Método Rápido de Agrupamento de Vértices para Detecção de Comunidades em Redes Complexas de Larga-escala

Gustavo S. Carnivali[1], **Alex B. Vieira**[2], **Paulo A. A. Esquef**[1], **Artur Ziviani**[1]

[1] Laboratório Nacional de Computação Científica (LNCC)

[2]Departamento de Ciência da Computação – Universidade Federal de Juiz de Fora (UFJF)

{carnivali,pesquef,ziviani}@lncc.br,alex.borges@ufjf.edu.br

Abstract. *This paper tackles the problem of community detecion in large-scale graphs. In the literature devoted to this topic, an iterative algorithm, called Louvain Method (ML), stands out as an effective and fast solution for the problem. However, the first iterations of the ML are the most costly. To overcome this, this paper introduces a fast algorithm for vertice groupping (MRAV), which can be a way faster option than the first iterations of the ML, yet similarly effective. The performance of the proposed multistage scheme for community detection, called MRAV+ML, is confronted against that of the ML using real-world networks from small to large sizes. Experimental results show that, for large-scale graphs, the MRAV+ML detects communities with mean modularity about 3% lower than the ML, but with a mean reduction of about 42% in computation time.*

Resumo. *Este artigo reporta resultados de uma investigação sobre o problema de detecção de comunidades em redes complexas. Na literatura dedicada a esse assunto, um algoritmo iterativo, denominado Método de Louvain (ML), se destaca como opção eficaz e rápida para o problema. Entretanto, as primeiras iterações do ML são sua parte mais custosa. Neste artigo, propõe-se um Método Rápido de Agrupamento de Vértices (MRAV) em grafos como uma opção mais rápida do que as primeiras iterações do ML. Através de experimentos envolvendo redes grandes do mundo real, demonstra-se que o sistema proposto MRAV+ML identifica comunidades com modularidade similar ao ML (redução média menor de 3%), mas com redução média de aproximadamente 42% no tempo de execução.*

1. Introdução

Um grafo é uma estrutura composta por um conjunto de objetos que podem estar conectados por arestas indicando a existência de uma relação entre um par de objetos. Convenientemente, grafos podem então ser utilizados para representar várias redes complexas do mundo real [Bondy et al., 1976]. Grafos de grande porte podem ser criados na representação de grandes serviços como Facebook e Twitter ou em outros que são comuns em domínios como o WWW [Kwak et al., 2010, Mislove et al., 2007]. De forma similar, grandes grafos encontram aplicação na representação de redes complexas de larga-escala em diversas áreas do conhecimento.

Em diversas redes complexas reais, a distribuição de arestas entre os vértices apresenta-se de forma altamente heterogênea. Isso leva, nesses casos, a uma alta concentração de arestas entre agrupamentos de vértices e uma baixa concentração de arestas

entre esses agrupamentos. Essa característica de redes reais é tipicamente chamada de estrutura de comunidades. Comunidades, também chamadas de agrupamentos, são grupos de vértices que provavelmente compartilham algumas propriedades comuns ou desempenham papéis semelhantes no grafo em estudo. Nesse contexto, considerando a análise de grafos que representam redes complexas reais, um problema comum é a detecção de comunidades [Fortunato, 2010, Harenberg et al., 2014, Khan e Niazi, 2017, Zhao, 2017], que será o alvo principal de investigação deste artigo.

1.1. Detecção de comunidades em redes complexas

O problema de detecção de comunidades consiste em encontrar, em um grafo, agrupamentos de vértices que possuam uma ou mais características em comum. Uma dessas características pode ser a vizinhança em comum entre vértices, que pode ser dada pelo número de arestas que os vértices de um mesmo agrupamento (i.e., uma comunidade) compartilham entre si. Várias métricas podem ser utilizadas para avaliar os agrupamentos e as semelhanças existentes em cada comunidade [Chakraborty et al., 2017]. Nesse contexto, uma métrica muito utilizada para avaliar a qualidade das comunidades é a modularidade Q [Girvan e Newman, 2002, Newman, 2006]. Intuitivamente, $-1 < Q < 1$ é uma medida do quão densamente conectados entre si estão os vértices do mesmo tipo ou classe. O valor de Q se torna mais positivo (resp. negativo) quanto maior (resp. menor) for o número de arestas conectando vértices de uma mesma classe, se comparado com uma distribuição aleatória de arestas entre os mesmos vértices.

O problema de detecção de comunidades, pode ser tratado como um problema de maximização da modularidade das partições encontradas em um grafo. Formalmente, a modularidade é dada por [Girvan e Newman, 2002, Newman, 2006]:

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

onde m é o número de arestas do grafo, $A_{ij} = 1$ se os vértices i e j forem conectados e $A_{ij} = 0$ caso contrário (i.e., A é a matriz de adjacência do grafo), k_i é o grau do vértice i (ou seja, o número de arestas incidentes sobre i), c_i é a classe ou tipo do vértice i e a função $\delta(c_i, c_j)$ é o δ de Kronecker, ou seja, $\delta(c_i, c_j) = 1$ se os vértices i e j forem da mesma comunidade ou tipo, isso é, $c_i = c_j$; e $\delta(c_i, c_j) = 0$ se os vértices i e j não forem da mesma comunidade ou tipo.

Um levantamento comparativo de diversos métodos de detecção de comunidades pode ser encontrado em [Lancichinetti e Fortunato, 2009, Leskovec et al., 2010]. Dentre eles, o chamado Método de Louvain (ML) [Blondel et al., 2008] destaca-se por identificar comunidades em grafos muito grandes, de modo rápido e eficaz (i.e., com alta modularidade). Um breve descritivo do ML é apresentado na próxima subseção.

1.2. Método de Louvain [Blondel et al., 2008]

O ML para detecção de comunidades em grafos emprega uma heurística iterativa baseada na maximização da modularidade. Inicialmente, o ML atribui cada vértice do grafo a uma comunidade distinta. No primeiro estágio, para todo vértice i do grafo, o ML considera a remoção de i de sua comunidade para a comunidade de cada vizinho j de i. De

forma gulosa, o ML escolhe mover i para a comunidade do vértice vizinho j que leve ao maior ganho em modularidade. Cabe notar que a variação de modularidade produzida pela mudança considerada de i para um vizinho j pode ser computada de forma local e eficiente [Blondel et al., 2008]. Esse processo é repetido para cada vértice até que não haja possível incremento na modularidade, completando a primeira etapa do ML.

