Algoritmos para o Problema de Particionamento

Este exemplar corresponde à redação final da Dissertação devidamente corrigida e defendida por Thiago de Paulo Faleiros e aprovada pela Banca Examinadora.

Campinas, 16 de dezembro de 2010.

Prof. Dr. Eduardo Candido Xavier
Instituto de Computação, Unicamp
(Orientador)

Dissertação apresentada ao Instituto de Computação, UNICAMP, como requisito parcial para a obtenção do título de Mestre em Ciência da Computação.

FICHA CATALOGRÁFICA ELABORADA PELA BIBLIOTECA DO IMECC DA UNICAMP
Bibliotecária: Maria Fabiana Bezerra Müller – CRB8 / 6162

Faleiros, Thiago de Paulo
F184a Algoritmos para o problema de particionamento/Thiago de Paulo Faleiros-- Campinas, [S.P. : s.n.], 2010.

Orientador : Eduardo Candido Xavier.
Dissertação (mestrado) - Universidade Estadual de Campinas, Instituto de Computação.

1.Particionamento. 2.Algoritmos. 3.Otimização combinatória. 4.Metaheuristica. 5.Teoria da computação. I. Xavier, Eduardo Candido. II. Universidade Estadual de Campinas. Instituto de Computação. III. Título.

Título em inglês: Algorithms for partitioning problem

Palavras-chave em inglês (Keywords): 1. Partitioning. 3. Algorithms. 4. Combinatorial optimization. Metaheuristics. 5. Theory of computing.

Titulação: Mestre em Ciência da Computação

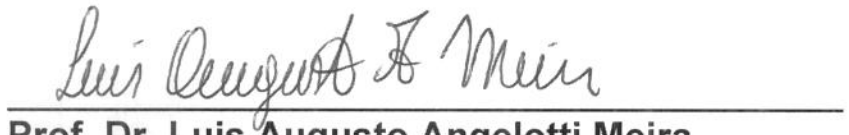
Banca examinadora: Prof. Dr. Eduardo Candido Xavier (IC – UNICAMP)
Prof. Dr. Luis Augusto Angelotti Meira (Univ. Fed. de São Paulo)
Prof. Dr. Guilherme Pimentel Telles (IC - UNICAMP)

Data da defesa: 16/12/2010

Programa de Pós-Graduação: Mestrado em Ciência da Computação

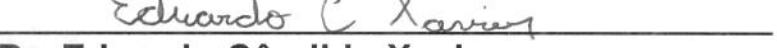
TERMO DE APROVAÇÃO

Dissertação Defendida e Aprovada em 16 de dezembro de 2010, pela Banca examinadora composta pelos Professores Doutores:



Prof. Dr. Luis Augusto Angelotti Meira
UNIFESP

Prof. Dr. Guilherme Pimentel Telles
IC / UNICAMP



Prof. Dr. Eduardo Cândido Xavier
IC / UNICAMP

Instituto de Computação
Universidade Estadual de Campinas

Algoritmos para o Problema de Particionamento

Thiago de Paulo Faleiros[1]

Dezembro de 2010

Banca Examinadora:

- Prof. Dr. Eduardo Candido Xavier
  Instituto de Computação, Unicamp (Orientador)
- Prof. Dr. Luis Augusto Angelotti Meira
  Universidade Federal de São Paulo
- Prof. Dr. Guilherme Pimentel Telles
  Instituto de Computação, Unicamp
- Prof. Dr. Orlando Lee
  Instituto de Computação, Unicamp (Suplente)
- Prof. Dr. José Coelho de Pina
  Instituto de Matemática e Estatística, USP (Suplente)

[1]Suporte financeiro de: Bolsa da FAPESP (processo 2008/06878-0) 2008–2010

Resumo

Investigamos Problemas de Particionamento de objetos que têm relações de similaridade entre si. Instâncias desses problemas podem ser representados por grafos, em que objetos são vértices e a similaridade entre dois objetos é representada por um valor associado à aresta que liga os objetos. O objetivo do problema é particionar os objetos de tal forma que objetos similares pertençam a um mesmo subconjunto de objetos. Nosso foco é o estudo de algoritmos para clusterização em grafos, onde deve-se determinar clusteres tal que arestas ligando vértices de clusteres diferentes tenham peso baixo e ao mesmo tempo as arestas entre vértices de um mesmo cluster tenha peso alto. Problemas de particionamento e clusterização possuem aplicações em diversas áreas, como mineração de dados, recuperação de informação, biologia computacional, entre outros. No caso geral estes problemas são NP-Difíceis. Nosso interesse é investigar algoritmos eficientes (com complexidade de tempo polinomial) e que gerem boas soluções, como Heurísticas, Metaheurísticas e Algoritmos de Aproximação. Dentre os algoritmos estudados, implementamos os mais promissores e fazemos uma comparação de seus resultados utilizando instâncias geradas computacionalmente. Por fim, propomos um algoritmo que utiliza a metaheurística GRASP para o problema considerado e mostramos que, para as instâncias de testes geradas, nosso algoritmo obtém melhores resultados.

