéricos de grafos multicamada:

$$Q_m = \frac{1}{2\mu} \sum_{ijrs} \left[\left(A_{ijs} - \gamma_s \frac{k_{is} k_{js}}{2m_s} \delta_{sr} \right) + \delta_{ij} C_{jsr} \right] \delta(g_{is}, g_{jr})$$

Como já vimos, essa fórmula trata de uma ampla variedade de grafos multicamada, e define resultados também mais complexos, como um particionamento por

grafo componente. Nesse trabalho nós simplificamos o tipo de grafos sendo utilizados, e utilizamos um particionamento único, ou seja, nós procuramos somente por "comunidades globais", que são comunidades bem conectadas em todos aspectos do grafo multicamada (amizade, interação, trabalho, etc). Pode-se dizer que esse tipo de particionamento divide os nós em comunidades, levando em conta todas informações dadas pelo grafo multicamada. Dessa forma, as variáveis que representam a atribuição de uma comunidade a um determinado nó em um algum grafo componente podem ser simplificadas como a atribuição de uma comunidade "global"para um vértice somente (e portanto g_{is} pode ser simplificado como g_i - a comunidade do vértice i).

Outra simplificação feita está no uso de uma resolução constante para o parâmetro γ. Se assumirmos que todos grafos são distintos e no mesmo nível (todos nós presentes e representando somente um nó), podemos fixar esse parâmetro com valor unitário para todos grafos componentes. O uso de grafos multicamada com componentes sendo o mesmo grafo em diferentes resoluções está fora do escopo desse trabalho e é deixado como trabalho futuro.

Com base nessas hipóteses, nós podemos reformular a função de modularidade. Como a resolução está fixada para todos grafos componentes com valor unitário, temos que $\gamma_s = 1$, e como temos somente um particionamento para todos grafos componentes, $g_{is} = g_i$ e $g_{jr} = g_j$. Dessa forma temos a seguinte fórmula simplificada:

$$Q_m = \frac{1}{2\mu} \sum_{ijrs} \left[\left(A_{ijs} - \frac{k_{is}k_{js}}{2m_s}\delta_{sr} \right) + \delta_{ij}C_{jsr} \right] \delta(g_i, g_j)$$

Dividindo os termos do somatório de forma a termos dois somatórios separados, teremos:

$$Q_m = \frac{1}{2\mu} \left[\sum_{ijrs} \left(A_{ijs} - \frac{k_{is}k_{js}}{2m_s}\delta_{sr} \right) \delta(g_i, g_j) + \sum_{ijrs} \delta_{ij}C_{jsr}\delta(g_i, g_j) \right]$$

Como δ_{ij} é diferente de zero somente quando $i = j$, então $\delta(g_i, g_j)$ só será calculado quando $i = j$, e então será sempre igual a 1: $\delta(g_i, g_j) = \delta(g_i, g_i) = 1$. Portanto, o valor dessa parte será sempre o mesmo, não importando o particionamento. Agora iremos chamar de α a parte restante da fórmula, sem a normalização por $\frac{1}{2\mu}$. Então temos:

$$\alpha = \sum_{ijrs} \left(A_{ijs} - \frac{k_{is}k_{js}}{2m_s}\delta_{sr} \right) \delta(g_i, g_j)$$

Separando o somatório primeiro pelos pares de grafos componentes e depois pelas arestas:

$$\alpha = \sum_{rs} \sum_{ij} \left(A_{ijs} - \frac{k_{is} k_{js}}{2m_s} \delta_{sr} \right) \delta(g_i, g_j)$$

Agora podemos reformular o somatório interno como um somatório sobre comunidades, já que a função $\delta(g_i, g_j)$ só será 1 quando i e j estão na mesma comunidade, e então os únicos componentes da soma que serão adicionados no valor final serão quando os pares estiverem na mesma comunidade. A passagem aqui é semelhante ao que foi feito a partir da fórmula 1.5 na seção 1.3.

$$\alpha = \sum_{rs} \left[m_s \sum_{c} \left(e_{scc} - a_{sc}^2 \delta_{sr} \right) \right]$$

Aqui, podemos ver que essa fórmula se assemelha muito à fórmula de modularidade para grafos comuns. O somatório interno, exceto pelo fator δ_{sr}, é o mesmo que a fórmula alternativa de modularidade, nesse caso, a modularidade do grafo componente s. O somatório externo soma esse valor para todos pares de grafos componentes. Isso sugere que um algoritmo guloso, como o para grafos comuns, mas usando ao invés da tabela ΔQ, uma tabela que seria a soma das estruturas ΔQ de todos grafos componentes, poderia dar melhores resultados com relação a modularidade multicamada do que aplicar o algoritmo de grafos convencionais em algum dos grafos componentes. Nós fazemos dessa ideia a base de nosso algoritmo, como veremos adiante.

2.2 Algoritmo

Nosso algoritmo é baseado no algoritmo guloso para otimização de modularidade que nós apresentamos na seção 1.4.1, com a modificação de normalização pelos tamanhos das comunidades sugerida por Wakita & Tsurumi [2007]. Como já vimos na seção 2.1, a parte da fórmula de modularidade multicamada que é dependente do particionamento do grafo é basicamente uma soma dos mesmos componentes que os da fórmula convencional de modularidade (como $e_i i$ e a_i). Portanto, no nosso algoritmo, nós somente modificamos a parte da inicialização, construindo a tabela ΔQ e o vetor a utilizando todas arestas de todos grafos componentes, o que é o equivalente a construir as tabelas ΔQ e os vetores a para cada grafo componente, e então somar todos em uma única tabela e vetor. Assim esperamos otimizar a modularidade multicamada sem precisar reformular todo o algoritmo para a nova fórmula. No algoritmo 2 podemos ver a nova parte de inicialização:

A etapa de aglomeração é idêntica ao algoritmo original, ou seja, é o mesmo al-

```
{Entrada: Grafo multicamada G (com componentes g_0, g_1, ..., g_k - cada um com n vértices e m_s arestas)}
{Variáveis: matriz ΔQ e vetor a_i globais}
{Inicializações}
para s = 1 a k faça
    para v = 1 a n faça
        para cada vizinho w de v no grafo componente s faça
            ΔQ_vw ← ΔQ_vw + (1/(2m_s) − k_sv k_sw/(2m_s)^2)
        fim para
        a_v ← a_v + (k_sv/(2m_s))
    fim para
fim para
```

Algoritmo 2: Etapa de inicialização do algoritmo guloso para particionamento de grafos multicamada em comunidades

goritmo, só alteramos a inicialização das estruturas de dados, no caso a matriz/tabela ΔQ e o vetor a. Porém, como a matriz ΔQ é bem mais densa nesse caso, pelo fato de ter entradas para todas arestas de todos grafos componentes, o tempo de execução é afetado, como veremos na seção 2.2.1 sobre a complexidade do algoritmo. Ao agregar todas informações na mesma tabela ΔQ, o algoritmo tende a maximizar a soma das modularidades dos grafos componentes, o que faz com que indiretamente a modularidade multicamada também tenda a crescer. Apesar dos coeficientes serem um pouco diferentes na fórmula de modularidade multicamada, o algoritmo bem simplificado como foi descrito já consegue otimizar bem a modularidade multicamada, sem precisar de um algoritmo mais complexo, e com um aumento de complexidade de tempo não além do esperado (somente da ordem da multiplicidade do grafo multicamada).

2.2.1 Complexidade

A complexidade do tempo de execução do algoritmo guloso original é dada por $O(md \log n)$, onde m é o número de arestas no grafo. Esse análise é feita com base na densidade da tabela ΔQ, que é inicializada com uma entrada para cada aresta do grafo. No caso do algoritmo para grafos multicamada, temos que a inicialização é feita adicionando uma entrada (ou incrementando uma entrada) para cada aresta em cada grafo componente do grafo multicamada. Assim, seria como se tivéssemos um grafo muito mais denso, com mais arestas, e a tabela ΔQ seria criada com essa densidade. Se o grafo componente s tem m_s arestas, então no grafo multicamada temos um total de $\sum_s m_s$ arestas. Assim temos a seguinte complexidade para o algoritmo multicamada:

$$O((\sum_s m_s) d \log n)$$

Se o número de arestas em cada grafo componente for limitado, por exemplo se tivermos $m_s = O(m) \forall s$, e sendo k a multiplicidade do grafo multicamada, podemos fazer:

$$\sum_s m_s = \sum_s O(m) = kO(m) = O(km)$$

Assim, temos que se $m_s = O(m)$, então $\sum_s m_s = O(km)$, ou seja, se o número de arestas em cada grafo componente é limitado por m, então a soma é limitada por km, e portanto teremos a seguinte complexidade:

$$O((\sum_s m_s) d \log n) = O(kmd \log n)$$

Onde k é a multiplicidade do grafo multicamada, m é o limite assintótico do número de arestas em cada grafo componente, d é a altura do dendrograma gerado durante a execução do algoritmo, e n é o número de vértices do grafo multicamada. Portanto, temos que a complexidade do tempo de execução para o algoritmo multicamada é k vezes a complexidade do algoritmo guloso normal, e podemos esperar então que o tempo de execução do algoritmo multicamada seja maior na ordem de k vezes.

Capítulo 3

Metodologia experimental

Vários experimentos foram feitos com a intenção de analisar o desempenho e a qualidade da modularidade multicamada do algoritmo. Os dados coletados são o tempo de execução dos algoritmos (dividido em tempo de inicialização e tempo de aglomeração) e o valor da modularidade multicamada resultante do particionamento do conjunto de vértices do grafo, que é a saída dos algoritmos. Assim podemos comparar os resultados de ambos algoritmos, tanto em relação ao desempenho quanto à qualidade do resultado, para analisarmos a viabilidade, validade e valor do novo algoritmo proposto.

Para avaliar os algoritmos é necessária a utilização de vários grafos multicamada como entrada, com vários tamanhos, diferentes estruturas de comunidades e várias multiplicidades (número de grafos componentes variado). Para esse fim, foi desenvolvido um gerador de grafos, que recebe vários parâmetros como entrada e gera um grafo com essas características. Assim, foi possível gerar uma grande variedade de grafos para avaliar os algoritmos. Na seção 3.1 temos mais detalhes sobre a geração de grafos.

Uma vez que tínhamos disponibilidade de vários grafos de entrada, vários experimentos foram executados para avaliar os algoritmos sob vários aspectos. Na seção 3.2 temos detalhes sobre os experimentos executados para vários tamanhos de grafos. Nesses experimentos, foram utilizados grafos de tamanhos variando desde algumas dezenas até milhões de vértices, para analisarmos o desempenho e qualidade dos algoritmos conforme o número de nós do grafo aumentava.

Na seção 3.3 são explicados os experimentos sobre a variação de multiplicidade do grafo multicamada, ou seja, os experimentos com grafos compostos de vários números de grafos componentes. Assim podemos analisar como o algoritmo se comporta quando o grafo multicamada se torna mais complexo, do ponto de vista do tempo gasto para execução e qualidade da resposta final.

3.1 Geração de grafos

Nessa seção explicaremos como foi feita a geração dos grafos de teste. Vários parâmetros foram definidos para serem utilizados pelo gerador de grafos, como o número de vértices, número de comunidades, probabilidade de arestas intracomunidade e intercomunidade. Gera-se então um grafo com essas características e arestas uniformemente distribuídas (dentro das probabilidades especificadas).

A primeira etapa de geração de um grafo de comunidades trata da construção da estrutura de comunidades. Os vértices são uniformemente distribuídos entre as comunidades. No caso dos grafos multicamada, cada grafo componente tem uma estrutura de comunidades com 10% dos vértices com comunidades diferentes do grafo componente anterior. Assim podemos gerar grafos componentes com distribuições variadas de arestas.

Uma vez definidas as comunidades de cada vértice em cada grafo componente, cada grafo componente é gerado separadamente, seguindo os mesmos parâmetros. Em cada grafo, para cada vértice calculamos o número de vizinhos dentro e fora da comunidade, a partir das probabilidades de arestas dadas como entrada (com uma variação uniforme de $\pm 40\%$), e então escolhemos vizinhos uniformemente distribuídos entre os vértices da mesma comunidade e os vértices fora. Uma vez feito isso para todos nós do grafo, a geração do grafo está completa.

Em todos experimentos, para cada tipo de grafo (um dado tamanho e multiplicidade) foram gerados 30 grafos daquele tipo. O número de comunidades foi usado como sendo 3,75% do tamanho do grafo, com uma variação uniforme de $\pm 25\%$. A probabilidade de arestas entre nós da mesma comunidade é distribuída uniformemente entre 0,20 e 0,40 (entre 20% e 40%). Já a probabilidade entre vértices de comunidades distintas é calculada de forma que os nós tenham em média k_m vizinhos, e portanto depende do tamanho do grafo, da probabilidade de arestas intracomunidade e do número de comunidades. A fórmula é aproximadamente $p_{inter} = \frac{k_m}{N(1-\frac{1}{C})} - \frac{p_{intra}}{C-1}$, onde N é o número de vértices (tamanho do grafo), C é o número de comunidades, p_{intra} é a probabilidade de arestas intracomunidade e k_m é o grau médio dos nós.