O segundo estágio do ML gera um novo grafo (reduzido) onde os novos vértices são as comunidades encontradas no passo anterior. A saída do segundo estágio é submetida à entrada do primeiro estágio e assim por diante, enquanto houver ganho de modularidade nas comunidades encontradas.

Segundo [Blondel et al., 2008], o ML possui complexidade computacional $\mathcal{O}(n \log n)$, onde n é o número total de vértices do grafo. O ML começa com um grafo de n vértices e, ao fim de cada rodada de seus dois estágios, a tendência é obter grafos com um número menor de vértices (comunidades). Assim, as primeiras rodadas do ML concentram a maior parte do custo computacional.

Apesar da baixa complexidade do ML, devido aos grandes grafos que podem ser encontrado atualmente, é possível encontrar na literatura propostas que visam otimizar o ML em relação ao seu tempo de execução. Em [Ozaki et al., 2016], os autores analisaram os processos do ML que implicavam algum desperdício temporal e propuseram soluções simples para otimizá-los. Há trabalhos que aceleram o ML através de implementações que usam computação paralela [Bhowmick e Srinivasan, 2013, Fazlali et al., 2017]. Em [Fortunato e Barthelemy, 2007], os autores investigam os limites de resolução do ML na detecção de comunidades, uma vez que a maximização da modularidade não favorece à detecção de pequenas comunidades. Já em [Aldecoa e Marín, 2013], os autores avaliam o desempenho de 18 métodos de detecção de comunidades e evidenciam a subotimalidade dos resultados quando a modularidade é maximizada. Eles propõem então um novo parâmetro global para avaliar a qualidade de partições do grafo, chamada de surpresa [Aldecoa e Marín, 2011], cuja maximização obtida pela combinação de múltiplos algoritmos alcança melhores resultados.

Outra classe de métodos encontrada na literatura se utiliza da estratégia de aplicar algoritmos rápidos ao grafo de entrada, entregando ao ML um grafo mais simples e, por conseguinte, reduzindo o tempo de execução da tarefa de detecção de comunidades. Em [Satuluri et al., 2011], o grafo original é reduzido a partir da eliminação de arestas que possivelmente fariam parte de uma mesma comunidade, antes de ser submetido ao ML. O procedimento reduz o tempo de execução da detecção de comunidades e pouco altera a qualidade das comunidades encontradas. Entretanto, o método se torna ineficiente em grafos com baixo grau médio de conexões (número pequeno de arestas em relação ao número de vértices). Outro método, apresentado em [Peng et al., 2014], gera um corte no grafo a partir de seu k-core, produzindo subgrafos do grafo original com todos os vértices que tenham pelo menos k conexões. Esses subgrafos são aplicados a um algoritmo de detecção de comunidades (como o ML). Após definidas as comunidades dos subgrafos, eles são reconectados apropriadamente.

1.3. Método Proposto

O método proposto neste artigo tem objetivos similares aos da última classe de técnicas analisadas na Seção 1.2: modificar o grafo original e entregar ao ML um grafo de tamanho

reduzido, visando a reduzir o tempo de execução do processo de detecção de comunidades, sem perda significativa na modularidade das comunidades identificadas. Propõe-se um Método Rápido de Agrupamentos de Vértices (MRAV), que faz um papel similar à primeira rodada (mais custosa) do ML, i.e., de gerar um grafo com comunidades bem estabelecidas próximas da estrutura do grafo original. O ML é acelerado pois sua primeira fase é substituída por uma que priorize a velocidade e não a qualidade, deixando a valorização da qualidade para as etapas seguintes do ML. A aplicabilidade do MRAV, tal como proposto neste artigo, é investigada em cenários restritos a grafos não-ponderados, i.e., que não possuem arestas ou vértices pontuados; grafos não-direcionados e com geração de comunidades disjuntas, i.e., comunidades sem sobreposição de vértices.

O MRAV consiste em um algoritmo iterativo onde um vértice é inicialmente atribuído a uma comunidade. De modo iterativo, cada vizinho a esse vértice é pontuado de acordo com uma medida de seu grau de conectividade com os vértices pertencentes à comunidade. É qualificado para integrar a comunidade do vértice de referência o vizinho com maior pontuação acima de um limite mínimo, controlado por uma variável chamada de K, com valor definido experimentalmente (detalhes na Seção 3). Em caso de empate, i.e, mais de um vizinho com a maior pontuação, sorteia-se aleatoriamente um deles para ingressar na comunidade. O vizinho recém-integrado à comunidade passa a ser o vértice de referência e as iterações continuam até não haver mais vizinhos que se qualifiquem para ingressar na comunidade. Nesse ponto, a comunidade é fechada, uma nova comunidade é criada e um vértice ainda sem comunidade é sorteado para ingressá-la. As iterações continuam dessa forma até que todos os vértices do grafo sejam associados a comunidades, que são transformadas em vértices de um grafo reduzido.

Na investigação reportada, inicialmente é avaliado o desempenho do MRAV em função de K e, após isso, o desenpenho do MRAV+ML em relação ao ML. Para a avaliação são utilizados grafos associados a redes reais com número de vértices entre aproximadamente 1 mil e 5 milhões. A investigação de K se torna necessária pois seu valor pode alterar indicadores como o tempo de execução e número de grupos (comunidades). Os testes mostraram que K pode ser ajustado no MRAV+ML para reduzir substancialmente o tempo de execução do método de detecção de comuinidades, ao custo de uma leve redução na modularidade das comunidades encontradas. Os resultados experimentais mostram que, para grandes redes, a proposta MRAV+ML produz uma redução de 45% no tempo médio de execução e perdas médias menores de 3% em qualidade, no sentido da modularidade.