Abstract

In this work we investigate Partitioning Problems of objects for which a similarity relations is defined. Instance to these problems can be represented by graphs where vertices are objects, and the similarity between two objects is represented by a value associated with an edge that connects objects. The problem objective is to partition the objects such that similar objects belong to the same subset of objects. We study clustering algorithms for graphs, where clusters must be determined such that edges connecting vertices of different clusters have low weight while the edges between vertices of a same cluster have high weight. Partitioning and clustering problems have applications in many areas, such as data mining, information retrieval, computational biology, and others. Many versions of these problems are NP-Hard. Our interest is to study efficient algorithms (with polynomial time complexity) that generate good solutions, such as Heuristics, Approximation Algorithms and Metaheuristics. We implemented the most promising algorithms and compared their results using instances generated computationally. Finally, we propose a GRASP based algorithm for the partition and clustering problem and show that, for the generated test instances, our algorithm achieves better results.

Agradecimentos

Primeiramente agradeço a Deus, por me dar força de vontade e saúde para concluir este trabalho.

Agradeço a minha família, especialmente meus pais, Vilma e João, por serem a base da minha vida e assim, me darem confiança de buscar novos desafios. Ao meu irmão Wesley, agradeço a compreensão, incentivo e toda a colaboração que ele ofereceu.

Ao meu orientador, o prof. Eduardo Cândido Xavier, que com enorme paciência corrigiu todos os textos que eu produzi, mostrando os erros e como melhorá-los. Graças a essa paciência pude perceber o quanto ainda devo melhorar.

Agradeço a todos os amigos que fizeram a minha vida na UNICAMP bem mais feliz. Aos amigos mais próximos, que auxiliaram no desenvolvimento desse trabalho ou simplesmente companheiros em todos os momentos. Em especial aos amigos: Lehilton (muito obrigado pela ajuda Lele), Thiago, André, Aline, Sheila e Robson. A Alessandra pelo incentivo, amizade, paciência, companheirismo e amor incondicional. Devo agradecer também a motivação dada por minha prima Anna Carolina, que, além de ser a embaixadora da família Faleiros em Campinas, me dava ânimo nos momentos de desespero.

E finalmente, agradeço à Fundação de Amparo à Pesquisa do Estado de São Paulo (FAPESP) pelo suporte financeiro.

Sumário

Lista de Figuras

Capítulo 1

Introdução

Problemas de otimização tem como objetivo encontrar um ponto máximo ou mínimo de uma função definida sobre um certo domínio. Em problemas de otimização combinatória temos que o espaço de soluções para uma instância é tipicamente finito e pode facilmente ser enumerado. Apesar disso, tentar encontrar a melhor solução de um problema de otimização combinatória, testando todos os elementos do domínio, mostra-se inviável na prática pois o domínio é exponencial em relação ao tamanho da instância. Além disso, sob a hipótese de que P≠NP, não é esperado encontrar algoritmos exatos eficientes (com complexidade de tempo polinomial) para problemas NP-difíceis.

Em resposta à dificuldade computacional de se encontrar soluções ótimas para muitos dos problemas de otimização combinatória, foram elaborados métodos que abrem mão da otimalidade em troca de soluções consideradas "boas" e que possam ser obtidas eficientemente. Em muitas ocasiões, percebe-se que, na falta de uma solução ótima do problema, pode-se utilizar a melhor solução disponível. Com isso, métodos eficientes que encontrem soluções não necessariamente ótimas, mas suficientemente boas podem ser muito úteis. Dentre esses métodos, destacamos heurísticas, programação inteira e algoritmos aproximados e probabilísticos.

Em programação inteira, formulamos o problema considerado como um programa linear onde algumas ou todas as suas variáveis devem assumir valores inteiros. A resolução de um programa linear inteiro (PLI) geralmente se dá por meio da resolução relaxada do PLI, e, em seguida, da aplicação de um algoritmo de *branch and bound* para obtenção de uma solução inteira. Outros métodos também podem ser usados para obtenção de uma solução inteira. Durante a busca por uma solução inteira válida, pode-se varrer o espaço de busca até que uma solução ótima seja encontrada (e provada ótima), ou pode-se achar uma solução inteira suficientemente boa e parar com a busca. Nesse último caso, não se pode afirmar que a solução seja ótima. Mais detalhes sobre programação inteira podem ser encontrados em [47, 56].

Outra estratégia para lidar com tais problemas de otimização combinatória são os Algoritmos de Aproximação. Algoritmos de Aproximação foram desenvolvidos para superar a dificuldade que há na tentativa de tratar problemas NP-Difíceis. Apesar desses algoritmos não necessariamente retornarem uma solução ótima para o problema, espera-se que sejam encontradas, eficientemente, soluções com boa qualidade. Um diferencial que há nos algoritmos de aproximação é a garantia de sempre encontrar uma solução cujo o valor guarda uma relação pré-estabelecida com o valor ótimo.

Quanto as heurísticas, elas englobam várias técnicas de resolução de problemas que obtém soluções não necessariamente ótimas. Dentre as várias técnicas, citamos a busca tabu, algoritmos genéticos e GRASP. Para mais informações sobre heurísticas, veja [45]. Heurísticas fornecem soluções sem um limite formal de qualidade, ao contrário de soluções geradas por algoritmos aproximados. Ainda assim, na prática, o uso de heurísticas tem sido promissor para a obtenção de soluções de boa qualidade.

Considerando os métodos para se encontrar soluções boas para os vários problemas de otimização combinatória, teremos, neste trabalho, interesse em investigar problemas de particionamento. O objetivo desse problema é particionar um dado conjunto de objetos de maneira que objetos "similares" pertençam à mesma parte, enquanto objetos em partes distintas sejam menos "similares" [40]. Tais problemas aparecem na literatura de maneira bem variada. Isso ocorre principalmente devido ao grande número de aplicações existentes, cada uma com uma definição apropriada da função de "similaridade". Os problemas considerados são NP-difíceis e, portanto, é improvável que sejam encontrados algoritmos exatos eficientes.