3.2 Tamanho do grafo

Os primeiros experimentos realizados tratam da variação no número de vértices nos grafos. Foram gerados grafos com 20, 200, 2000, 20000, 200000 e 2000000 de vértices, e com multiplicidades de 2, 3 e 4 grafos componentes. Para cada tamanho e multiplicidade, foram gerados 30 grafos. Os experimentos foram realizados separadamente, uma

vez para cada multiplicidade. Em cada experimento, para cada grafo foi executado o algoritmo multicamada para o grafo completo e o algoritmo guloso original para cada grafo componente. A saída de cada algoritmo é o particionamento do conjunto de vértices, ou seja, uma lista onde para cada vértice temos a sua comunidade. Para cada particionamento (do algoritmo multicamada e do algoritmo guloso original para cada grafo componente), é calculada a modularidade multicamada do grafo.

A análise da qualidade da solução foi feita comparando-se a melhor modularidade multicamada dada pelos particionamentos do algoritmo guloso original com a modularidade da solução dada pelo algoritmo multicamada. Dessa forma podemos verificar se o algoritmo multicamada gera particionamentos melhores (em termos de modularidade multicamada) do que o algoritmo guloso original em qualquer grafo componente. O ganho percentual foi então calculado, e foi feita a média entre os 30 grafos do mesmo tipo. Foi feita também uma análise do tempo de execução dos algoritmos, onde para cada execução, foi feita a média e a soma dos tempos do algoritmo guloso original para os grafos componentes, e foi calculada a razão do tempo do algoritmo multicamada sobre essa média e essa soma. Assim podemos saber o quão lento o algoritmo multicamada é em relação ao algoritmo guloso original.

A tabela 3.1 mostra alguns dados sobre os grafos utilizados nesses experimentos.

3.3 Multiplicidade do grafo

Os experimentos seguintes foram feitos sobre a variação da multiplicidade do grafo multicamada. O número de grafos componentes utilizados foi de 2 a 10. O tamanho de grafo foi fixado para 200 mil vértices, e para cada multiplicidade foram gerados 30 grafos deste tamanho. No experimento, para cada grafo foi executado o algoritmo multicamada para o grafo completo e o algoritmo guloso original para cada grafo componente, como no experimento anterior. Dadas as saídas, que são os particionamentos dos conjuntos de vértices, foram calculadas as modularidades multicamada de cada grafo, para cada particionamento existente para aquele grafo (um particionamento dado pelo algoritmo multicamada, e um particionamento para cada grafo componente dado pelo algoritmo guloso original). Com isso pode-se verificar se o novo algoritmo é consistentemente melhor para diferentes multiplicidades.

A análise foi feita de forma semelhante ao experimento anterior, onde a análise da qualidade da solução foi feita comparando-se a modularidade multicamada do algoritmo multicamada com a melhor modularidade encontrada pelo algoritmo guloso original com os grafos componentes. Então é calculada a média e o desvio padrão do ganho

Tabela 3.1: Dados sobre os grafos utilizados nos experimentos sobre tamanho dos grafos

Quantidade de arestas

Tamanho	*k=2*	*k=3*	*k=4*
20	95,7±9,32	97,3±7,21	96,3±8,72
200	1261±86,1	1258±76,9	1250±81,3
2000	12127±572	12103±564	12084±571
20000	120530±5127	120701±5351	121315±4412
200000	1181899±46829	1203316±52932	1201383±54431
2000000	12199813±487760	11992531±540033	12137030±554668

Grau médio dos vértices

Tamanho	*k=2*	*k=3*	*k=4*
20	9,57±0,932	9,73±0,723	9,63±0,877
200	12,6±0,861	12,6±0,769	12,5±0,813
2000	12,1±0,573	12,1±0,564	12,1±0,572
20000	12,1±0,513	12,1±0,535	12,1±0,441
200000	11,8±0,468	12,0±0,529	12,0±0,544
2000000	12,2±0,488	11,1±0,540	12,1±0,555

(absoluto e percentual) de modularidade, e o resultado é mostrado para a variação na multiplicidade. Foi feita também a análise do tempo de execução assim como no experimento anterior, onde foi calculada a média e desvio padrão do tempo de execução, e a média e desvio padrão da razão do tempo do algoritmo multicamada sobre o tempo do algoritmo guloso original.

Na tabela 3.2 temos algumas estatísticas sobre os grafos utilizados nesse experimento.

Tabela 3.2: Dados sobre os grafos utilizados nos experimentos sobre multiplicidade dos grafos

Multiplicidade	*Quantidade de arestas por grafo componente*	*Grau médio dos vértices*
2	1207782±48854	12.08±0.49
3	1215857±51576	12.16±0.52
4	1202056±54699	12.02±0.55
5	1204943±47792	12.05±0.48
6	1200147±37048	12.00±0.37
7	1225181±49291	12.25±0.49
8	1214881±46482	12.15±0.46
9	1187147±48696	11.87±0.49
10	1218735±39275	12.19±0.39

Capítulo 4

Resultados

Os resultados são explicados nas seções a seguir. A primeira seção trata dos experimentos para comparação entre o algoritmo multicamada e o algoritmo guloso para grafos comuns, para vários tamanhos de grafo. Veremos como o algoritmo multicamada se comporta com relação ao tempo de execução e a qualidade da modularidade resultante para grafos de vários tamanhos (várias ordens de grandeza do número de vértices).

Na seção seguinte observamos o algoritmo no aspecto da multiplicidade do grafo multicamada, ou seja, da quantidade de grafos componentes. Analisamos o tempo de execução conforme o número de grafos componentes aumenta, assim como o valor de modularidade resultante. Assim podemos comparar os resultados obtidos com a complexidade esperada do algoritmo.

4.1 Tamanho do grafo

Nessa seção iremos analisar os resultados do algoritmo multicamada em comparação com o algoritmo guloso original para grafos de vários tamanhos. Primeiro iremos analisar os resultados para o grafo multicamada de $k = 2$ componentes, em seguida para $k = 3$ e por fim para $k = 4$. Os resultados mostrados são:

- Tempo de execução total dos algoritmos
- Tempo de execução da etapa de inicialização dos algoritmos
- Tempo de execução da etapa de aglomeração dos algoritmos
- Razão entre o tempo de execução total do algoritmo multicamada e o tempo do algoritmo guloso original para os grafos componentes - onde temos as razões

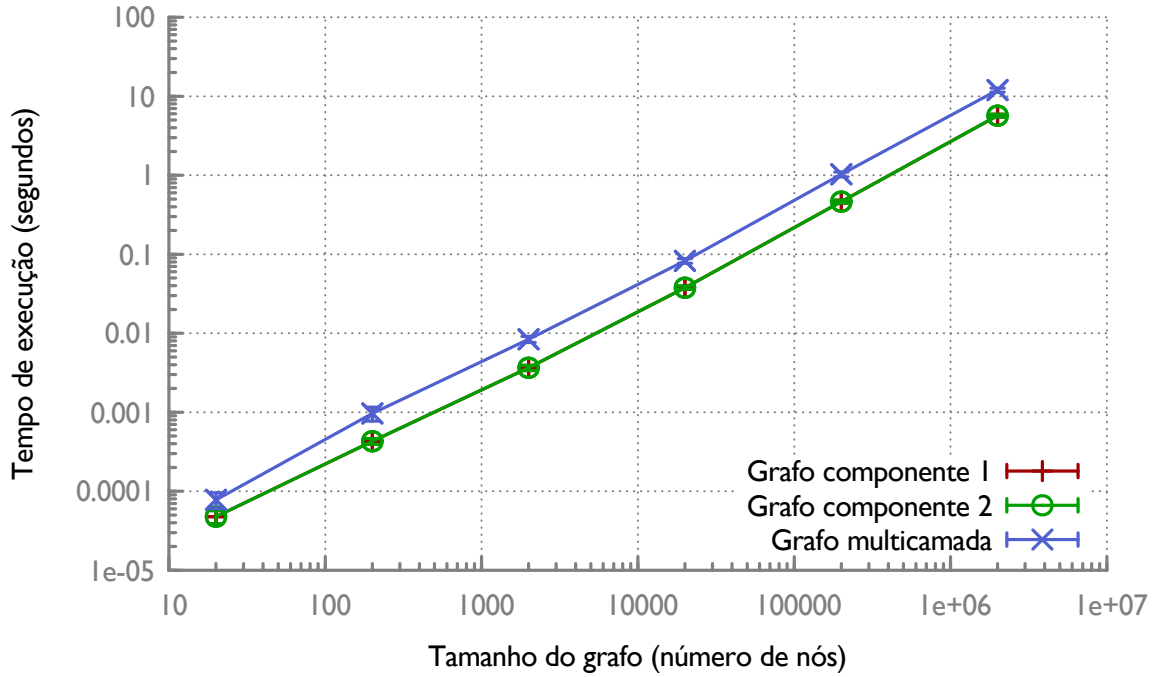


Figura 4.1: Gráfico do tempo de inicialização dos algoritmos para $k = 2$

contra a média e a soma dos tempos de execução do algoritmo guloso original para os grafos componentes

- Diferença entre o valor da modularidade multicamada encontrada pelo algoritmo multicamada e o melhor valor de modularidade encontrado pelo algoritmo guloso original

- Diferença percentual entre o valor da modularidade multicamada encontrada pelo algoritmo multicamada e o melhor valor de modularidade encontrado pelo algoritmo guloso original

Para todos pontos nos gráficos temos a média e a barra do desvio padrão para a execução dos algoritmos com 30 grafos distintos do mesmo tamanho. O tamanho dos grafos e os tempos de execução estão em escala logarítmica.

Nas figuras 4.1, 4.2 e 4.3 temos, respectivamente, os tempos de inicialização, de aglomeração e os tempos totais dos algoritmos. Como podemos ver, o desvio padrão é muito baixo, o que indica que o tempo de execução é determinado principalmente pelo número de vértices e arestas do algoritmo. A curva também sugere um tempo de execução linear, dado que se aproxima de uma reta com inclinação próxima de 1, o que está de acordo com as complexidades dos algoritmos.