1.4. Organização do Artigo

Além desta introdução, o artigo é organizado da seguinte forma. Uma descrição detalhada do MRAV é apresentado na Seção 2, que também inclui a metodologia usada nos experimentos. Os resultados experimentais da avaliação de desempenho do MRAV e da composição MRAV+ML são reportados na Seção 3. Por fim, as conclusões da investigação feita e possíveis desdobramentos futuros são resumidos na Seção 4.

2. Método Rápido de Agrupamento de Vértices (MRAV)

Nesta seção será apresentado em detalhes o método proposto neste artigo. Os objetivos do MRAV e um breve resumo de sua heurística foram descritos na Seção 1.3.

Algoritmo 1: Método Rápido de Agrupamento de Vértices (MRAV)

```
1  início
2      Todos os vértices são rotulados como Desagrupado
3      Enquanto (Existir vértice definido como Desagrupado){
4          Novo grupo G iniciado
5          Pontuacao de todo vértice Desagrupado é zerada
6          Limite = 0
7          Enquanto (){
8              Vertices[] = conjunto de vértices Desagrupado com maior
           Pontuacao
9              Vertice = vertice aleatório de Vertices
10             Se (Pontuacao do Vertice ≥ Limite) {
11                 Adiciona Vertice à G
12                 Vertice é rotulado como Agrupado
13                 Limite é aumentado (Equação 4)
14                 Para Todo (I vizinho de Vertice) {
15                     Pontuacao de I é aumentada (Equação 3)
16                 }
17             }
18             Se Nao {
19                 Encerra grupo G
20             }
21         }
22     }
23 fim
```

O pseudocódigo do MRAV pode ser visto no Algoritmo 1. Como entrada, é recebido um grafo e o algoritmo retorna um conjunto de comunidades (grupos) populadas por vértices. Todo vértice começa rotulado como $Desagrupado$ (Linha 2) e com $Pontuacao$ 0 (Linha 5). Um novo grupo vazio G é criado (Linha 4). A variável $Limite$ é iniciada com 0 (Linha 6). As variáveis $Pontuacao$ e $Limite$ estão relacionadas ao critério (Linha 10), que qualifica um vértice a ingressar no grupo. A pontuação dos vértices candidatos a ingressar no grupo é feita pelo *loop* interno (Linha 7).

A variável $Vertices[]$ (Linha 8) vai conter, tipicamente, o vértice (e pontuação associada) desagrupado que obteve a maior das pontuações, dentre aqueles que são vizinhos ao vértice recém alocado ao grupo. Na eventualidade de empate na pontuação máxima, i.e., $Vertices[]$ terá mais de um vértice candidato, um dos quais será sorteado (Linha 9) e atribuído à variável $Vertice$. A exceção se dá quando da criação de um grupo, quando a pontuação de todos os vértices desagrupados é zerada (Linha 5). Nesse caso, como $Vertices[]$ contém todos os vértices desagrupados, a primeira passagem pelas Linhas 8 e 9 implica fazer uma seleção aleatória dentre os vértices desagrupados.

O vértice selecionado na Linha 9 será incluído no grupo G (Linha 11) e rotulado como tal (Linha 12), caso passe o teste da linha 10. Ou seja, o vértice selecionado é incluído no grupo G e rotulado como agrupado se sua $Pontuacao$ for maior ou igual

ao valor de $Limite$. Caso contrário, o grupo é encerrado e, se ainda houver vértices desagrupados, um novo grupo é criado.

Neste ponto, é oportuno esclarecer o papel conceitual da comparação entre as variáveis $Pontuacao$ e $Limite$, assim justificando as linhas 13 e 15 do pseudocódigo. Uma medida razoável do nível de conectividade de um vértice desagrupado B com um vértice agrupado A pode ser dada por:

$$\mathcal{C}(B, A) = \frac{Viz(A, B) + 1}{Grau(B)}, \tag{2}$$

onde $Viz(A, B)$ é o número de vizinhos comuns entre A e B, sendo $Grau(B)$ o número total de vizinhos de B. A soma de uma unidade no numerador da equação é feita para refletir a própria conexão entre A e B, nesse caso, $Viz(A, B) + 1$ pode assumir valores entre 0 e $Grau(B)$ ($0 \leq Viz(A, B)+1 \leq Grau(B)$) tornando $\mathcal{C}(B, A)$ uma porcentagem dos vizinhos de B. Considere o conjunto de valores de $\mathcal{C}(B, A_j)$, com $j = 1, 2, \ldots, N$, computados para cada vértice A_j já agrupado até a iteração N. Se a ***média*** dos valores de $\mathcal{C}(B, A_j)$, i.e., $\bar{\mathcal{C}} = N^{-1} \sum_{j=1}^{N} \mathcal{C}(B, A_j)$, for superior ou igual a um limite mínimo K, arbitrariamente prescrito para o nível de conectividade, então B se qualifica a ingressar no grupo.

Graças à natureza iterativa do algoritmo, em que o grupo vai sendo gradativamente populado com um vértice por vez, optou-se por uma implementação prática do critério que não envolve o cômputo de médias. Na iteração N, pode-se de modo equivalente comparar $N\bar{\mathcal{C}} = \sum_{j=1}^{N} \mathcal{C}(B, A_j)$ com NK. A soma dos valores de $\mathcal{C}(B, A_j)$ é calculada de forma recursiva (Eq. 3) e armazenada na variável $Pontuacao$ (Linha 15). De modo análogo, NK é obtido de forma recursiva (Eq. 4) e armazenado na variável $Limite$ (Linha 13).

Pelo arrazoado acima, a pontuação de um vértice desagrupado B, candidato a ingressar no grupo é dada pela seguinte dinâmica recursiva:

$$Pontuacao(B) \leftarrow Pontuacao(B) + \frac{Viz(A, B) + 1}{Grau(B)}, \tag{3}$$

com condição inicial $Pontuacao(B) = 0$, forçada após a criação de um grupo (Linha 5). De modo análogo, a dinâmica recursiva da variável $Limite$ é dada por:

$$Limite \leftarrow Limite + K,\ 0 \leq K \leq 1, \tag{4}$$

também com condição inicial $Limite = 0$ (Linha 6).