Existem diversas aplicações para problemas de particionamento. Em processamento de imagens por exemplo, há problemas de segmentação de imagens que podem ser resolvidos através de problemas de Particionamento. Neste caso, cada *pixel* da imagem corresponde a um objeto. A vizinhança de cada objeto corresponde aos *pixeis* vizinhos a este. Gostaríamos de simplificar ou alterar a representação de uma imagem em algo mais significativo e fácil de analisar. Sendo assim, os *pixeis* serão separados em regiões distintas. *Pixeis* de uma mesma região são similares em relação a alguma característica comum, como cor, intensidade ou textura. Regiões adjacentes são significadamente diferentes em relação a essas características. Podemos montar um **grafo de similaridade** considerando os pixeis como vértices/objetos e com arestas com pesos indicando relações de similaridade entre as características comuns dos *pixeis*. *Pixeis* vizinhos devem ter uma similaridade alta, pois possivelmente possuem características comuns, já *pixeis* não vizinhos devem ter similaridade baixa pois provavelmente não possuem as mesmas características.

Outro problema real que pode ser modelado como um problema de particionamento é encontrar subconjuntos de páginas da Web que estão relacionadas entre si. Com o aumento de informações divulgadas na Internet, tornou-se importante considerar a relação

entre as páginas da Web para a busca de informações mais específicas. No passado, métodos de busca de informação (*information retrieval*) consideravam documentos como conjuntos de palavras (em geral baseados em modelo vetorial) de maneira independente. Hoje, como o número de documentos disponíveis na Internet é significativamente maior, é importante também levar em conta as relações obtidas pelos links apontados por cada página. Atualmente, há mais de 10 bilhoes de páginas na internet e cada uma contém em torno de 10 links para outras páginas. Além disso, a estrutura e conteúdo da Internet muda rapidamente: mais de 20% das páginas mudam por dia e 10% das páginas têm no máximo uma semana de criação [13, 51]. Tais números indicam que é necessário buscar algoritmos rápidos para instâncias gigantescas.

Outras aplicações relevantes para problema de particionamento ocorrem na divisão de programas em partes (trechos de código) para serem executadas em computadores paralelos (ou computadores com vários núcleos), no Particionamento em Redes Wireless Ad Hoc [49], e na análise de dados de uma expressão de genes baseada em partição [59, 8].

1.1 Organização da Dissertação

Inicialmente, no Capítulo 2, apresentaremos as funções de avaliação de particionamento consideradas neste trabalho e uma breve discussão sobre cada uma delas.

No Capítulo 3 é discutido os algoritmos mais promissores para o problema de particionamento em grafos. Inicialmente, discuti-se os conceitos gerais de Clusterização Espectral (*Spectral Clustering*) para que, em seguida, possamos apresentar algoritmos específicos.

No Capítulo 4 propomos um algoritmo utilizando a metaheurística GRASP para o problema tratado neste trabalho.

Além disso, no Capítulo 5, descreveremos como as instâncias de testes foram geradas, o procedimento para a realização dos testes computacionais e a comparação dos algoritmos estudados neste trabalho.

Por fim, no Capítulo 6, são feitas as considerações finais e sugestões para trabalhos futuros.

1.2 Problemas de Clusterização e Particionamento

Vamos considerar o problema de *particionamento* de um grafo. Neste trabalho, também chamaremos esse problema de *clusterização*. Temos um grafo $G = (V, E)$ com pesos nas arestas que indicam a similaridade entre os pares de vértices. O objetivo é encontrar uma partição de vértices, de forma que cada parte, ou cluster, tenha arestas entre vértices internos com peso alto e o peso das arestas entre vértices de clusteres diferentes seja

baixo.

De maneira trivial, podemos varrer exaustivamente o conjunto de todos os particionamentos possíveis, na busca pelo particionamento ótimo de um grafo. Entretanto, para isso, é necessário lidar com uma importante questão: o que exatamente é um particionamento ótimo de um grafo? Para responder a essa questão, é necessário definir uma função de avaliação do particionamento. Há vários trabalhos que investigam essas funções. As funções de avaliação mais conhecidas, no âmbito de clusterização, em geral, consideram que os objetos estão representados como pontos no espaço $\mathbb{R}^n$ e que cada cluster contenha um vértice central. Dessa forma, podemos medir um particionamento calculando a soma do quadrado da distância de cada ponto até o centro mais próximo. Podemos também, calcular a soma da distância de cada ponto até o centro mais próximo ou calcular a maior distância de um ponto até o centro mais próximo. Quanto menores forem esses valores calculados para uma mesma instância, melhor será a qualidade do particionamento.

Um fato importante sobre as funções de avaliação encontradas na literatura é que não há um consenso sobre qual a melhor função para o problema de particionamento. Todas, de alguma forma, possuem exemplos de instâncias que apresentam partições diferentes daquelas percebidas intuitivamente, ou seja, dado uma instância do problema de particionamento com uma partição que, intuitivamente é a "melhor" possível, o algoritmo que otimiza uma dada função de avaliação encontra uma outra partição. No artigo [34] são discutidas falhas em algumas funções de avaliação existentes e também é desenvolvida uma função bicriteriosa com o objetivo de evitar tais falhas.

Não é encontrada uma função de avaliação completamente aceita para o problema de particionamento em grafos. Entretanto, dado um particionamento $C = \{C_1, ..., C_k\}$ de V, dizemos que ele é um "bom" particionamento se cada subgrafo induzido por cada C_i é denso, para $1 \leq i \leq k$, e, existe pouca conectividade entre os vértices desse subgrafo e os vértices no resto do grafo.