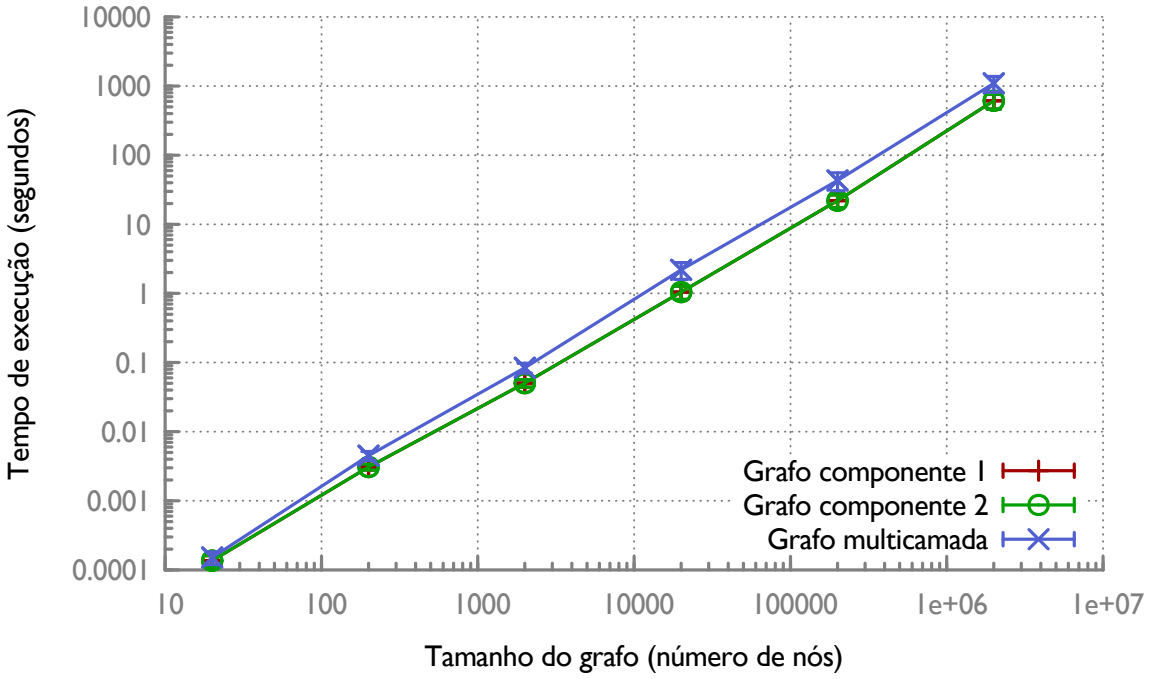


Figura 4.2: Gráfico do tempo de aglomeração dos algoritmos para $k = 2$

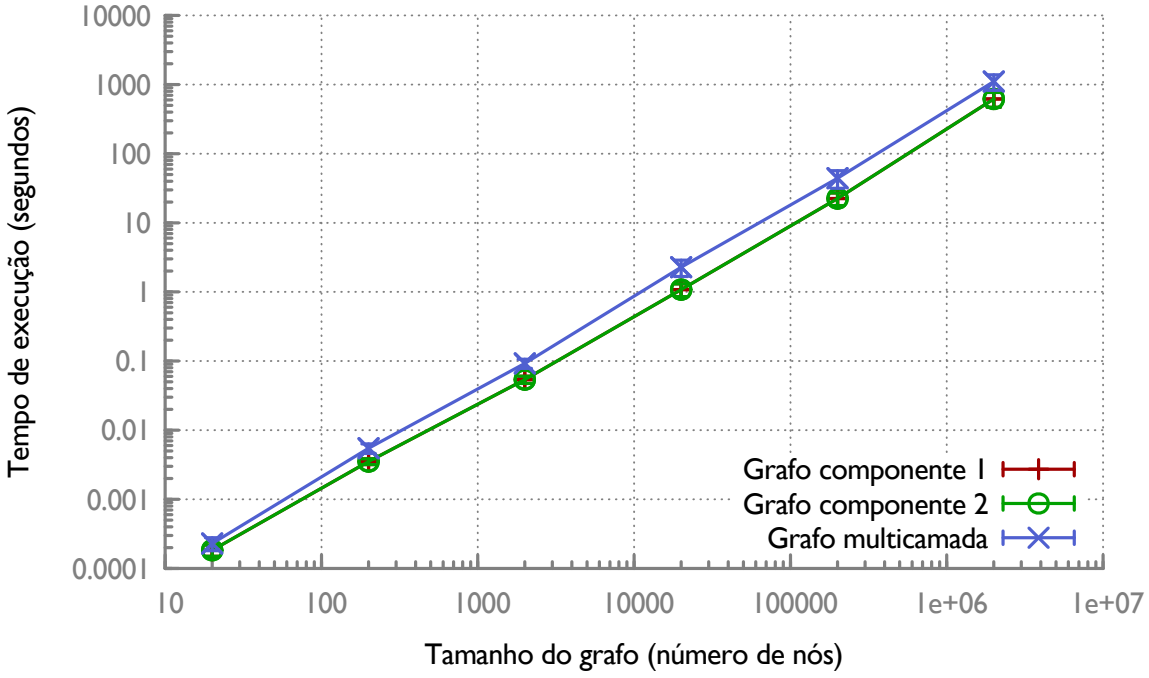


Figura 4.3: Gráfico do tempo total dos algoritmos para $k = 2$

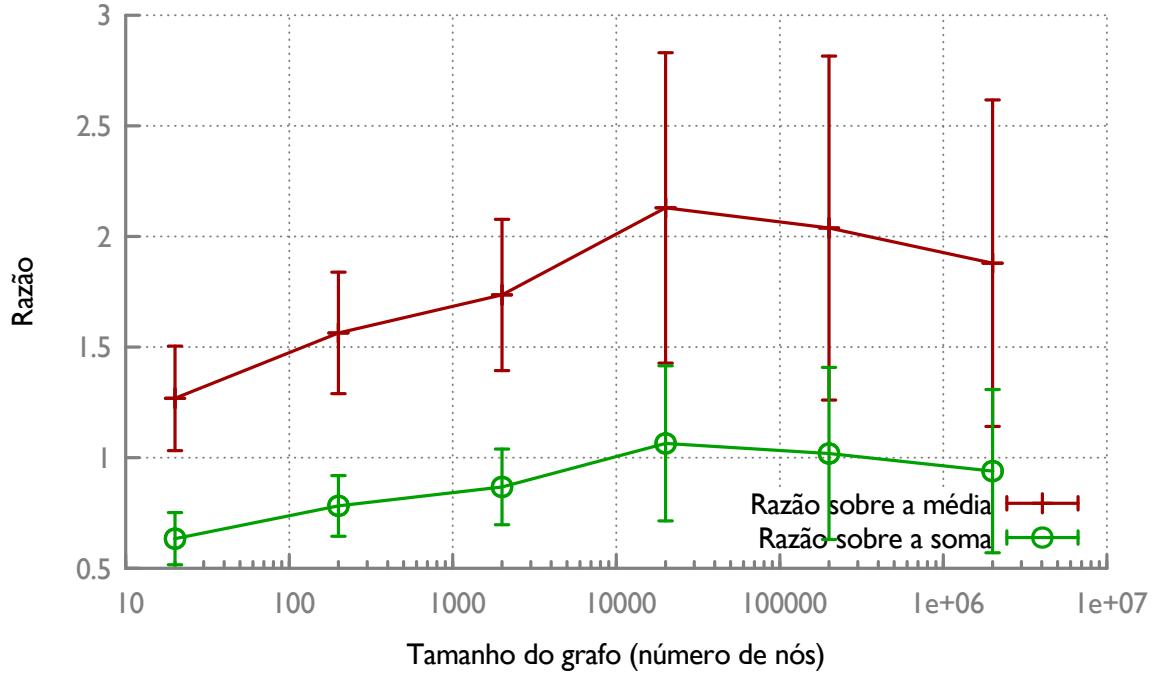


Figura 4.4: Gráfico da razão entre os tempos dos algoritmos para $k = 2$

Na figura 4.4, temos as razões entre os tempos de execução do algoritmo multicamada e as médias e somas dos tempos de execução do algoritmo guloso original. Podemos ver que para as razões sobre as médias do algoritmo guloso original, o valor varia em torno de 2 conforme o tamanho dos grafos aumenta, que é justamente a multiplicidade do grafo multicamada em questão. Já no caso da razão sobre a soma, temos o valor em torno de 1, ou seja, o tempo gasto pelo algoritmo multicamada é aproximadamente o tempo gasto para executar o algoritmo guloso para os dois grafos componentes. Para grafos pequenos (2000 ou menos vértices), o tempo de execução é menor que a soma, subindo gradativamente a medida que o tamanho do grafo aumenta.

Nas figuras 4.5 e 4.6 temos a diferença entre a modularidade multicamada encontrada pelo algoritmo multicamada e a melhor modularidade multicamada encontrada executando o grafo guloso original contra os grafos componentes, sendo que na primeira figura temos a diferença absoluta, enquanto na segunda temos a razão entre os mesmo valores (diferença percentual). Podemos observar que as curvas tem um comportamento parecido, o que se dá pelo fato da modularidade já ser um valor normalizado pelo tamanho do grafo. Pelo gráfico da diferença percentual, temos que o algoritmo multicamada apresenta um ganho em torno de 30% para a modularidade multicamada conforme aumentamos o tamanho dos grafos.

Nas figuras seguintes temos os resultados para os grafos multicamada com 3 grafos componentes. Os gráficos nas figuras 4.7, 4.8 e 4.9 mostram novamente os gráficos

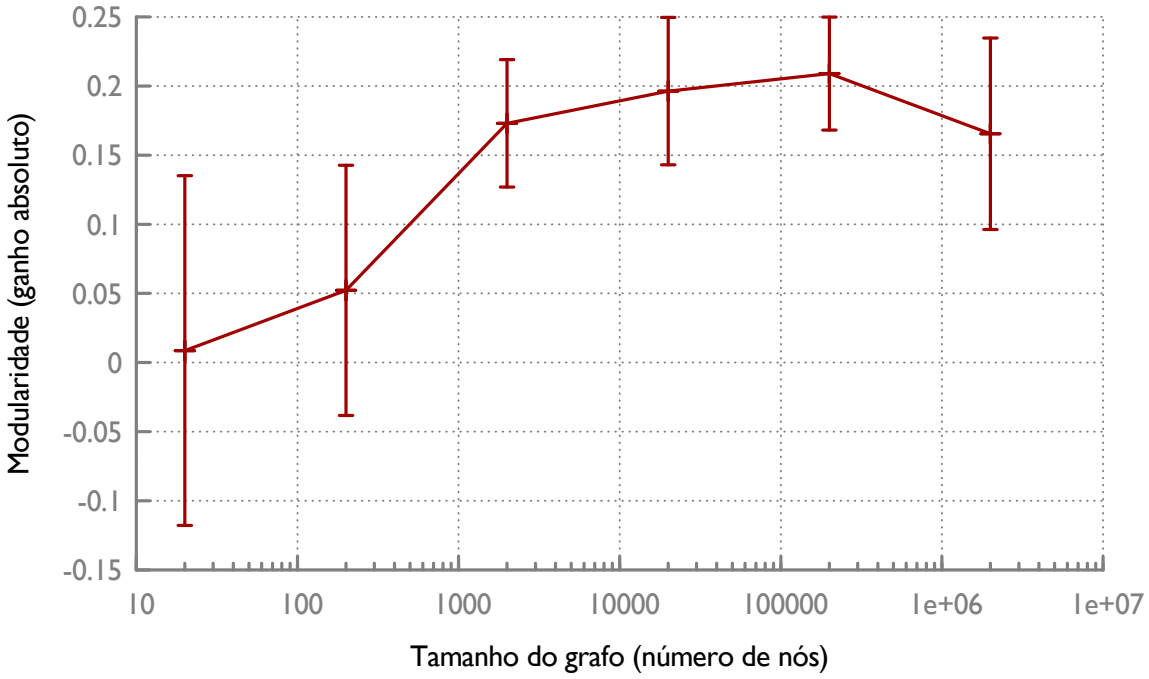


Figura 4.5: Gráfico da diferença de modularidade dos algoritmos para $k = 2$

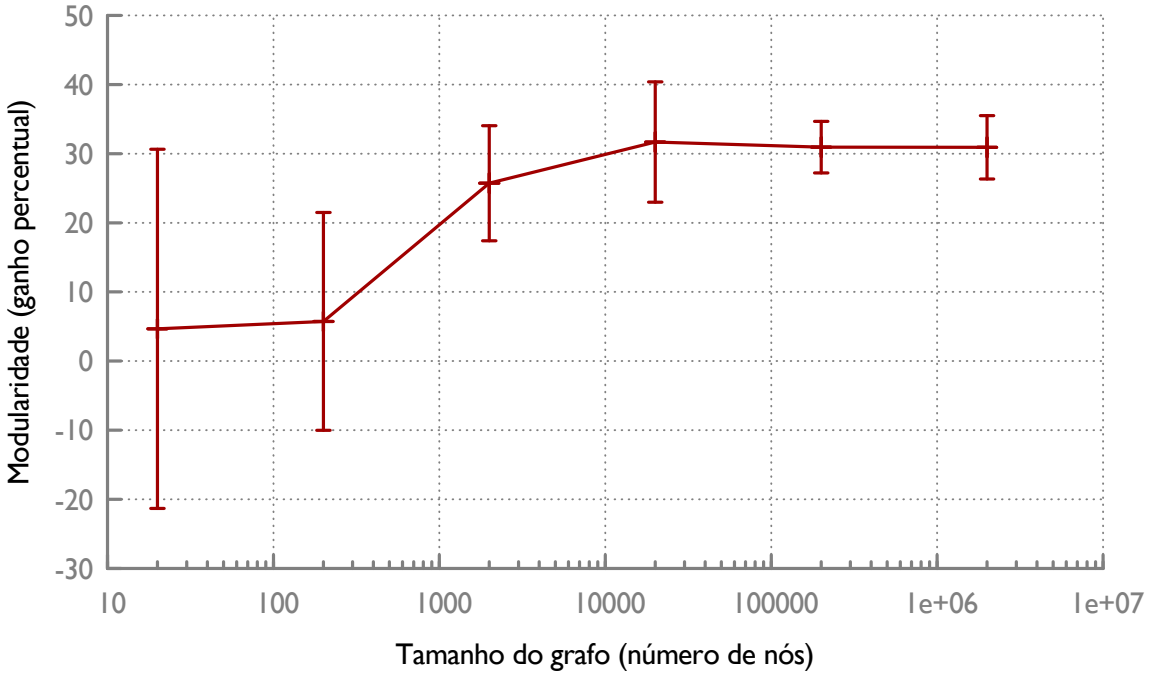


Figura 4.6: Gráfico da diferença percentual de modularidade dos algoritmos para $k = 2$