O vértice recém-integrado ao grupo (Linha 12) passa a ser o vértice de referência para efeito do levantamento dos novos vértices candidatos a ingressar o grupo, i.e., aqueles desagrupados e vizinhos ao vértice de referência. A pontuação desses vértices candidatos é atualizada segundo a Equação 3 (Linha 15). Vale notar que a atualização da pontuação dos vértices candidatos implica na modificação da variável $Vertices[]$, da qual um candidato com a maior pontuação será selecionado (Linha 9) e avaliado quanto à qualificação para ingressar no grupo (Linha 10). Como mencionado anteriormente, se o melhor candidato não atender ao critério de ingresso no grupo, esse é fechado (Linha 19). Outro grupo é criado se ainda houver vértices desagrupados (Linha 3).

A complexidade temporal do algoritmo é dada pela presença de dois $loops$: (i) um $loop$ principal que percorre todos os vértices do grafo; e (ii) um $loop$ secundário que percorre todos os vizinhos de cada vértice. Como em cada vértice todos os seus vizinhos são percorridos, a complexidade é dependente do número de vizinhos que os vértices possuem. Logo, o algoritmo MRAV possui complexidade computacional de $\mathcal{O}(e)$ onde e é o número total de arestas do grafo.

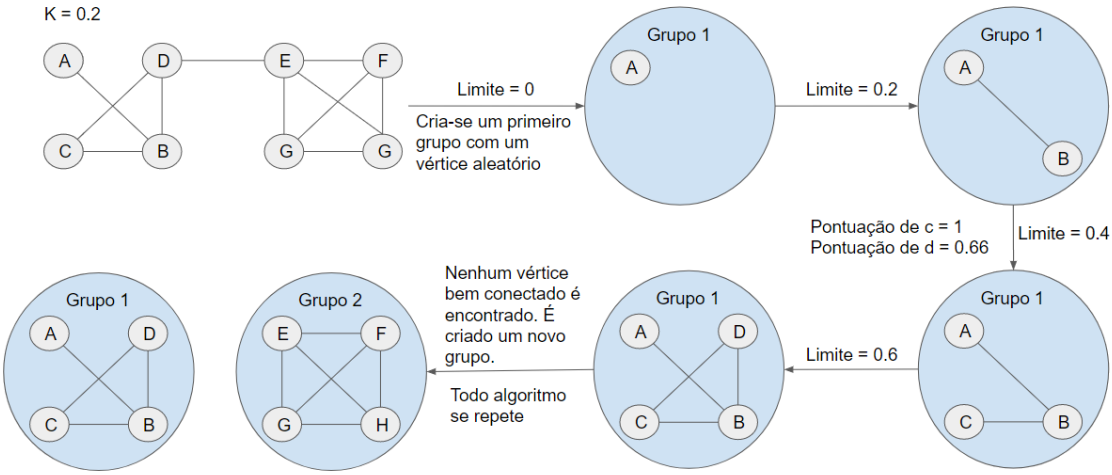

Figura 1. Exemplo ilustrativo das fases do algoritmo MRAV com $K = 0.2$.

Um exemplo visual das iterações do MRAV (com $K = 0.2$[1]) sobre um grafo simples pode ser visto na Figura 1. Inicialmente, um novo grupo é iniciado (Grupo 1) e um vértice é aleatoriamente alocado a ele (vértice A). Como B é o único vizinho de A, ele é o único candidato a ser agrupado a A. Ademais, como $Limite = 0$, B é adicionado ao mesmo grupo de A. Após adicionar B, é possível adicionar C ou D ao grupo. Porém, usando a Equação 3, $Pontuacao(C) = 1$ e a $Pontuacao(D) = 0.66$. Escolhendo a maior $Pontuacao$, o vértice C é adicionado ao grupo, já que a $Pontuacao(C) = 1$ é maior que $Limite = 0.2$. Após C ser adicionado, o único vizinho disponível de C é D. Como a $Pontuacao(D) = 1.32$ é maior que $Limite = 0.4$, D é também adicionado ao grupo. O único vizinho disponível de D é E. No entanto, $Pontuacao(E) = 0.25$ é menor que $Limite = 0.6$ e, portanto, o Grupo 1 é encerrado e um novo grupo é criado (Grupo 2). Pela conexão completa dos vértices E, F, G e H, o algoritmo os adicionaria ao mesmo grupo (Grupo 2) de forma análoga à formação do Grupo 1. Dois grupos são, assim, gerados ao fim da execução do algoritmo MRAV nesse exemplo ilustrativo.

3. Experimentos e Resultados

Nesta seção, são descritas a metodologia de testes e as redes complexas reais empregadas na avaliação de desempenho. Em seguida, são reportados os resultados experimentais.

3.1. Metodologia de Testes

Testes experimentais foram projetados e realizados para analisar o desempenho do método proposto (MRAV+ML) em relação ao ML na sua forma original. Pela construção do MRAV, apenas o valor de K (ver Equação 4) pode alterar a eficiência do método em relação ao tempo de execução e à modularidade das comunidades encontradas.

[1]Por conveniência, usamos "." como separador decimal neste artigo.

Tabela 1. Redes reais utilizadas para os testes [Bai et al., 2017, Peng et al., 2014].

Nome	# Vértices	# Arestas
Email [Guimera et al., 2003]	1,133	5,451
Polblogs [Adamic e Glance, 2005]	1,490	16,718
Reactome [Joshi-Tope et al., 2005]	6,327	147,547
PGP [Boguná et al., 2004]	10,680	24,316
DBLP [Yang e Leskovec, 2015]	317,080	1,049,866
Amazon [Yang e Leskovec, 2015]	334,863	925,872
Youtube [Leskovec e Krevl, 2014]	1,134,890	2,987,624
LiveJournal [Mislove et al., 2007]	5,204,176	49,174,620

O valor de K define um limite mínimo para o percentual das conexões de um vértice fora de uma comunidade com vértices dentro dessa comunidade. Se o referido percentual excede K, o vértice se qualifica a ingressar na comunidade. Se configurado com valor muito baixo para K, o MRAV relaxa a exigência de nível de conectividade para ingresso em uma comunidade. Logo, vértices são mais facilmente agrupados e o MRAV gera poucos grupos mais populosos. Isso tende a reduzir o tempo de execução do conjunto (MRAV+ML), mas, possivelmente, também tende a reduzir a modularidade das comunidades encontradas. Ao contrário, K com valor alto faz com que o MRAV tenda a produzir muitas comunidades com poucos membros, o que não cumpre o propósito do MRAV de entregar ao ML um grafo reduzido. Como consequência, MRAV+ML tende a ter desempenho próximo ao do ML, tanto em termos de tempo de execução e modularidade. O melhor valor de K no MRAV+ML é aquele que minimiza o tempo de execução e a perda de modularidade das comunidades resultantes, em relação ao ML. Para encontrar tal valor, serão feitos testes sobre 8 grafos comumente utilizados em trabalhos de detecção de comunidades [Bai et al., 2017, Peng et al., 2014], cujas dimensões em número de vértices e arestas são apresentadas na Tabela 1.