Essa relação entre os vértices internos de um subgrafo e todos os outros vértices apresenta um paradigma geral para particionamento. Esse paradigma relaciona a densidade intracluster, definida como a soma dos pesos das arestas internas do subgrafo induzido por C_i, à esparsidade intercluster, definida como a soma das arestas que saem do subgrafo, ou seja, as arestas do corte induzido pelo particionamento.

Se definirmos uma função não negativa f, que mede a densidade intracluster e uma função g, também não-negativa, que mede a esparsidade intercluster, podemos definir um bom indício para mensurar a qualidade de um cluster. Denotaremos esse indício de $index(C)$ [12], onde C representa um subgrafo induzido. A fim de normalizar a dimensão de $index(C)$, é definida outra função não negativa, $N(G)$, dependente apenas do grafo de entrada. A função $N(G)$ pode ser a soma das funções f e g aplicadas a todos os

particionamentos do grafo, por exemplo, segue a composição da função $index(C)$.

$$index(C) = \frac{f(C) + g(C)}{N(G)} \tag{1.1}$$

Nosso objetivo é encontrar partições que maximizem a densidade intracluster e que também maximizem a esparsidade intercluster.

1.2.1 Problema Correlato

Existem problemas que são bastante semelhantes ao particionamento. Um problema correlato, que destacaremos, é o problema de classificação. Devido à similaridade entre a classificação e o particionamento, muitas das estratégias para encontrar uma solução para o problema de classificação podem ser úteis para o problema de particionamento. A seguir, descrevemos um pouco mais sobre o problema de classificação.

Problema de Classificação

Em problemas de classificação, temos um conjunto P de n objetos que desejamos classificar e um conjunto L de k possíveis *labels*. Uma atribuição de *labels* para P é uma função $f : P \to L$, que atribui um *label* para cada objeto. Considere também um grafo $G = (P, E)$ com pesos nas arestas, sendo que cada vértice corresponde a um elemento de P.

A qualidade da classificação dos objetos é baseada na contribuição de dois conjuntos de termos. O primeiro termo se refere ao custo de se atribuir um *label* $i \in L$ para cada objeto $p \in P$. Denotamos por $c(p, i)$ o custo de atribuição de um *label* $i \in L$ a um objeto $p \in P$. O segundo termo é o valor pago para separar pares de objetos em *labels* distintos, o que é calculado da seguinte forma: dada uma função de distância $d(i, j)$ entre *labels* i, j de L e o peso w_e de cada aresta $e = (p, q)$ de E, o custo da separação é $w_e d(f(p), f(q))$. O custo total de uma atribuição f é

$$Q(f) = \sum_{p \in P} c(p, f(p)) + \sum_{e=(p,q) \in E} w_e d(f(p), f(q)). \tag{1.2}$$

No problema de classificação o objetivo é minimizar o custo $Q(f)$.

Existem muitas variações do problema de classificação. Em geral, essas variações dizem respeito a diferentes funções de distância entre elementos do conjunto L.

Dizemos que (L, d) é uma métrica se, e somente se, para todo trio $i, j, k \in L$, valem $d(i, i) = 0$, $d(i, j) > 0$, se $i \neq j$, $d(i, j) = d(j, i)$ e $d(i, j) + d(j, k) \geq d(i, k)$. Além disso, dizemos que (L, d) é uma *métrica uniforme* quando $d(i, j) = 1$, se $i \neq j$, e $d(i, j) = 0$ caso contrário. Em uma *métrica linear*, os *labels* são numeradas consecutivos, *i.e*, $L = \{1, 2, ..., k\}$, e a função de distância é definida como $d(i, j) = |i - j|$. Uma *métrica linear*

truncada é definida como uma função $d(i,j) = min\{M, |i-j|\}$, onde M é um valor máximo.

1.3 Revisão Bibliográfica

A meta da revisão bibliográfica foi fazer um levantamento de trabalhos que apresentam algoritmos para o problema de particionamento. Entre os vários artigos encontrados, demos preferência àqueles que consideravam a entrada do problema como um grafo e que, a princípio, poderiam apresentar uma melhor aplicabilidade prática (algoritmos rápidos). Embora nem todos os trabalhos apresentem algoritmos com garantia de aproximação, acreditamos que eles podem levar a bons resultados práticos.

Nesta seção, descrevemos os vários artigos estudados.

1.3.1 Algoritmos Heurísticos

A Graph-Based Clustering Method and Its Applications [23] - 2007: Neste artigo, são tratados problemas de clusterização em que apenas parte dos dados são interessantes para a aplicação. Isso acontece quando entre os elementos de interesse estão presentes "ruídos". "Ruídos" são elementos que não fazem parte dos dados de interesse e são bastante dissimilares a qualquer outro elemento. Algoritmos que pressupõem uma distribuição esférica (ou Gaussiana) falham ao detectar clusteres em dados com ruídos, pois os "elementos ruidosos" podem estar inclusos dentro da mesma distribuição.

Algoritmos que representam os elementos como um grafo são bons para tratar problemas de clusterização com "elementos ruidosos". O algoritmo de clusterização proposto no artigo [23] (*Fuzzy C-means MST Clustering algorithm* – FMC) tem como entrada um grafo, onde cada elemento é representado por um vértice e a relação entre cada par de elementos é uma aresta. O grafo possui pesos nas arestas que relacionam pares de elementos. Quanto maior o peso da aresta que representa a relação do par, maior será a similaridade entre os elementos desse par.