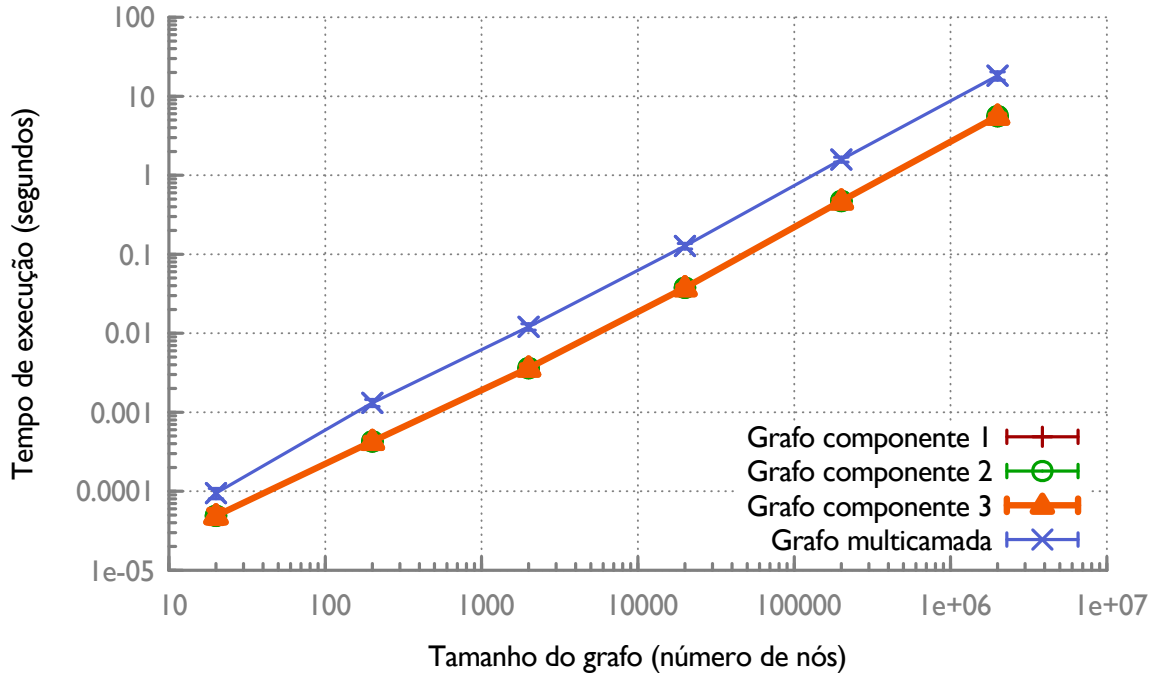


Figura 4.7: Gráfico do tempo de inicialização dos algoritmos para $k = 3$

dos tempos de inicialização, aglomeração e totais dos algoritmos, agora para o grafo multicamada de multiplicidade 3. Aqui temos um comportamento bem parecido, com as curvas se aproximando a uma reta, e o tempo do algoritmo multicamada acima do algoritmo guloso original em todos gráficos.

Na figura 4.10 temos o gráfico das razões entre os tempos de execução do algoritmo multicamada e do algoritmo guloso original. Temos aqui também um comportamento parecido, onde a razão sobre a média dos tempos é aproximadamente a multiplicidade do algoritmo multicamada ($k = 3$) e a razão sobre a soma é aproximadamente 1.

Nas figuras 4.11 e 4.12 temos agora os gráficos das diferenças de modularidade para os grafos multicamada com $k = 3$. Aqui temos um ganho um pouco maior na modularidade multicamada, com um ganho percentual médio entre 40% e 50% para os maiores grafos.

As figuras seguintes mostram os resultados para os grafos multicamada com 4 grafos componentes ($k = 4$). Os gráficos nas figuras 4.13, 4.14 e 4.15 mostram os tempos de inicialização, aglomeração, e totais dos algoritmos. Novamente, temos a mesma tendência, com a curva se aproximando a uma reta. Podemos ver que o algoritmo multicamada consegue executar para o grafo de 2 milhões de nós em aproximadamente 4000 segundos, ou aproximadamente 67 minutos, o que é um tempo bastante aceitável.

A figura 4.16 contém os resultados das razões dos tempos do algoritmo multicamada sobre os tempos do algoritmo guloso original. Aqui as razões apresentam um