Para cada grafo presente na Tabela 1, o desempenho do MRAV será avaliado para $K \in \{0.01, 0.02, 0.03, 0.1, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9\}$, em termos de indicadores como tempo de execução e número de grupos (comunidades) encontrado. Para aferir o efeito do sorteio aleatório do vértice inicial sobre o desempenho do MRAV após a criação de um novo grupo, o MRAV será executado 20 vezes para cada valor de K. Assim, valores médios e respectivos intervalos de 99% de confiança são reportados. Testes análogos aos descritos acima são feitos para avaliar o desempenho do MRAV+ML em função de K no que tange a indicadores como tempo de execução e modularidade das comunidades encontradas. Com a bateria de testes proposta, espera-se ser possível definir um K para o MRAV+ML que minimize tanto o tempo de execução quanto a perda de modularidade das comunidades encontradas, em relação ao ML.

Com o valor de K estudado e estabelecido, o MRAV+ML será testado em relação ao ML. Para isso, o ML será também aplicado aos 8 grafos, também 20 vezes, e as médias dos resultados encontrados, para o tempo de execução e a modularidade, serão comparadas com aquelas obtidas pelo MRAV+ML, com o valor de K fixado.

Todos os experimentos foram realizados em um servidor com processador Intel Xeon Dual Quad Core (2.27 GHz) com 48 GB de RAM e 1 TB para armazenamento.

3.2. Desempenho do MRAV em função de K

Os resultados dos testes especificados na Seção 3.1 serão apresentados e analisados nesta seção. Para isso, o desempenho do MRAV em função de K é avaliado para indicadores como o tempo de execução e o número de grupos encontrados.

Em todos os gráficos apresentados (Figuras 2 a 4), o eixo das abscissas representa o valor de K. Para cada rede, a curva sólida expressa a média dos 20 testes realizados, enquanto a região colorida que a envolve representa o intervalo de confiança para cada medida. Em alguns casos, os intervalos de confiança não são visíveis na escala mostrada.

Os resultados do tempo de execução do MRAV, em função de K, indicam que o tempo de execução é razoavelmente estável em função de K para a grande maioria das redes listadas na Tabela 1.[2] Nesse caso, os intervalos de confiança são pequenos, sugerindo que a escolha aleatória do vértice inicial que popula um grupo tem pouco efeito sobre o tempo de execução do MRAV.

A Figura 2 mostra o número de grupos gerados pelo MRAV em função de K, para as redes listadas na Tabela 1. É possível perceber um aumento no número de grupos com o aumento de K. Esse efeito é o esperado, já que um valor alto de K representa uma exigência de um percentual alto de vizinhos compartilhados entre os vértices dentro do grupo e um vértice fora do grupo, para que esse vértice se qualifique a integrá-lo.

Cada grupo encontrado pelo MRAV se torna um vértice de um grafo reduzido a ser entregue ao ML. Como a complexidade do ML é $\mathcal{O}(n \log_2 n)$, com n sendo o número de vértices do grafo de entrada, é desejável empregar o MRAV com K pequeno. Cabe ainda notar que, nas escalas empregadas nos gráficos da Figura 2, não são visíveis os intervalos de confiança associados ao número médio de grupos. Isso sugere que a escolha aleatória do vértice inicial que popula um grupo tem efeito desprezível sobre o número médio de grupos produzidos pelo MRAV, em função de K.

Para verificar experimentalmente a análise anterior, a Figura 3 apresenta o tempo total gasto pelo MRAV+ML em função de K. É possível confirmar a tendência de crescimento do tempo médio de execução do MRAV+ML, com o aumento de K, exceto para valores muito pequenos de K. Nota-se ainda uma tendência do intervalo de confiança decrescer com a redução de K.

Por um lado, deseja-se reduzir o tempo médio de execução do MRAV+ML em relação ao ML. Por outro lado, deseja-se que sejam similares as modularidades das comunidades encontradas pelo MRAV+ML e pelo ML. Para verificar isso, foi calculada a modularidade das comunidades geradas pelo MRAV+ML, em função de K. O resultado é expresso na Figura 4. Percebe-se que a modularidade média, para todos os grafos testados, é aproximadamente constante para $K \geq 0.2$ e tende a decrescer para $K < 0.2$.

Dos resultados mostrados nas Figuras 3 e 4, fica claro que, na faixa $0 \leq K \leq 0.1$ (em destaque na Figura 4), reduzir K tende a diminuir o tempo médio de execução do MRAV+ML, ao custo de uma queda leve na modularidade média das comunidades detectadas. Em princípio, é possível especular que esteja contido na faixa supracitada o valor de K que otimize em média o tempo de execução e a perda da modularidade. Para

[2] A figura associada a esses resultados foi omitida devido à limitação de espaço, uma vez que, por haver muito baixa variância, há pouco valor na representação visual desses resultados.