O algoritmo FMC é baseado no algoritmo descrito por Zahn [60]. O algoritmo de Zahn constrói uma árvore geradora mínima do grafo. Depois disso, arestas com peso inferior a um limiar pré-definido λ são removidas. Cada componente conexo da árvore geradora mínima define os clusteres resultantes do algoritmo.

O problema da utilização do algoritmo proposto por Zahn é a necessidade de determinar um limiar λ. Esse limiar é bastante específico do problema; grafos diferentes podem requerer valores de λ diferentes. A proposta do artigo é então descartar a necessidade deste limiar e, para isso, o problema é reformulado como o particionamento do conjunto de arestas da árvore geradora mínima em dois clusteres, de acordo com o peso. Dentre

os dois clusteres gerados, um cluster conterá as arestas com os menores pesos e terá as arestas preservadas na árvore geradora mínima. As arestas do outro cluster, que têm peso alto, serão removidas da árvore geradora mínima.

O procedimento do algoritmo FMC é semelhante ao algoritmo de Zahn. A diferença está na ausência do limiar λ para determinar as arestas com peso alto. No final, após a remoção das arestas, cada componente conexo corresponderá a um cluster.

Para particionar as arestas da árvore geradora mínima em dois clusteres, é utilizado o algoritmo *Fuzzy C-Means* (FCM). FCM é uma técnica de clusterização baseada na minimização da seguinte função objetivo:

$$J_m = \sum_{i=1}^{N} \sum_{j=1}^{C} u_{ij}^m (x_i - c_j)^2, \tag{1.3}$$

onde m é um número real maior do que um, x_i é o peso da i-ésima aresta da árvore geradora mínima, c_j é o valor do centro do cluster j, u_{ij} é o grau de relacionamento da i-ésima aresta com o cluster j, C é o número de clusteres (no caso $C = 2$) e N é o número de objetos (arestas) a serem clusterizados.

O algoritmo FCM prossegue em um método iterativo com o objetivo de minimizar a Função (1.3) e está descrito no algoritmo 1. Inicialmente, os graus de relacionamento são atribuídos aleatoriamente. A cada iteração o grau de relacionamento u_{ij} e o centro do cluster c_j são atualizados da seguinte maneira:

$$u_{ij} = \frac{1}{\sum_{k=1}^{C} \left(\frac{x_i - c_j}{x_i - c_k} \right)^{\frac{2}{m-1}}} \quad \text{e} \quad c_j = \frac{\sum_{i=1}^{N} u_{ij}^m x_i}{\sum_{i=1}^{N} u_{ij}^m}. \tag{1.4}$$

O algoritmo para quando:

$$\max_{ij} \{ |u_{ij}^{(k+1)} - u_{ij}^{(k)}| \} < \epsilon,$$

onde ϵ é um parâmetro que define o critério de parada e os graus de relacionamento

correspondem a seus valores nas iterações de índices (k) e $(k+1)$.

Algoritmo 1: Fuzzy C-Mean (FCM)

1 Inicialize a matriz $U = [u_{ij}]$ aleatoriamente, obtendo $U^{(0)}$;

2 Na k-ésima iteração, calcule o vetor central $C^{(k)} = [c_j]$ com $U^{(k)}$:

$$c_j = \frac{\sum_{i=1}^{N} u_{ij}^m x_i}{\sum_{i=1}^{N} u_{ij}^m};$$

3 Atualize $U^{(k)}$, obtendo $U^{(k+1)}$:

$$u_{ij} = \frac{1}{\sum_{k=1}^{C} \left(\frac{x_i - c_j}{x_i - c_k} \right)^{\frac{2}{m-1}}};$$

4 Se $\max\{|u_{ij}^{(k+1)} - u_{ij}^{(k)}|\} < \epsilon$, então PARE, caso contrário retorne ao passo 2 ;

Após a realização do algoritmo FCM, todas as arestas da árvore geradora mínima são separadas em dois clusteres. Cada aresta i será atribuída a algum cluster r, tal que:

$$r = \arg\max_j u_{ij}.$$

Em seguida, são removidas da árvore geradora mínima todas as arestas que pertencem ao cluster s de maior centro, isto é:

$$s = \arg\max_j c_j$$

O método proposto no artigo pode ser descrito da seguinte maneira:

1. Construa um grafo completo $G = (V, E)$ a partir de um conjunto $S \subseteq \mathbb{R}^d$ tal que
   - cada vértice $v_i \in V$ corresponda um elemento $i \in S$;
   - cada aresta $e = (v_i, v_j) \in E$ possui um peso $w_e = d(i, j)$, onde $d(i, j)$ é a distância entre i e j no espaço $\mathbb{R}^d$.
2. Determine a árvore geradora mínima de G.
3. Remova da árvore geradora mínima as arestas peso maior que ϵ usando o algoritmo FCM.

Por fim, os clusteres resultantes do algoritmo são definidos a partir de cada componente conexo da árvore geradora mínima.

O valor ϵ deve ser determinado anteriormente, entretanto, é descrito no trabalho de Pasquale *et al.* [23] que o valor de ϵ é independente do grafo de entrada. Os testes do artigo foram realizados para $\epsilon = 0,5$.