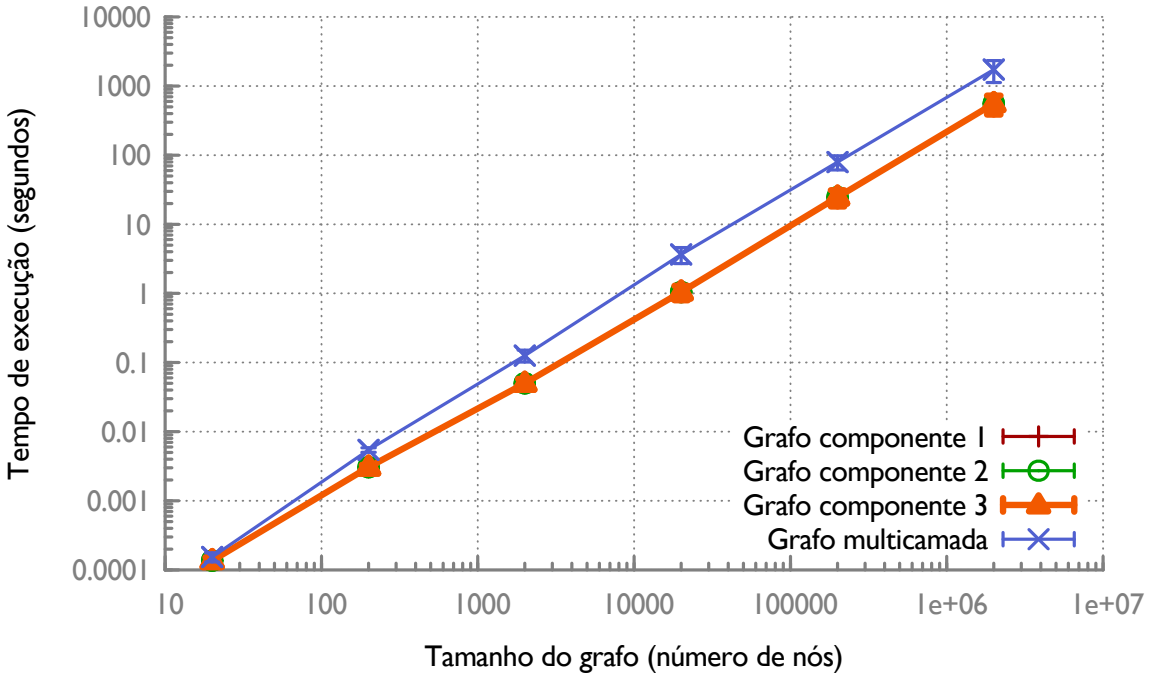


Figura 4.8: Gráfico do tempo de aglomeração dos algoritmos para $k = 3$

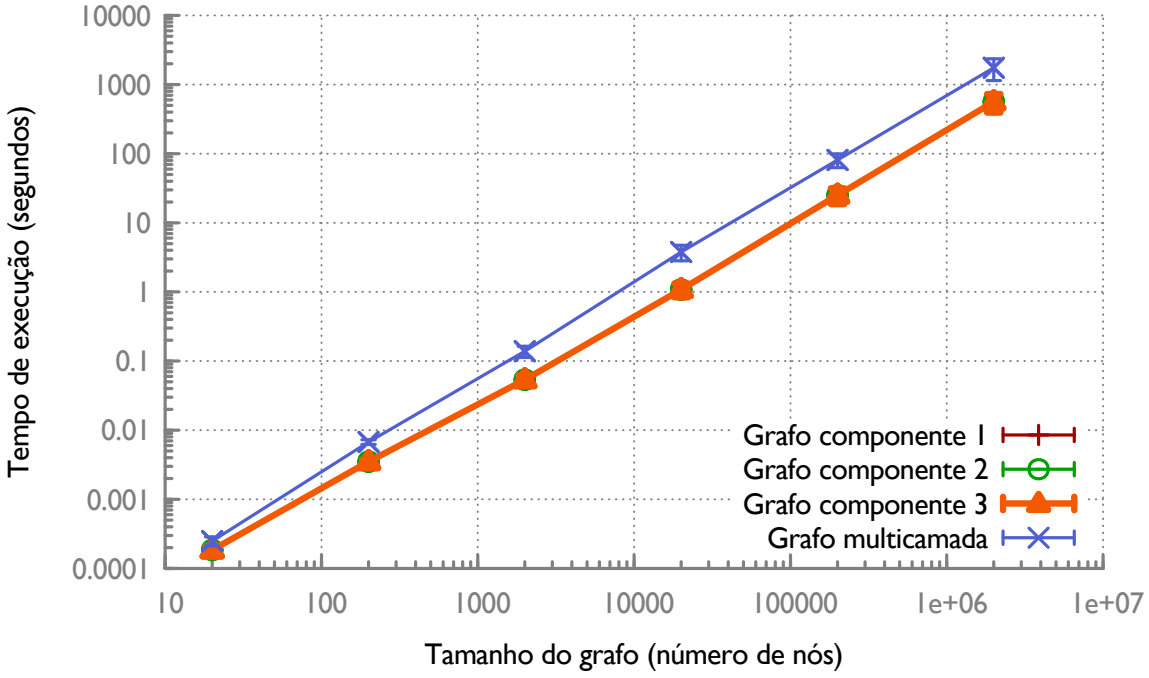


Figura 4.9: Gráfico do tempo total dos algoritmos para $k = 3$

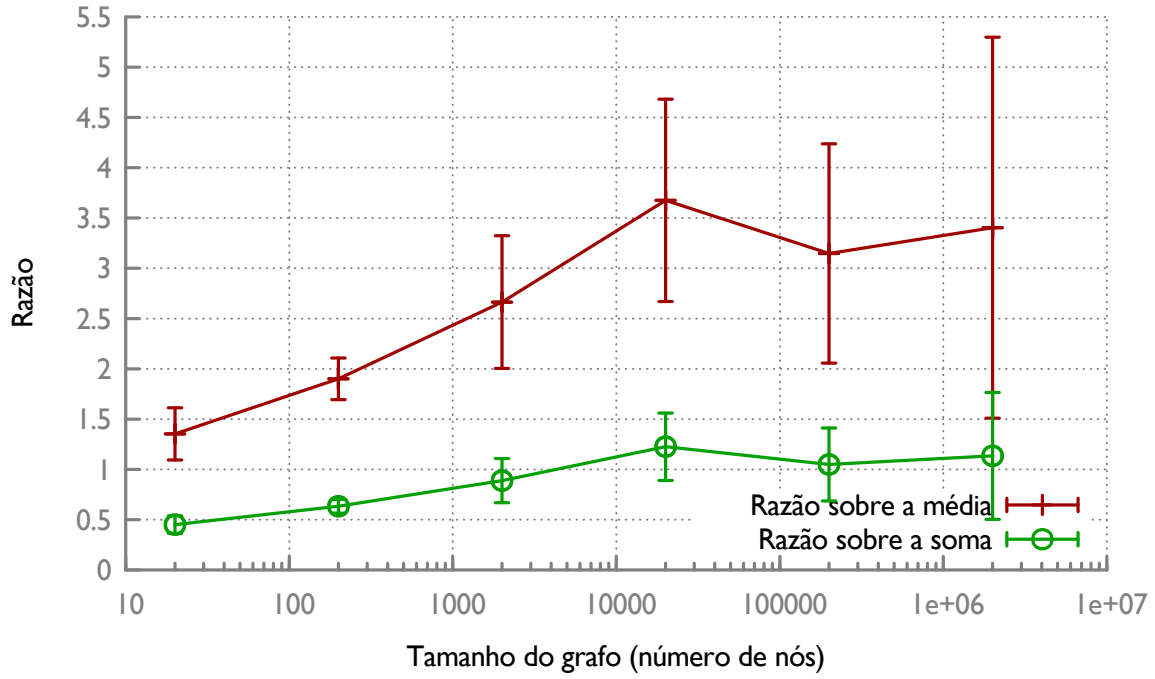


Figura 4.10: Gráfico da razão entre os tempos dos algoritmos para $k = 3$

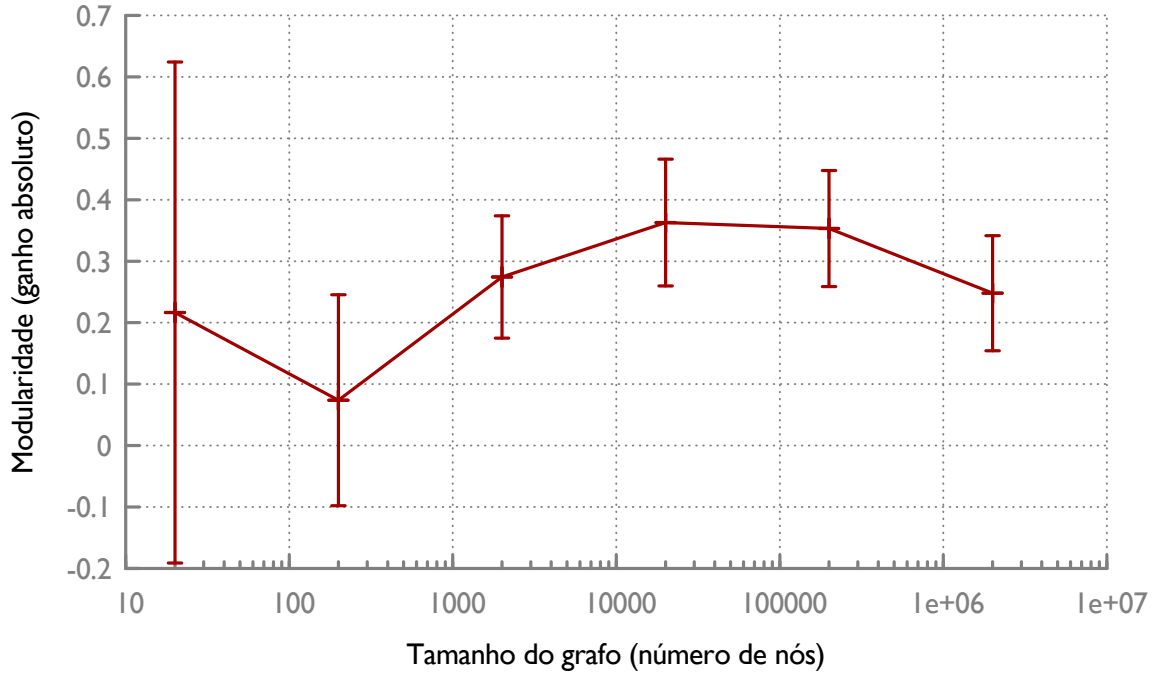


Figura 4.11: Gráfico da diferença de modularidade dos algoritmos para $k = 3$

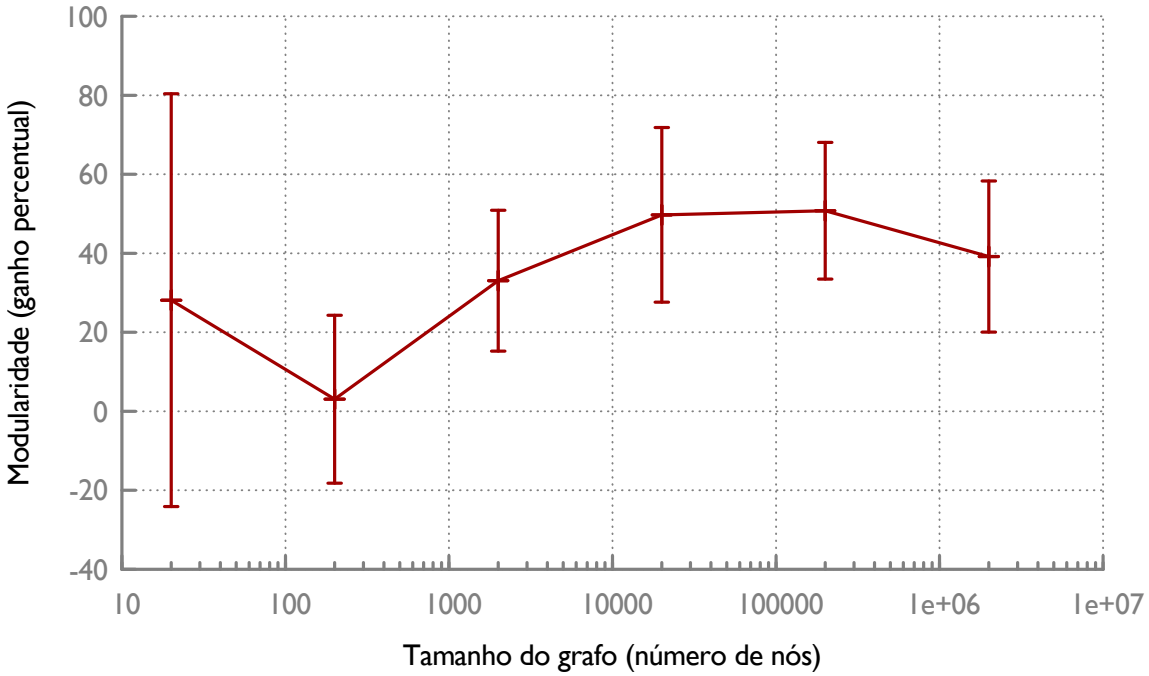


Figura 4.12: Gráfico da diferença percentual de modularidade dos algoritmos para $k = 3$