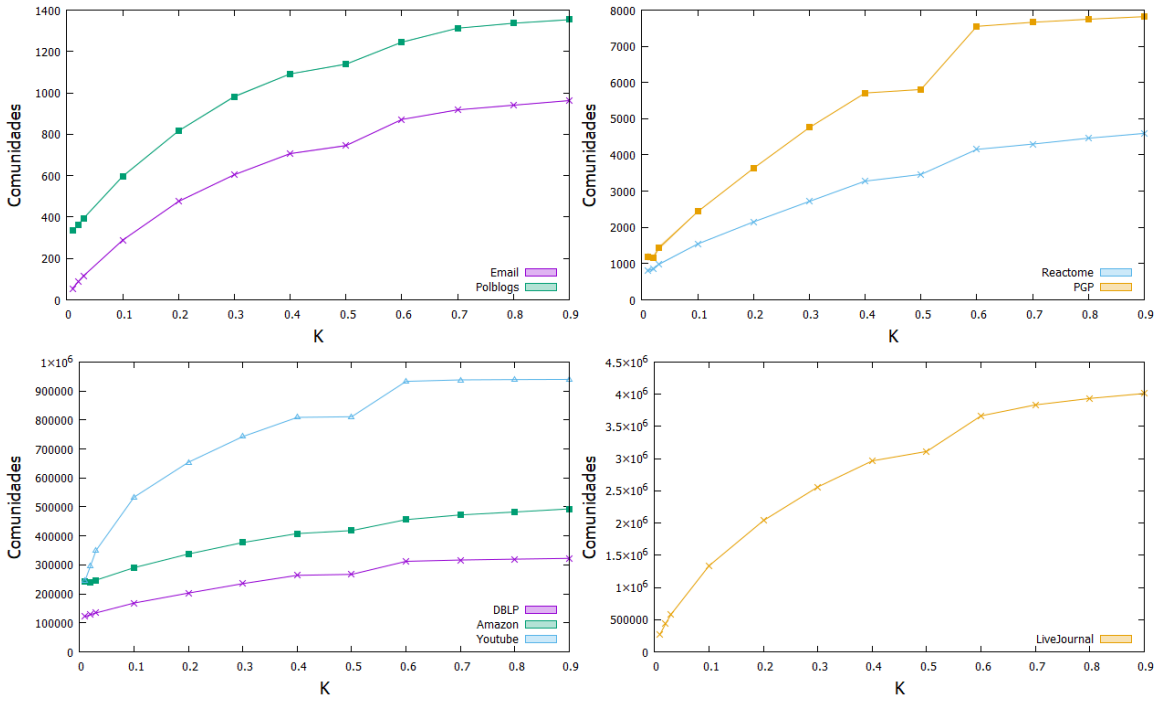

Figura 2. Número de comunidades geradas pelo MRAV em função de K.

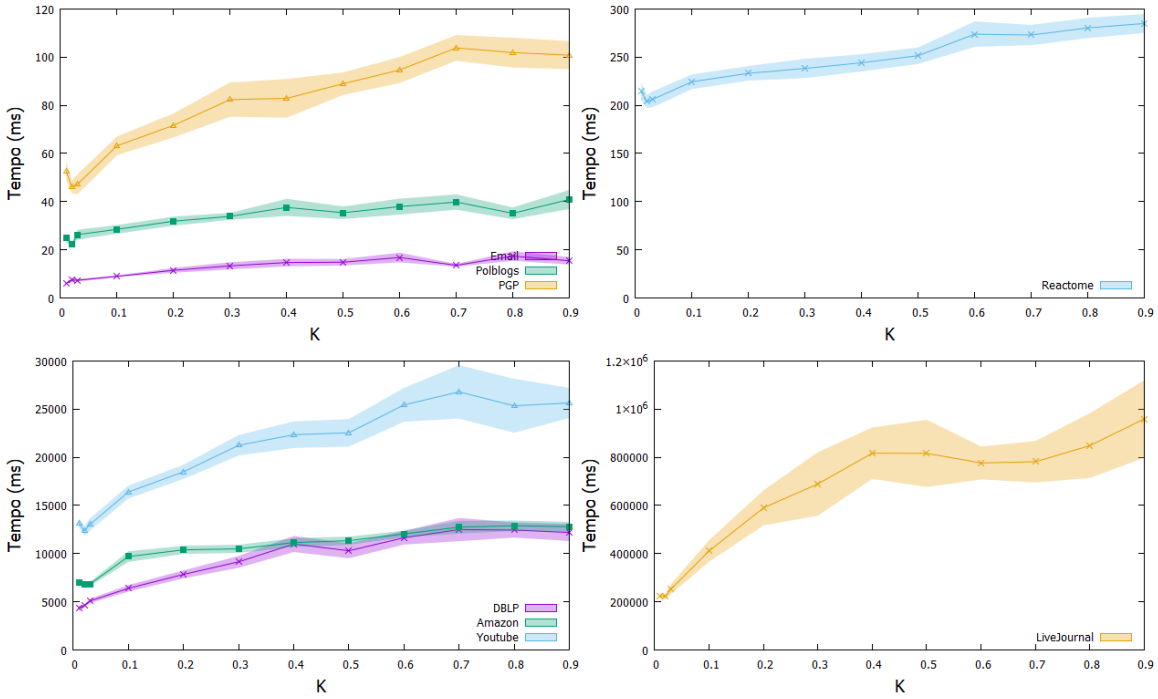

Figura 3. Tempo de execução consumido pelo MRAV+ML em função de K.

explorar mais a questão da escolha de K, a próxima seção reporta resultados de testes comparativos entre o ML e o MRAV+ML, com $K = 0.01$, $K = 0.03$ e $K = 0.1$, no que tange ao tempo médio de execução e a modularidade das comunidades detectadas.

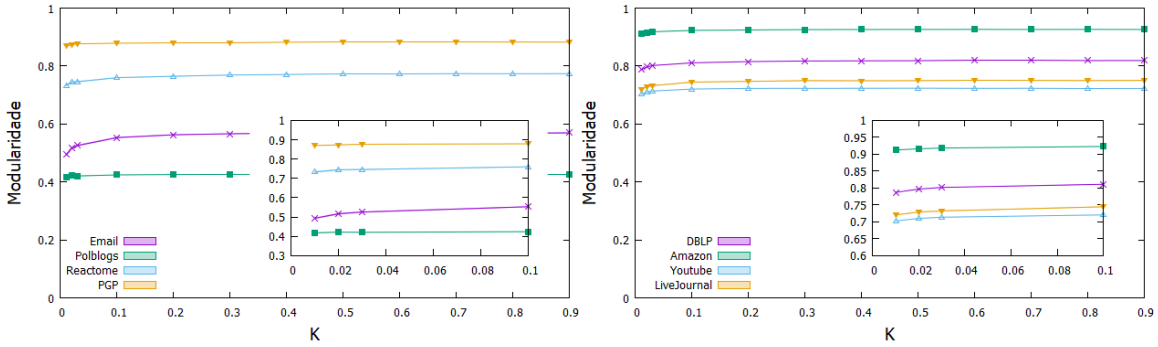

Figura 4. Modularidades das comunidades encontradas pelo MRAV+ML, em função de K.