Alem disso, o algoritmo FMC foi comparado com três outros algoritmos: *Markov Clustering Algorithm* (MCL), *Iterative Conductance Cutting Algorithm* (ICC) e o *Geometric MST Clustering Algorithm* (GMC). A comparação foi realizada comparando o desempenho dos algoritmos em instâncias reais. Dois domínios de aplicações foram consideradas: clusterização cenas de âncoras em vídeo de noticiário e detecção de microcalcificação em imagens de mamografia.

Para fins de comparação dos resultados, foram utilizadas medidas que avaliam a efetividade da detecção dos clusteres. Basicamente, essas medidas relacionam a quantidade de verdadeiros positivos, que são os clusteres corretamente detectados; o número de falsos positivos, que são os clusteres detectados que não fazem parte da solução ótima; e o número de falsos negativos, que são aqueles clusteres não detectados pelo algoritmo.

Para essas medidas, o algoritmo FCM apresentou resultados superiores aos outros algoritmos. O algoritmo ICC mostrou-se inadequado para esses tipos de problemas, tendo resultados inferiores. Os outros algoritmos, MCL e GMC apresentaram bons resultados.

1.3.2 Algoritmos Aproximados para os Problemas *K-Means*, *K-Median* e *K-Center*

Um dos métodos de clusterização mais estudados é o problema dos *k-means*. O problema é definido da seguinte forma. Dado um conjunto P de n pontos no espaço $\mathbb{R}^d$ com d dimensões e um inteiro k, o objetivo é determinar o conjunto de k pontos em $\mathbb{R}^d$, chamados centros, tais que a soma dos quadrado da distância de cada ponto de P até o centro mais próximo seja minimizada. Para o problema *k-median*, o objetivo é minimizar a soma da distância de cada ponto de P até o centro mais próximo. No problema do *k-center*, o objetivo é minimizar a maior distância de um ponto até o centro mais próximo.

A Local Search Approximation Algorithm for k-Means Clustering [35] - 2002: O algoritmo de aproximação apresentado neste artigo assume que é gerado um conjunto C contendo os candidados a centros, sendo que os k centros podem ser escolhidos desse conjunto. Matousek [43] mostrou que o conjunto C de candidatos a centros pode ser escolhido como uma $\epsilon-$aproximação para o conjunto de centroides com tamanho $O(n\epsilon^{-d}\log(\frac{1}{\epsilon}))$ e com tempo $O(n\log n + n\epsilon^{-d}\log\frac{1}{\epsilon})$. O algoritmo de aproximação opera selecionando um conjunto inicial S com os k centros selecionados do conjunto de centros candidatos C e, em seguida, tentando melhorar a solução com a troca de um

centro de $s \in S$ por um elemento $s' \in C - S$. Essa troca de centros é realizada até que nenhuma melhora ocorra. No artigo, é também apresentada uma maneira eficiente para realizar a troca. O fator de aproximação de $9 + \epsilon$ é obtido com a realização de várias trocas simultâneas, diferentemente da técnica de uma simples troca, que fornece uma $(25 + \epsilon)-$aproximação (trabalho de Arya [4]).

A constant-factor approximation algorithm for the k-median problem [14] - 1999: Neste artigo, é tratado o problema *k-median* com métrica. O algoritmo utiliza arrendondamento da solução relaxada de um programa linear. É demonstrado um fator de aproximação $4-$aproximação.

A Simple Linear Time $(1+\epsilon)-$Approximation Algorithm for k-Means Clustering in Any Dimensions [39] - 2004 Neste artigo, é apresentado um algoritmo $(1+\epsilon)-$aproximado para o problema dos *k-means*. Tratando k e ϵ como constantes (caso contrário o algoritmo é exponencial em k), o tempo de execução é de $O(nd)$, onde n é o número de pontos e d o número de dimensões. A idéia principal do algoritmo é que, dado um conjunto de pontos, seus centróides podem ser muito bem aproximados por amostras de um número constantes de pontos. O algoritmo utiliza um fato importante, chamado de $(k, \epsilon)-$reducibilidade. Seja P um conjunto de n pontos no espaço $\mathbb{R}^d$ e $\Delta_k(P)$ o custo da solução ótima para o problema dos *k-means* em relação a P. Um problema é dito $(k, \epsilon)-$redutível se $\Delta_{k-1}(P) \leq (1 + 32\epsilon)\Delta_k(P)$. Isso significa que a solução ótima com $k - 1$ médias é no máximo $(1 + O(e))$ da solução ótima com k médias.

Local Search Heuristics for k-median and Facility Location Problems [4] - 2002: Neste artigo, é apresentado um algoritmo que utiliza técnica de busca local para resolver o problema dos *k-median*. O procedimento de busca local tenta, em cada iteração, melhorar uma solução viável inicial. Para melhorar uma solução inicial, a única operação permitida é o procedimento de troca de centros. A troca de centros é simplesmente a remoção de um centro $s \in S$ (S é o conjunto de k centros da solução inicial) e a adição de um novo centro $s' \notin S$. É demonstrado que o procedimento de busca local com trocas simples (apenas um centro é trocado em cada iteração) tem um fator de aproximação 5. É demonstrado também uma generalização do algoritmo que considera múltiplas trocas de centros realizadas simultaneamente em cada iteração. Essas múltiplas trocas garantem um fator de aproximação de exatamente $3 + \frac{2}{k}$.