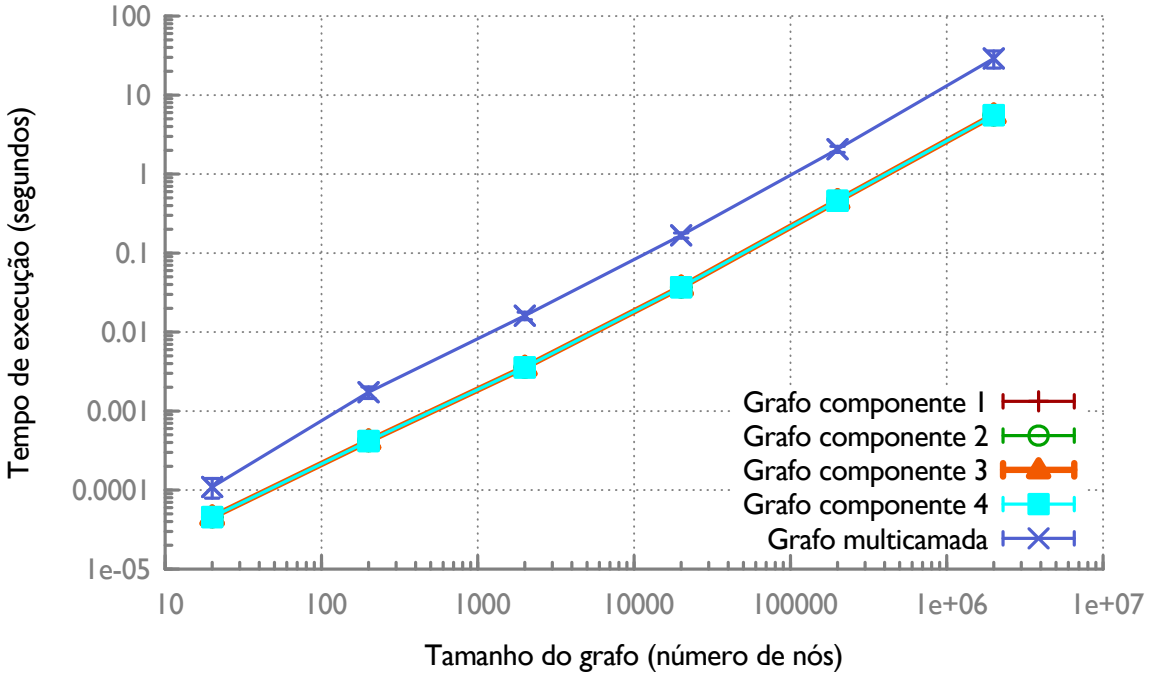


Figura 4.13: Gráfico do tempo de inicialização dos algoritmos para $k = 4$

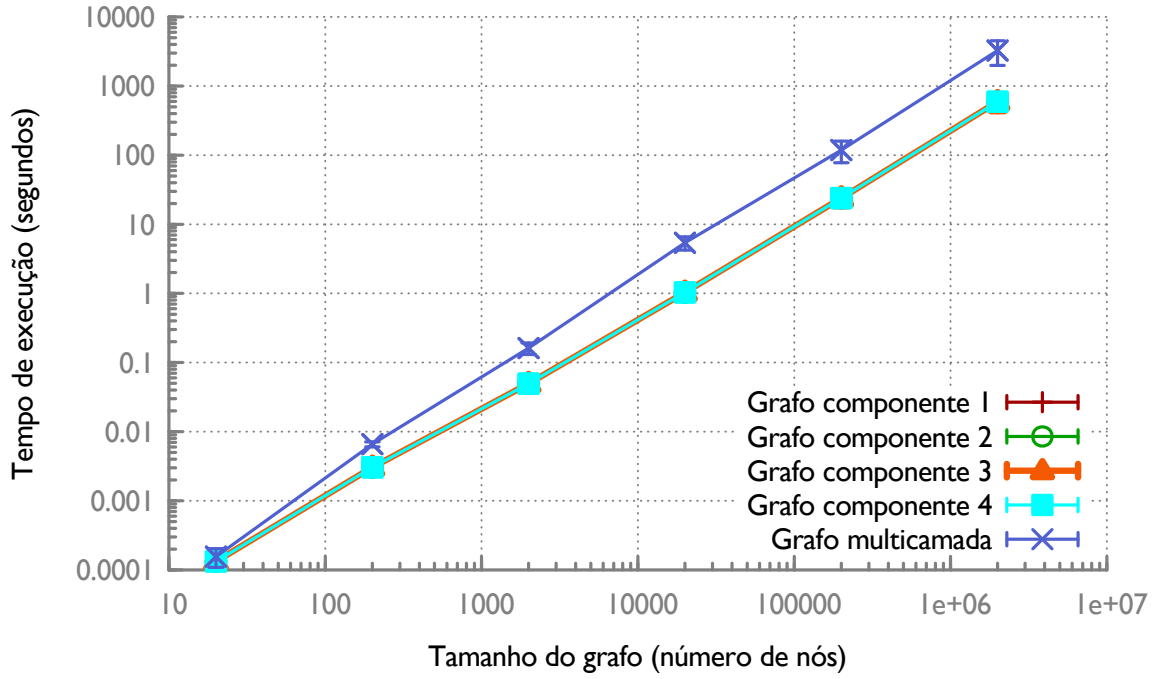


Figura 4.14: Gráfico do tempo de aglomeração dos algoritmos para $k = 4$

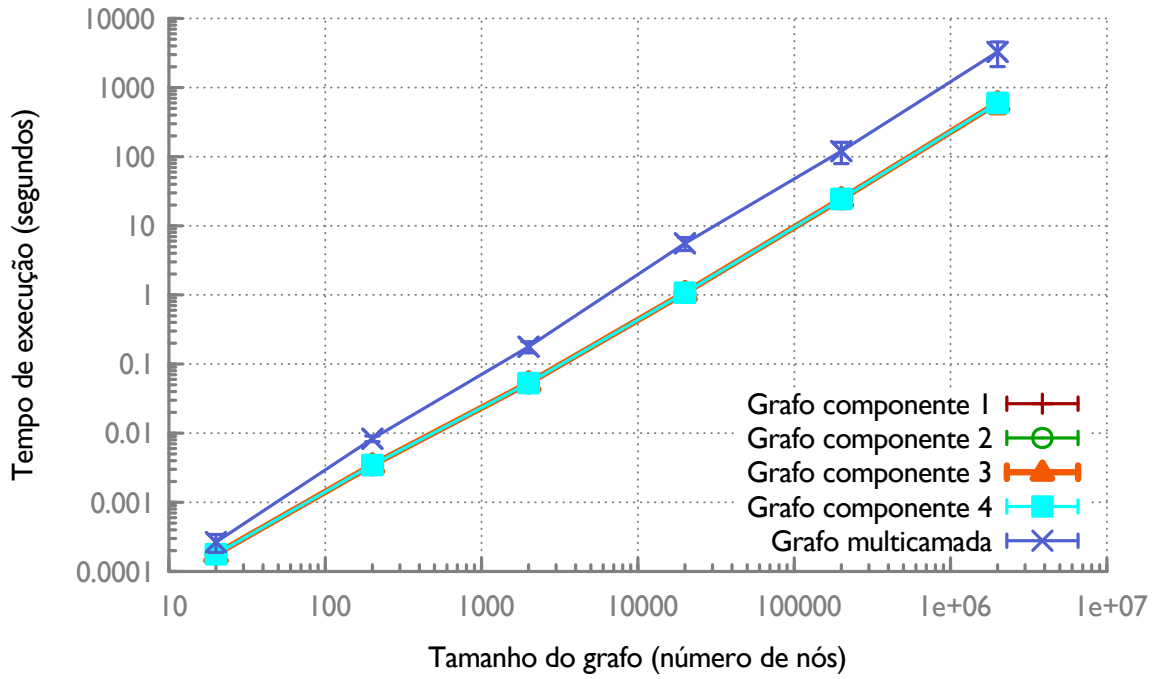


Figura 4.15: Gráfico do tempo total dos algoritmos para $k = 4$

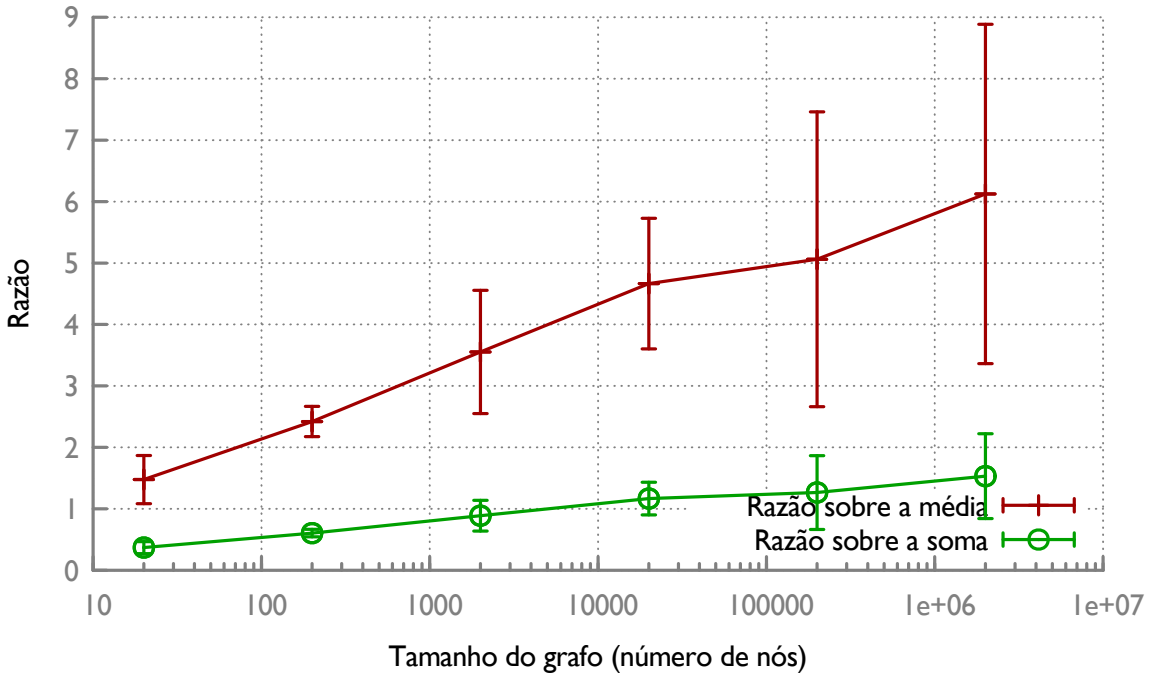


Figura 4.16: Gráfico da razão entre os tempos dos algoritmos para $k = 4$

comportamento um pouco diferente, com uma razão média bem acima de 4 (que é a multiplicidade do grafo nesse caso) para os grafos maiores, no caso da razão sobre a média dos tempos do algoritmo guloso original. No caso da razão sobre a soma dos tempo do algoritmo guloso original, temos a razão se aproximando a 1,5 para os maiores grafos. Na seção 4.2 iremos analisar em maior detalhe o que acontece com essa razão ao aumentarmos a multiplicidade do grafo.

Os gráficos das figuras 4.17 e 4.18 nos mostram os resultados do ganho de modularidade multicamada sobre o algoritmo guloso original (ganho absoluto e percentual, respectivamente). Podemos ver que o ganho absoluto é um pouco maior, o que se dá pelo fato da modularidade multicamada não ser normalizada pela multiplicidade do grafo. No caso do ganho percentual, já temos um comportamento bem parecido com o caso onde $k = 3$, onde temos um ganho médio de 30% para os grafos maiores.

4.2 Multiplicidade do grafo multicamada

Nessa seção analisaremos o comportamento do algoritmo multicamada conforme aumentamos a multiplicidade dos grafos multicamada, ou seja, conforme variamos k, o número de grafos componentes. Iremos ver como o tempo de execução varia conforme aumentamos k, e como o ganho percentual de modularidade multicamada se comporta

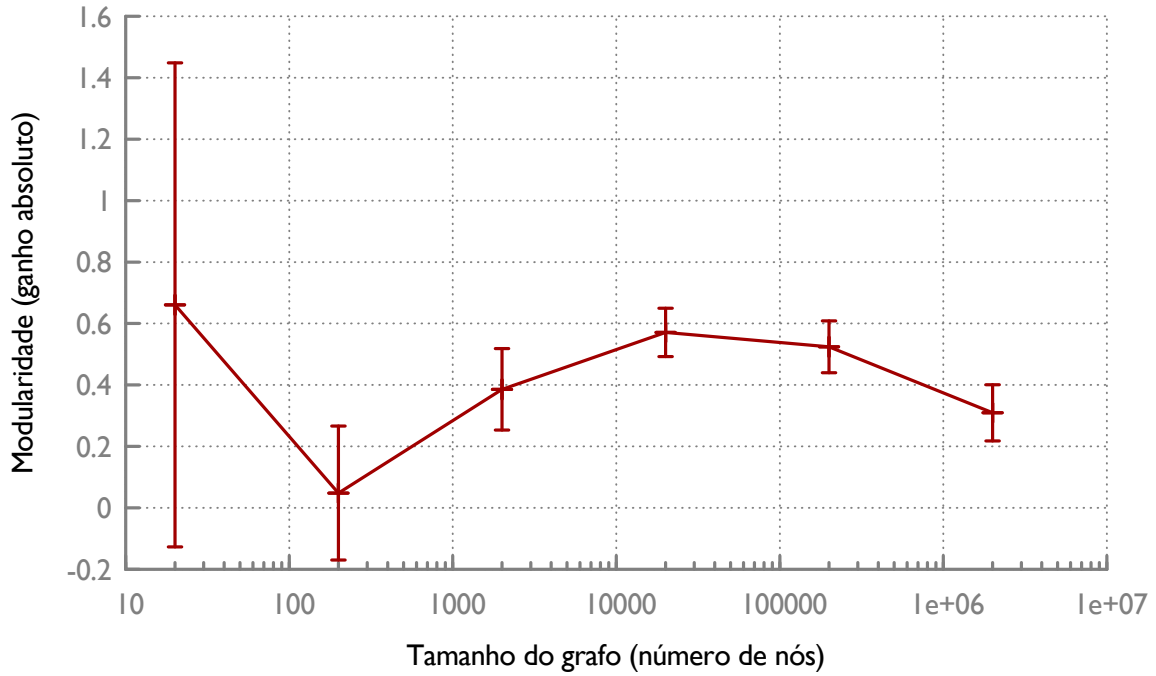


Figura 4.17: Gráfico da diferença de modularidade dos algoritmos para $k = 4$

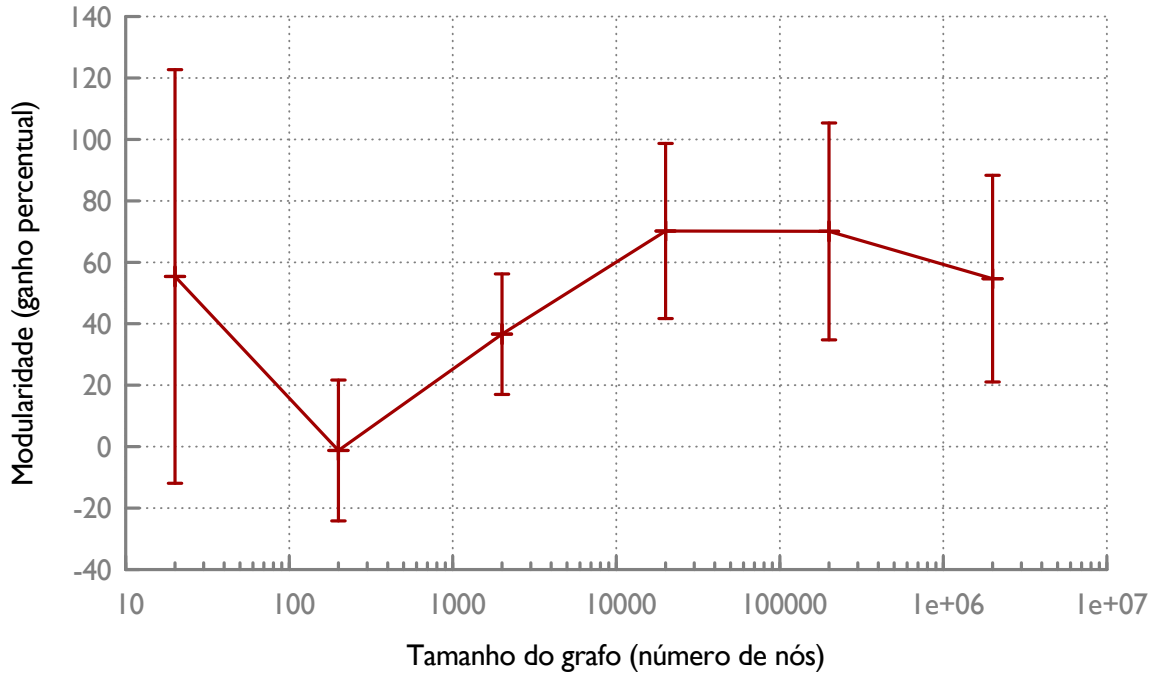


Figura 4.18: Gráfico da diferença percentual de modularidade dos algoritmos para $k = 4$

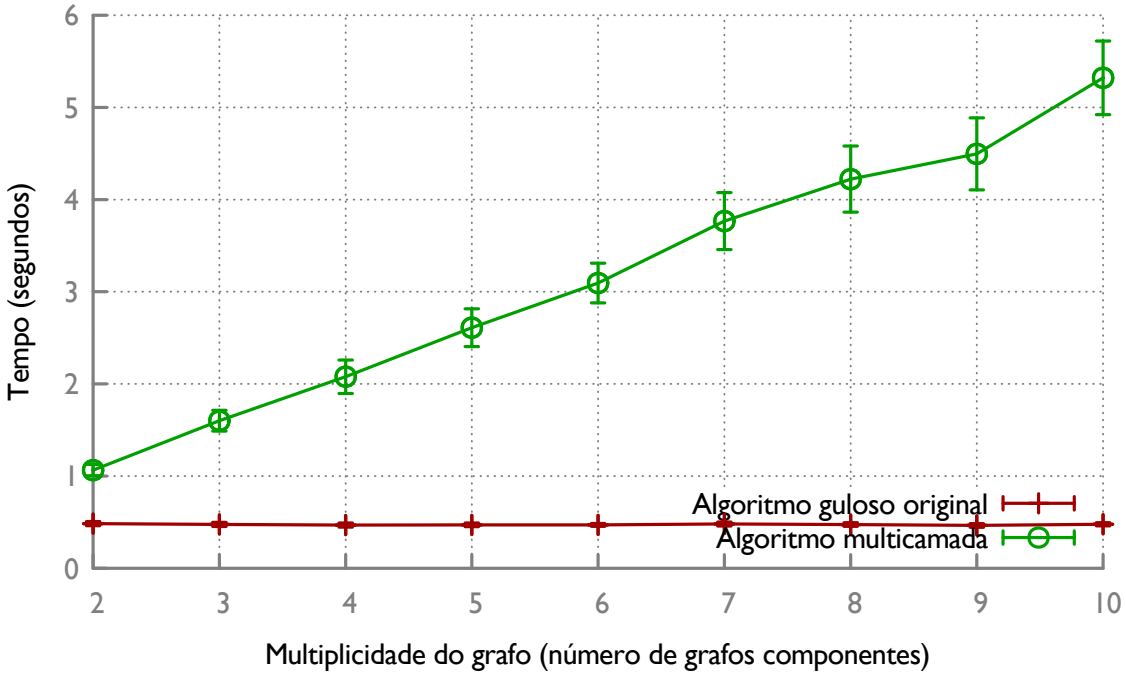


Figura 4.19: Gráfico do tempo de inicialização dos algoritmos

com essa variação. Em todos testes, o tamanho do grafo é o mesmo (200 mil nós).

As figuras 4.19, 4.20 e 4.21 contém respectivamente os gráficos dos tempos de execução da etapa de inicialização, da etapa de aglomeração e tempo total dos algoritmos. Podemos ver uma curva aproximadamente linear para o algoritmo multicamada, o que está conforme a complexidade do algoritmo, que determina um tempo de execução linear na multiplicidade do grafo multicamada em ambas etapas do algoritmo. O tempo do algoritmo guloso original é aproximadamente o mesmo sempre pois o tamanho do grafo não varia.