3.3. Comparação de Desempenho: MRAV+ML vs ML

Nesta seção, são comparados, para $K \in \{0.01, 0.03, 0.1\}$, os desempenhos do MRAV+ML contra os do ML, em termos do tempo médio de execução e da modularidade média das comunidades detectadas. Os resultados encontrados para as 8 redes da Tabela 1 são reportados na Tabela 2. Os intervalos de confiança das 20 medidas experimentais são pequenos (e.g., $< 0.1\%$) e não foram mostrados na tabela para melhor visualização dos resultados. Os valores destacados em negrito na Tabela 2 são os melhores resultados.

Tabela 2. Desempenhos comparativos entre ML e MRAV+ML.

	Tempo (ms)				Modularidade			
		K				K		
		0.01	0.03	0.1		0.01	0.03	0.1
REDES	**ML**	**MRAV+ML**			**ML**	**MRAV+ML**		
Email	8.8	**6.0**	7.3	9.0	**0.567**	0.493	0.525	0.553
Polblogs	**13.6**	24.9	26.3	28.5	**0.426**	0.417	0.420	0.424
Reactome	**65.5**	214.3	206.2	224.5	**0.776**	0.733	0.745	0.759
PGP	70.2	**52.7**	47.2	63.2	**0.882**	0.870	0.876	0.878
DBLP	12200.3	**4350.0**	5083.3	6400.0	**0.820**	0.787	0.801	0.811
Amazon	11620.5	7000.0	**6845.1**	9700.0	**0.926**	0.912	0.918	0.922
Youtube	18278.8	13200.0	**13018.1**	16400.0	**0.720**	0.702	0.713	**0.720**
LiveJournal	426830	**224500**	253032	411350	**0.748**	0.720	0.732	0.744

Tabela 3. Taxas médias de redução de tempo de execução e modularidade (%).

Redes	DBLP	Amazon	Youtube	LiveJournal	Média
Tempo $K = 0.01$	64.3	39.7	27.7	47.4	44.8
Tempo $K = 0.03$	58.3	41.0	28.7	40.7	42.2
Modularidade $K = 0.01$	4.0	1.5	2.5	3.7	2.9
Modularidade $K = 0.03$	2.3	0.8	0.9	2.1	1.5

O MRAV+ML produziu modularidades inferiores ao ML em todos os grafos testados. Porém, tal redução se mostrou relativamente pequena. Nota-se ainda no MRAV+ML um natural decaimento de modularidade com o decaimento do valor de K.

O ML possui complexidade $\mathcal{O}(n \log n)$, enquanto o MRAV possui complexidade $\mathcal{O}(e)$, sendo n o número de vértices e e o número de arestas. Logo, é esperado que o desempenho do MRAV+ML seja superior para grafos esparsos. Isso é visto nos testes com redes pouco esparsas (Polblogs e Reactome) em que o MRAV+ML obteve piores resultados em relação ao ML. Outros dois resultados inferiores, em relação ao tempo médio de execução, foram obtidos em outras 2 redes menores (Email e PGP). Nesses casos, por o ML já apresentar uma complexidade próxima à linear, é esperado que sua eficiência já seja próxima da ótima, não havendo espaço para melhoria adicional de desempenho.

Entretanto, dentre os 4 maiores grafos (de 0.3 a 5 milhões de vértices), observa-se na Tabela 3 que, em comparação com o ML, o MRAV+ML[3] é capaz de detectar comunidades com uma queda em modularidade não superior a 4%, mas com uma redução não inferior a 28% no tempo médio de execução, podendo alcançar reduções expressivas como 64%, como no caso da rede DBLP. Em termos de medidas médias de desempenho do MRAV+ML, neste conjunto de redes, observa-se que aumentar K favorece a modularidade e penaliza o tempo de execução. Na média, por exemplo, para $K = 0.03$, observa-se uma redução de 42% no tempo de execução e uma perda de 1.5% na modularidade.

4. Conclusão e Trabalhos Futuros

Este artigo reportou uma investigação sobre detecção de comunidades em grafos. Foi proposto um Método Rápido de Agrupamento de Vértices (MRAV) que, em conjunto com o Método de Louvain (ML), é capaz de detectar comunidades em grafos com milhões de vértices, com mais rapidez do que o ML, ao custo de uma pequena perda na modularidade das comunidades encontradas.

No MRAV, a alocação de um vértice em um grupo é feita se seu percentual de conectividade com o grupo ultrapassar um limite $0 < K < 1$. O MRAV usa uma heurística gulosa que prioriza a velocidade da tarefa e entrega um grafo de tamanho reduzido, onde os grupos são os novos vértices. Esse resultado pode ser entendido como similar ao produzido pelas primeiras iterações do ML, só que de modo mais rápido. Portanto, é vantajoso um esquema multi-estágio em que o MRAV transforma o grafo original em um grafo reduzido sobre o qual o ML completa a detecção de comunidades.

Experimentos computacionais foram conduzidos envolvendo grafos associados a redes reais com número de vértices entre mil e cinco milhões. Para aferir a robustez do MRAV à escolha aleatória do vértice inicialmente alocado a um grupo, foram executadas 20 rodadas, para cada configuração experimental. Foram reportados, portanto, a média e o intervalo de confiança das medidas de desempenho feitas sobre as 20 rodadas.