Approximation Algorithms for clustering and related problems [58] - *Tese de Doutorado* - 2006: Na tese de Xu [58] são apresentados algoritmos de aproximação para problemas correlatos à clusterização. Damos destaque para o problema dos k-centros

(*k-center*) com grupo de informações prescritas (*k-center problem with prescribed grouping information* - KCWPGI). Nesse problema, é dada como entrada uma coleção de conjuntos de pontos em $\mathbb{R}^d$, $S_1, S_2, ..., S_n$, e um inteiro $k \geq 1$. O objetivo é encontrar as k esferas congruentes de raio mínimo, $B_1^*, B_2^*, ..., B_k^*$, tais que, para cada conjunto de pontos S_i, $1 \leq i \leq n$, exista uma bola $B_{j_i}^*$ envolvendo todos os pontos em S_i.

Um primeiro algoritmo para KCWPGI apresentado na tese é uma $3-$aproximação com complexidade de tempo $O(m + n \log k)$, onde $m = \sum_{j=1}^{n} |S_j|$. O algoritmo funciona da seguinte forma: é dado como entrada um conjunto de n centros $C_B = \{c_1, ..., c_n\}$, tais que cada conjunto de pontos S_j esta contido em uma esfera com centro c_j e raio r_j; é escolhido aleatoriamente um ponto c_{i_1} de C_B como o centro do primeiro cluster e em seguida é escolhido o ponto c_{i_2} mais distante de c_{i_1} como o centro da segunda bola; esse processo de escolha é repetido até que todos os k centros sejam identificados. Uma vez que os k centros forem identificados, para cada $1 \leq j \leq k$, o cluster C_j conterá todo conjunto S_l cujo centro c_l é mais perto de c_{i_j} do que de qualquer outro centro.

Além do algoritmo acima, são apresentados também algoritmos com melhoras no fator de aproximação para o problema KCWPGI. As melhoras correspondem a um algoritmo $(1 + \sqrt{3})-$ aproximado e com tempo $O(km)$ e uma $(2 + \epsilon)-$aproximação com tempo $O(\frac{dm}{\epsilon} + 2^{O(k \log \frac{k}{\epsilon})} dn + (\frac{1}{\epsilon})^5 n)$.

Uma peculiaridade do problema estudado é a entrada. São dados como entrada n conjuntos de pontos. Os algoritmos devem calcular os n centros correspondentes a cada conjunto de pontos. Portanto, além do tempo para encontrar os k clusteres, também deve ser considerado o tempo para encontrar os n centros de cada conjunto de pontos.

Quick k-Median, k-Center, and Facility Location for Sparse Graphs. [53] - 2005: Neste trabalho, é apresentado um algoritmo com fator de aproximação constante e complexidade de tempo e espaço $\tilde{O}(m)$, sendo m o número de arestas do grafo $G = (V, E)$, dado como entrada. A notação $\tilde{O}$ significa que ignoramos o fator logarítmico. O fator de aproximação do algoritmo é de $12+o(1)$ para *k-median*, 2 para *k-center* e $3+o(1)$ para *facility location*. Segundo o próprio autor do trabalho [53], o algoritmo com fator de tempo e espaço de $\tilde{O}(m)$ para o problema *k-median* é bastante complicado e improvável de ter relevância na prática. Entretanto, no desenvolvimento do trabalho, também é apresentado um algoritmo simples para prover um conjunto com tamanho $O(k)$ de candidatos a centros. É garantido que esse conjunto contenha pelo menos um subconjunto de k centros com no máximo duas vezes o custo da solução ótima. Esse algoritmo pode ser usado como pré-processamento para outros algoritmos que resolvem o problema *k-median*. Utilizando o como pré-processamento do algoritmo proposto por Jain e Vazirani [33] para o problema *k-median*, é encontrado o fator de aproximação de $12 + o(1)$, com tempo de $\tilde{O}(kn)$.

The Reverse Greedy Algorithm For The Metric k-median Problem [16]. - 2006: O algoritmo *Reverse Greedy* para o problema *k-median* é um algoritmo bem simples e trabalha de forma gulosa. Podemos descrever o funcionamento do algoritmo da seguinte maneira: Primeiro, ele considera cada ponto como um centro. Em cada passo, é escolhido o vértice cuja a remoção deixe o menor valor da soma da distância de um ponto até o centro mais próximos ainda não removido. O algoritmo para quando restarem k centros. Neste artigo, a principal contribuição é provar que, se a função de distância entre os pontos obedecem a uma métrica, então o fator de aproximação do algoritmo está entre $\Omega(\frac{\log n}{\log \log n})$ e $O(\log n)$.

Approximating k-Median with Non-Uniform Capacities. [18] - 2005: Neste artigo, é considerado o problema *k-median* com capacidade. Como o problema *k-median* com capacidade é semelhante ao problema *facility location*, a notação para definir a entrada do *facility location* é utilizada na definição da entrada do problema *k-median* com capacidade. No artigo, a entrada do problema é o conjunto de clientes C (ou conjunto de pontos), um conjunto de potenciais facilidades (conjunto de centros), uma função não negativa u que atribui capacidade a cada facilidade, $u : F \to \mathbb{N}$, uma métrica d em $C \cup F$ e um inteiro positivo k. A saída desejável é um subconjunto de facilidades abertas $F' \in F$ de tamanho no máximo k e a atribuição de clientes para as facilidades abertas $\varphi : C \to F'$ tal que, para todo $i \in F'$, $|\varphi^{-1}(i)| \leq u(i)$. O objetivo é minimizar o custo da atribuição total $\sum_{j \in C} d(j, \varphi(j))$. No problema *facility location*, é incluido o custo de abrir cada facilidade.