O gráfico da figura 4.22 mostra a razão do tempo do algoritmo multicamada sobre o tempo do algoritmo guloso original. A curva que mostra a razão sobre a soma tem uma leve inclinação, não distanciando muito do valor 1, mas crescendo um pouco conforme a multiplicidade aumenta. A curva da razão sobre a média do algoritmo guloso original nos mostra o porque disso. Conforme aumentamos a multiplicidade, a razão sobre a média cresce além do valor da multiplicidade do grafo. Apesar de ainda ser um crescimento linear, podemos ver que a inclinação é maior que 1.

Os gráficos dos ganhos de modularidade absoluto e percentual são mostrados nas figuras 4.23 e 4.24, respectivamente. Podemos ver que o valor do ganho absoluto cresce de forma aproximadamente linear com a multiplicidade do grafo, enquanto que o valor do ganho percentual tem uma variação um pouco irregular. Isso se deve ao fato de que a modularidade não é normalizada pela multiplicidade do grafo e, portanto, conforme

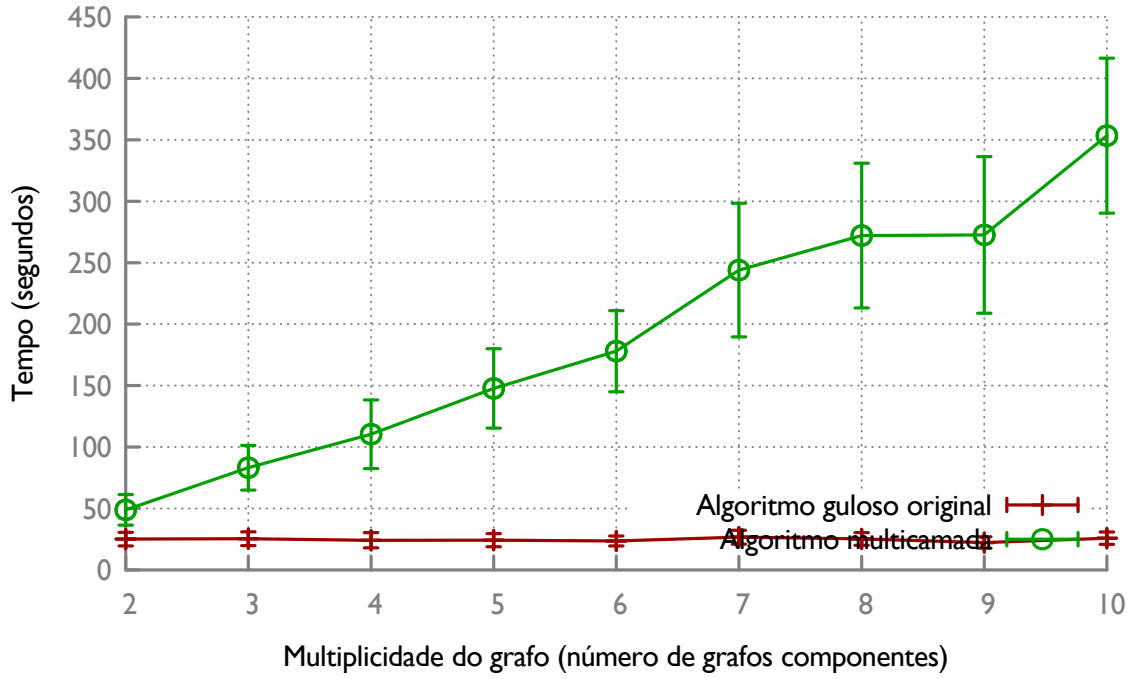


Figura 4.20: Gráfico do tempo de aglomeração dos algoritmos

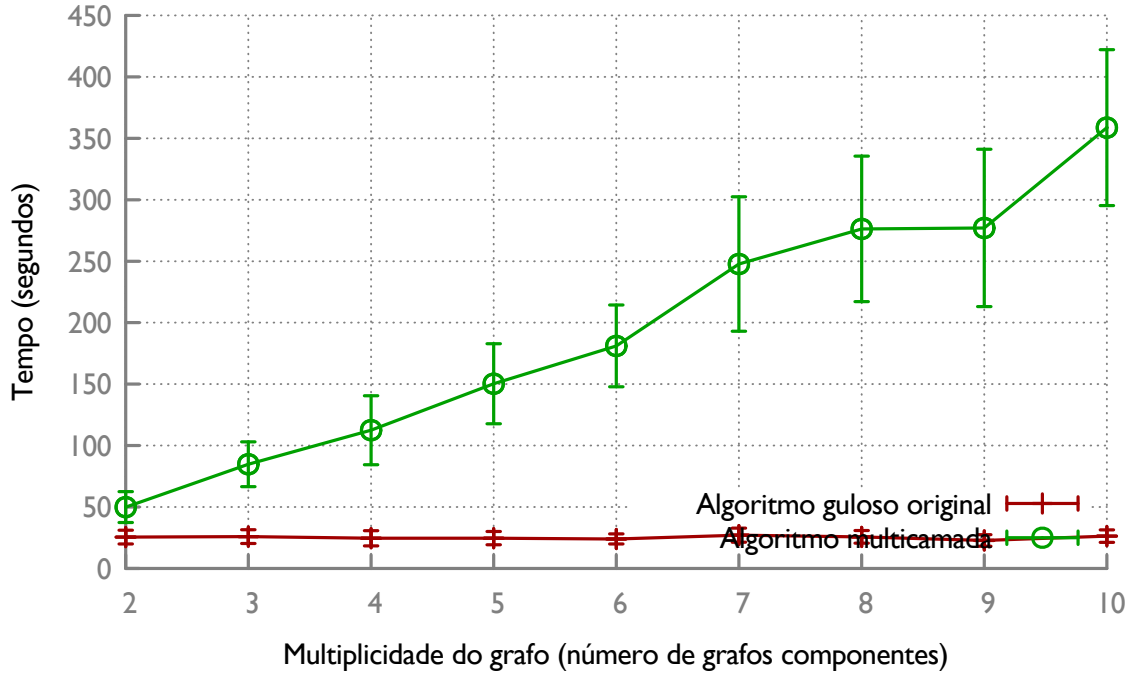


Figura 4.21: Gráfico do tempo total dos algoritmos

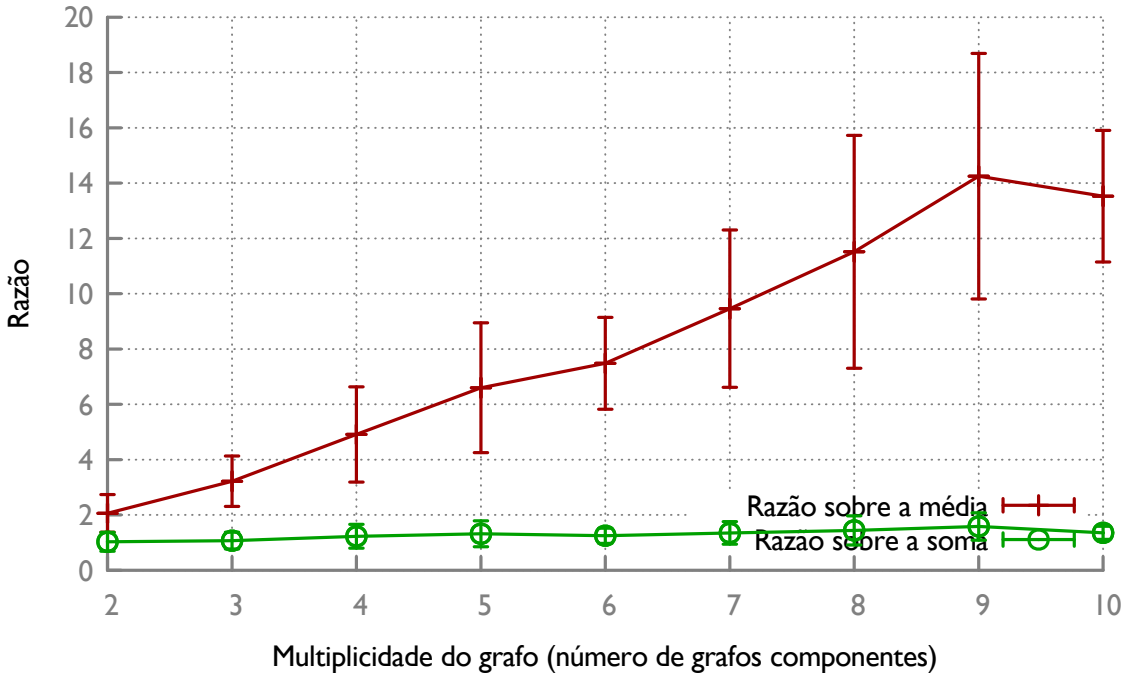


Figura 4.22: Gráfico da razão entre os tempos dos algoritmos

aumentamos o número de grafos componentes, o valor da modularidade aumenta em geral, e, como o ganho percentual é limitado, o ganho absoluto tende a crescer conforme o valor da modularidade cresce em razão do aumento de multiplicidade do grafo multicamada.

Podemos observar que, conforme o número de grafos componentes do grafo multicamada cresce, o ganho percentual varia com valores entre 70% e 100% de ganho, uma melhora significativa e esperada, dado que o algoritmo guloso original trata somente de um grafo por vez, e portanto não leve em conta a distribuição das arestas nos outros grafos componentes. Isso leva a estruturas de comunidade que são ruins no contexto do grafo multicamada como um todo. Já o novo algoritmo leva em conta todas arestas e portanto consegue estruturas de comunidade mais adequadas ao grafo como um todo.