Para redes com um grande número de vértices (0.3 a 5 milhões), o MRAV+ML é uma solução para a detecção de comunidades em que o parâmetro K regula as reduções médias de tempo de execução e de modularidade, em relação ao ML. Entretanto, a redução de tempo é muito mais substancial do que a perda de modularidade. Por exemplo, para $K = 0.03$, a redução média de tempo de execução é de 42%, ao custo de uma perda média de 1.5% na modularidade. Já para $K = 0.01$, a redução média de tempo de execução é de aproximadamente 45%, ao custo de uma perda média de 3% na modularidade.

[3]O código do MRAV+ML implementado em C++ e os grafos utilizados neste trabalho estão disponíveis em http://lps.lncc.br/index.php/demonstracoes/wperformance18

Como possíveis desdobramentos da investigação pode-se pensar em comparar o MRAV+ML com outros métodos de detecção de comunidades ou outros métodos de otimização do ML. Pode-se analisar mais detalhadamente o comportamento do MRAV na faixa $0 < K < 0.1$, com o intuito de encontrar um valor ótimo de K que minimize a perda de modularidade e o tempo de execução. Outra questão a ser explorada é a possibilidade de estender o escopo de aplicação do método para detectar comunidades em grafos ponderados ou grafos direcionados.

Agradecimentos

À CAPES, FAPERJ, FAPEMIG, FAPESP e ao CNPq, pelo apoio financeiro.

Referências

Adamic, L. A. e Glance, N. (2005). The political blogosphere and the 2004 US election: divided they blog. In *Proc. 3rd Int. Workshop on Link Discovery*, pages 36–43. ACM.

Aldecoa, R. e Marín, I. (2011). Deciphering network community structure by surprise. *PloS one*, 6(9):e24195.

Aldecoa, R. e Marín, I. (2013). Surprise maximization reveals the community structure of complex networks. *Scientific Reports*, 3:1060.

Bai, L., Cheng, X., Liang, J., e Guo, Y. (2017). Fast graph clustering with a new description model for community detection. *Information Sciences*, 388:37–47.

Bhowmick, S. e Srinivasan, S. (2013). A template for parallelizing the louvain method for modularity maximization. In *Dynamics on and of Complex Networks, Volume 2*, pages 111–124. Springer.

Blondel, V. D., Guillaume, J.-L., Lambiotte, R., e Lefebvre, E. (2008). Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2008(10):P10008.

Boguná, M., Pastor-Satorras, R., Díaz-Guilera, A., e Arenas, A. (2004). Models of social networks based on social distance attachment. *Physical Review E*, 70(5):056122.

Bondy, J. A., Murty, U. S. R., et al. (1976). *Graph theory with applications*. Elsevier.

Chakraborty, T., Dalmia, A., Mukherjee, A., e Ganguly, N. (2017). Metrics for community analysis: A survey. *ACM Computing Surveys (CSUR)*, 50(4):54.

Fazlali, M., Moradi, E., e Malazi, H. T. (2017). Adaptive parallel Louvain community detection on a multicore platform. *Microprocessors and Microsystems*, 54:26–34.

Fortunato, S. (2010). Community detection in graphs. *Physics reports*, 486(3-5):75–174.

Fortunato, S. e Barthelemy, M. (2007). Resolution limit in community detection. *Proceedings of the National Academy of Sciences*, 104(1):36–41.

Girvan, M. e Newman, M. E. (2002). Community structure in social and biological networks. *Proceedings of the National Academy of Sciences*, 99(12):7821–7826.

Guimera, R., Danon, L., Diaz-Guilera, A., Giralt, F., e Arenas, A. (2003). Self-similar community structure in a network of human interactions. *Physical Review E*, 68(6):065103.

Harenberg, S., Bello, G., Gjeltema, L., Ranshous, S., Harlalka, J., Seay, R., Padmanabhan, K., e Samatova, N. (2014). Community detection in large-scale networks: a survey and empirical evaluation. *Wiley Interdisciplinary Reviews: Computational Statistics*, 6(6):426–439.

Joshi-Tope, G., Gillespie, M., Vastrik, I., D'Eustachio, P., Schmidt, E., de Bono, B., Jassal, B., Gopinath, G., Wu, G., Matthews, L., et al. (2005). Reactome: a knowledgebase of biological pathways. *Nucleic Acids Research*, 33(suppl_1):D428–D432.

Khan, B. S. e Niazi, M. A. (2017). Network community detection: A review and visual survey. *CoRR*, abs/1708.00977.

Kwak, H., Lee, C., Park, H., e Moon, S. (2010). What is twitter, a social network or a news media? In *Proc. 19th Int. Conf. on World Wide Web*, pages 591–600. ACM.

Lancichinetti, A. e Fortunato, S. (2009). Community detection algorithms: a comparative analysis. *Physical Review E*, 80(5):056117.

Leskovec, J. e Krevl, A. (2014). SNAP Datasets: Stanford large network dataset collection. `http://snap.stanford.edu/data`.

Leskovec, J., Lang, K. J., e Mahoney, M. (2010). Empirical comparison of algorithms for network community detection. In *Proc. 19th Int. Conf. on World Wide Web*, pages 631–640. ACM.

Mislove, A., Marcon, M., Gummadi, K. P., Druschel, P., e Bhattacharjee, B. (2007). Measurement and analysis of online social networks. In *Proc. 7th ACM SIGCOMM Conf. on Internet Measurement*, pages 29–42. ACM.

Newman, M. E. (2006). Modularity and community structure in networks. *Proc. National Academy of Sciences*, 103(23):8577–8582.

Ozaki, N., Tezuka, H., e Inaba, M. (2016). A simple acceleration method for the Louvain algorithm. *Int. Journal of Computer and Electrical Engineering*, 8(3):207.

Peng, C., Kolda, T. G., e Pinar, A. (2014). Accelerating community detection by using k-core subgraphs. *arXiv preprint arXiv:1403.2226*.

Satuluri, V., Parthasarathy, S., e Ruan, Y. (2011). Local graph sparsification for scalable clustering. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 721–732.

Yang, J. e Leskovec, J. (2015). Defining and evaluating network communities based on ground-truth. *Knowledge and Information Systems*, 42(1):181–213.

Zhao, Y. (2017). A survey on theoretical advances of community detection in networks. *Wiley Interdisciplinary Reviews: Computational Statistics*, 9(e1403):1–13.