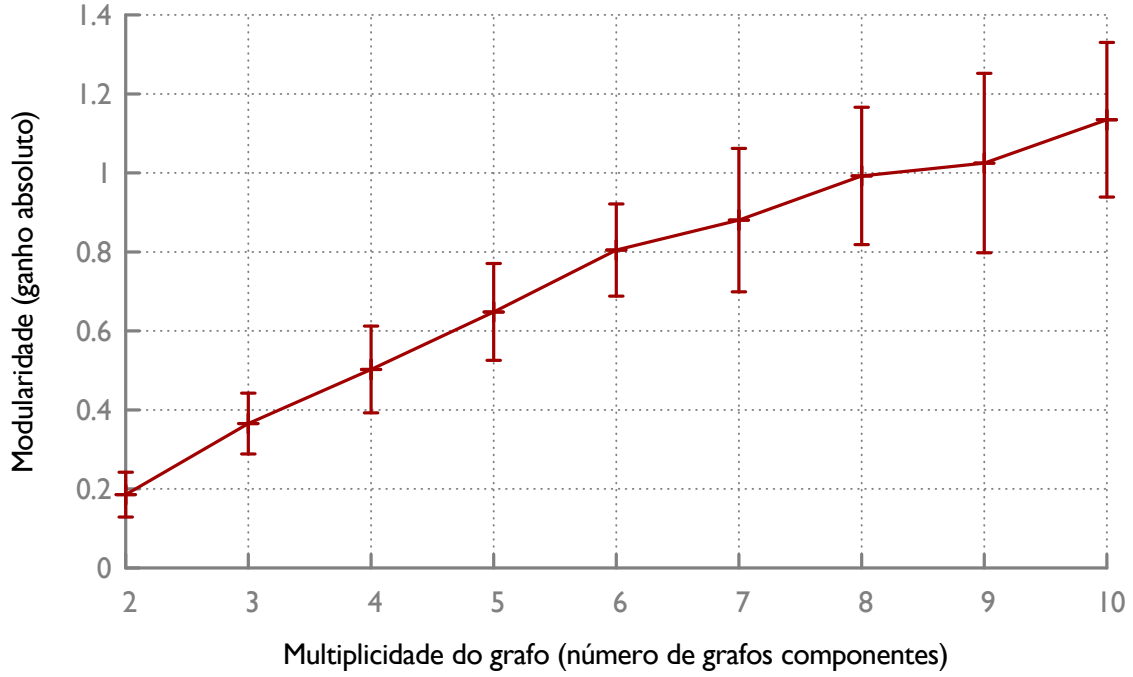


Figura 4.23: Gráfico da diferença de modularidade dos algoritmos

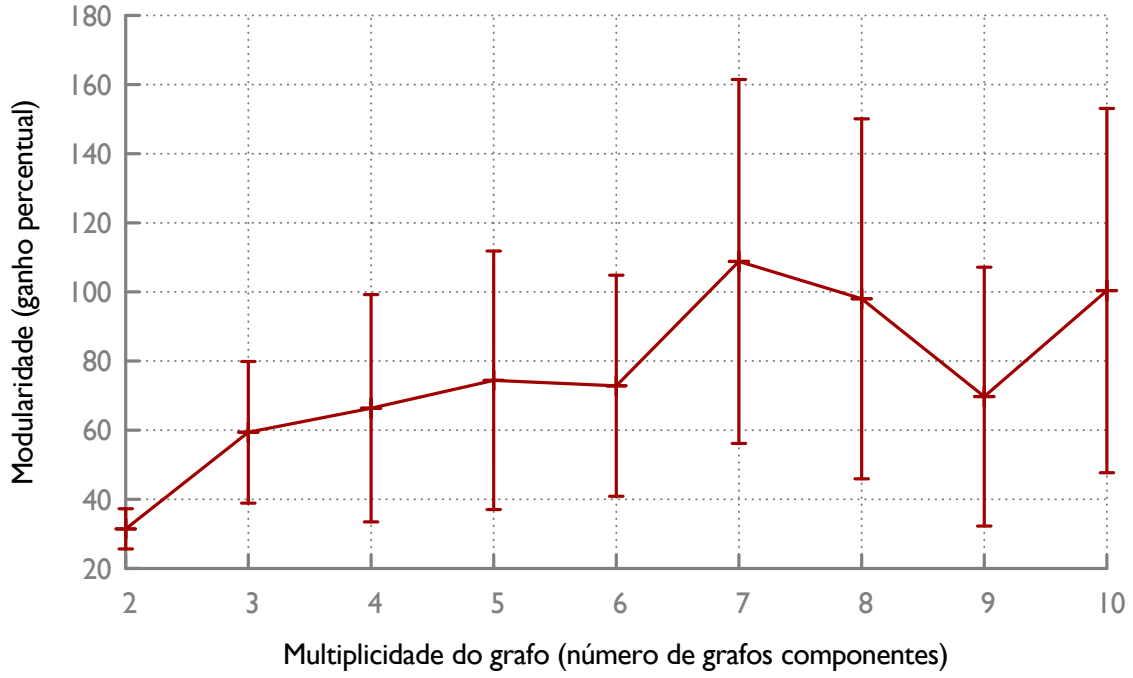


Figura 4.24: Gráfico da diferença percentual de modularidade dos algoritmos

Capítulo 5

Conclusões e trabalhos futuros

Vimos ao longo desse trabalho que, com pequenas modificações feitas no algoritmo guloso original para particionamento de grafos em comunidades, nós podemos obter melhores resultados de modularidade multicamada em grafos multicamada, com uma perda de eficiência da ordem do número de componentes do grafo multicamada. Assim, para valores limitados de k, temos que o algoritmo continua eficiente o bastante para execução em grafos muito grandes, com milhões de vértices. O tempo de execução para grafos com 2 milhões de nós e 4 grafos componentes ficou em torno de 3200 segundos, ou cerca de 53 minutos, em uma computador comum e sem paralelismo, o que representa um tempo viável para análise desses grafos, e indica a possibilidade de uso com grafos ainda maiores com tempos aceitáveis.

Acreditamos que o algoritmo possua ainda a possibilidade de usar processamento paralelo para otimizar ainda mais o tempo de execução. Visto que cada etapa de aglomeração depende da etapa anterior, temos que essas etapas tem que ser executadas de maneira serial. No entanto, os passos da aglomeração em si podem ser feitos de forma concorrente, pois a atualização de cada linha da tabela ΔQ é independente das demais. Assim, podemos executar essas instruções paralelamente, o que pode resultar em um grande ganho de eficiência. A paralelização do algoritmo é deixada portanto como um trabalho futuro promissor.

Com o ganho da modularidade multicamada sendo mais de 30% em média, temos então um algoritmo com melhores resultados de particionamento em comunidades para grafos multicamada muito grandes, o que pode ajudar nos estudos sobre estruturas de comunidades em grafos grandes vistos no mundo real. Com a proliferação de redes sociais na web, bem como outros tipos de grafos que são analisados a cada dia, esse algoritmo pode trazer grandes benefícios no estudo desses grafos que podem possuir vários tipos de conexões e portanto precisam um tratamento especial ao estudo das

estruturas de comunidades dos seus nós. Um dos trabalhos futuros é obter um grafo multicamada real no qual possamos utilizar o algoritmo multicamada para analisar as estruturas de comunidade obtidas, e compará-las com as estruturas de comunidade obtidas por outros algoritmos.

Outra possibilidade futura é o estudo da função de modularidade para grafos não só multicamada, mas grafos temporais, multi-escala, etc., para tentar desenvolver um algoritmo de detecção de comunidades também para esses tipos de grafos, o que permitiria a aplicação em vários outros tipos de grafos existentes no mundo real. Isso inclui permitir o particionamento por grafo componente como parte do algoritmo, ao invés de somente um particionamento para todo o grafo multicamada.

Acreditamos ter contribuído para o campo de comunidades em grafos com o desenvolvimento desse novo algoritmo por vários motivos. Um deles é o algoritmo ser simples e eficiente, e baseado em técnicas já conhecidas. Outro aspecto importante é o fato de estarmos iniciando estudos na área de grafos multicamada, que ainda é pouco explorada. Acreditamos que existem muitas aplicações para esses grafos, tanto no meio acadêmico quanto no meio industrial/comercial e técnico.

Referências Bibliográficas

Blondel, V. D.; Guillaume, J.-L.; Lambiotte, R. & Lefebvre, E. (2008). Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment*, 10:8–+.

Clauset, A.; Newman, M. E. J. & Moore, C. (2004). Finding community structure in very large networks. *Physical Review*, 70(6):066111–+.

Danon, L.; Díaz-Guilera, A. & Arenas, A. (2006). The effect of size heterogeneity on community identification in complex networks. *Journal of Statistical Mechanics: Theory and Experiment*, 11:10–+.

Du, H.; Feldman, M. W.; Li, S. & Jin, X. (2007). An algorithm for detecting community structure of social networks based on prior knowledge and modularity. *Complexity*, 12(3):53--60. ISSN 1099-0526.

Fortunato, S. (2010). Community detection in graphs. *Physics Reports*, 486:75–174.

Girvan, M. & Newman, M. E. J. (2002). Community structure in social and biological networks. *Proceedings of the National Academy of Sciences*, 99(12):7821--7826. ISSN 0027-8424.

Krishnamurthy, B. & Wang, J. (2000). On network-aware clustering of web clients. *SIGCOMM Comput. Commun. Rev.*, 30:97--110. ISSN 0146-4833.

Mucha, P. J.; Richardson, T.; Macon, K.; Porter, M. A. & Onnela, J.-P. (2010). Community Structure in Time-Dependent, Multiscale, and Multiplex Networks. *Science*, 328(5980):876–878.

Newman, M. E. (2004a). Fast algorithm for detecting community structure in networks. *Physical Review*, 69(6):066133–+.

Newman, M. E. J. (2004b). Analysis of weighted networks. *Physical Review*, 70(5):056131–+.

Newman, M. E. J. & Girvan, M. (2004). Finding and evaluating community structure in networks. *Phys. Rev. E*, 69(2):026113.

Perkins, C. E. (2001). Ad hoc networking. capítulo Ad hoc networking: an introduction, pp. 1--28. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

Pujol, J. M.; Béjar, J. & Delgado, J. (2006). Clustering algorithm for determining community structure in large networks. *Phys. Rev. E*, 74:016107.

Reddy, P. K.; Kitsuregawa, M.; Sreekanth, P. & Rao, S. S. (2002). A graph based approach to extract a neighborhood customer community for collaborative filtering. Em *Proceedings of the Second International Workshop on Databases in Networked Information Systems*, DNIS '02, pp. 188--200, London, UK, UK. Springer-Verlag.

Schuetz, P. & Caflisch, A. (2008a). Efficient modularity optimization by multistep greedy algorithm and vertex mover refinement. *Phys. Rev. E*, 77:046112.

Schuetz, P. & Caflisch, A. (2008b). Multistep greedy algorithm identifies community structure in real-world and computer-generated networks. *Phys. Rev. E*, 78:026112.

Shah, D. & Zaman, T. (2010). Community Detection in Networks: The Leader-Follower Algorithm. *ArXiv e-prints*.

Wakita, K. & Tsurumi, T. (2007). Finding community structure in mega-scale social networks. *CoRR*, abs/cs/0702048.

Xiang, B.; Chen, E. & Zhou, T. (2009). Finding community structure based on subgraph similarity. *CoRR*, abs/0902.2425.

Ye, Z.; Hu, S. & Yu, J. (2008). Adaptive clustering algorithm for community detection in complex networks. *Phys. Rev. E*, 78:046115.

Apêndice A

Algoritmos de geração de grafos

A.1 Geração de grafos multicamada

{Entrada:
$N \leftarrow$ número de vértices do grafo a ser gerado,
$C \leftarrow$ número de comunidades,
$K \leftarrow$ multiplicidade (número de grafos componentes),
$p_{in} \leftarrow$ probabilidade de arestas dentro de uma comunidade,
$p_{out} \leftarrow$ probabilidade de arestas entre comunidades diferentes,
$p_{change} \leftarrow$ probabilidade de comunidade de um vértice ser diferente da comunidade do mesmo no grafo componente anterior,
$\gamma \leftarrow$ variação no número de arestas por vértice}
{Variáveis:
$c \leftarrow$ matriz K por N com a comunidade de cada vértice em cada grafo (c_{ki} é a comunidade do vértice i no grafo k),
$v_{kc} \leftarrow$ vetores contendo os vértices de cada comunidade c no grafo k,
$G \leftarrow$ grafo multicamada (G_i sendo um grafo componente),
}
para $i = 1$ a N **faça**
  $a \leftarrow$ inteiro aleatório entre 1 e C (comunidade)
  para $j = 1$ a K **faça**
    $c_{ji} \leftarrow a$
    se número aleatório entre 0 e 1 $< p_{change}$ **então**
      $a \leftarrow$ inteiro aleatório entre 1 e C (comunidade)
    fim se
  fim para
fim para
para $j = 1$ a K **faça**
  para $i = 1$ a N **faça**
    $v_{jc_{ji}} \leftarrow$ adiciona i ao final
  fim para
fim para
para $j = 1$ a K **faça**
  $G_i \leftarrow$ gera grafo com algoritmo da seção A.2 e entradas $N, C, c_j, v_j c, p_{in}, p_{out}, \gamma$
fim para
retorna G

Algoritmo 3: Algoritmo de geração de grafo multicamadas

A.2 Geração de grafos simples

{Entrada:
$N \leftarrow$ número de vértices do grafo a ser gerado,
$C \leftarrow$ número de comunidades,
$c \leftarrow$ vetor indicando a comunidade de cada vértice (c_i),
$v_c \leftarrow$ vetores contendo os vértices de cada comunidade c,
$p_{in} \leftarrow$ probabilidade de arestas dentro de uma comunidade,
$p_{out} \leftarrow$ probabilidade de arestas entre comunidades diferentes,
$\gamma \leftarrow$ variação no número de arestas por vértice}
{Variáveis:
$G \leftarrow$ grafo de N vértices a ser gerado (inicialmente sem arestas),
$u \leftarrow$ número médio de arestas por vértice,
$I \leftarrow$ razão entre número de arestas dentro da comunidade e número de arestas fora da comunidade,
$e \leftarrow$ número de arestas a serem geradas para um determinado vértice,
$k \leftarrow$ vértice escolhido para formar uma aresta,
$a \leftarrow$ comunidade escolhida para formar uma aresta}
{Inicialização}
$E \leftarrow \left(p_{in} \frac{N}{C}\right) + \left[p_{out} \left(N - \frac{N}{nc}\right)\right]$
$I \leftarrow \frac{p_{in} \frac{N}{C}}{E}$
para $i = 1$ a N **faça**
  $e \leftarrow$ inteiro aleatório entre $(1-\gamma)\,E$ e $(1+\gamma)\,E$
  se $e \geq N$ **então**
    $e \leftarrow N - 1$
  fim se
  {Escolhe arestas dentro da comunidade}
  para $j = 1$ a $e \times I$ **faça**
    $k \leftarrow$ escolhe vértice aleatório entre os vértices v_{c_i} da comunidade c_i
    enquanto k = i **faça**
      $k \leftarrow$ escolhe vértice aleatório entre os vértices v_{c_i} da comunidade c_i
    fim enquanto
    adiciona aresta entre i e k no grafo G
  fim para
  {Escolhe arestas fora da comunidade}
  para $j = 1$ a $e\,(1-I)$ **faça**
    $a \leftarrow$ escolhe comunidade aleatória (entre 1 e C)
    enquanto $a = c_i$ **faça**
      $a \leftarrow$ escolhe comunidade aleatória (entre 1 e C)
    fim enquanto
    $k \leftarrow$ escolhe vértice aleatório entre os vértices v_a da comunidade a
    adiciona aresta entre i e k no grafo G
  fim para
fim para
retorna G

Algoritmo 4: Algoritmo de geração de arestas para um grafo particionado em comunidades