No artigo, é apresentado um algoritmo aproximado para o problema *facility location* que utiliza técnica primal-dual. O mesmo algoritmo foi usado na aproximação do problema *k-median* com capacidade. O algoritmo é dividido em duas fases. Na primeira fase, o objetivo é encontrar uma solução viável no dual e, na segunda fase, a solução será convertida em uma solução que satisfaça as restrições relaxadas do primal. Na análise realizada no artigo, o algoritmo encontra uma solução com o custo no máximo 40 vezes o ótimo e a capacidade de cada facilidade pode ser extrapolada no máximo 50 vezes o valor permitido.

Approximation Algorithms for Metric Facility Location and k-Median Problems Using the Primal-Dual Schema and Lagrangian Relaxation. [33] - 2001: Neste artigo, é apresentado um algoritmo para o problema *facility location* com métrica (i.e. satisfaz desigualdade triangular) e o problema *k-median* com métrica. É alcançada uma garantia de aproximação de 3 para o problema de *facility location* com métrica e 6 para *k-median* com métrica. Os tempos de execução são respectivamente $O(m \log m)$ e $O(m \log m(L + \log(n)))$, onde m é número total de arestas, n o número de vértices e L é o numero de bits necessários para representar o custo de conexão. Para k-medianas, o

custo de conexão é o peso nas arestas.

A idéia principal do algoritmo é utilizar o esquema primal-dual para obter a aproximação. O algoritmo para resolver o problema *k-median* utiliza como sub-rotina o algoritmo que resolve o problema de localização de facilidades. Este consiste em duas fases. Na primeira fase, ele procura uma solução viável para o dual e determina o conjunto de arestas "justas" (arestas que ligam um cliente a facilidade) e as facilidades correspondentes temporariamente abertas. Na segunda fase, o objetivo é encontrar o conjunto de facilidades que serão abertas e encontrar uma atribuição que mapeia cada cliente a uma facilidade. Esta fase é necessário pois o algoritmo pode atribuir mais de uma facilidade para um mesmo cliente.

O problema *k-median* difere do problema da localização de facilidades em dois aspectos: não existe custo para abrir as facilidades (as facilidades serão centros) e existe um limite superior para o número de facilidades que podem ser abertas. A partir da formulação do programa linear do problema *facility location*, é formulado um esquema de programação linear para o problema *k-median*, alterando as restrições que diferem entre os dois problemas.

Considere o seguinte programa linear para o problema *k-median*. Teremos y_i uma variável binária que indica se a facilidade i é aberta e x_{ij} é uma variável binária denotando se haverá conexão entre o cliente j com a facilidade i. A primeira restrição garante que cada cliente é conectado a pelo menos uma facilidade, a segunda garante que essa facilidade seja aberta e a terceira garante que, no máximo, k facilidades sejam abertas. Abaixo segue a formulação primal.

$$\begin{array}{lll} \min & \sum_{i\in F, j\in C} c_{ij}x_{ij} & \\ s.a. & \sum_{i\in F} x_{ij} \geq 1 & \forall j \in C \\ & y_i - x_{ij} \geq 0 & \forall i \in F, j \in C \\ & \sum_{i\in F} -y_i \geq -k & \\ & x_{ij} \geq 0 & \forall i \in F, j \in C \\ & y_i \geq 0 & \forall i \in F \end{array} \tag{1.5}$$

O programa dual é:

$$\begin{array}{lll} \max & \sum_{j\in C} \alpha_j - zk & \\ s.a. & \alpha_j - \beta_{ij} \leq c_{ij} & \forall i \in F, j \in C \\ & \sum_{j\in C} \beta_{ij} \leq z & \forall i \in F \\ & \alpha_j \leq 0 & \forall j \in C \\ & \beta_{ij} \geq 0 & \forall i \in F, j \in C \\ & z \geq 0 & \end{array} \tag{1.6}$$

O método proposto no artigo não resolve o programa linear do problema *k-median*,

mas ao invés disso, resolve a formulação do problema *facility location*. Para perceber como explorar a similaridade entre esses dois problemas é feito o seguinte. Peguemos uma instância do problema *k-median* e o transformamos em uma instância do problema de *facility location* acrescentando um custo z para abrir cada facilidade. Em seguida, encontre uma solução ótima (x, y) para o programa linear primal do problema *facility location* e outra solução ótima (α, β) para o dual. Considerando que as soluções (x, y) e (α, β) acontecem para exatamente k facilidades, é fácil verificar que (x, y) e (α, β, z) são soluções ótimas da formulação linear para o problema *k-median* (formulações (1.5) e (1.6)) .

A solução retornada pela formulação do problema *facility location* não garante que exatamente k facilidades sejam abertas. Dessa forma, uma solução encontrada com mais de k facilidades abertas será inviável. Uma outra solução com menos de k facilidades também pode ser obtida e viável dentro da formulação *k-median*, entretanto, ainda será uma solução ruim. O problema agora é determinar como aplicar o algoritmo primal-dual proposto no artigo e garantir que sejam abertas k facilidades. Para isso, é necessário encontrar o valor de z que garanta que apenas k facilidades sejam abertas. Quando $z = 0$, o algoritmo irá abrir todas as facilidades, quando z é muito grande, serão abertas apenas uma facilidade. A proposta do artigo para encontrar o valor de z é realizar uma busca binária no intervalo $[0, nc_{max}]$, onde n é o número de vértices e c_{max} é o peso da maior aresta. Na busca, serão encontrados dois valores z_2 e z_1 que abrem respectivamente $k_2 > k$ e $k_1 < k$ facilidades e que também $z_1 - z_2 \leq (\frac{c_{min}}{12n^2})$, onde c_{min} é o valor da aresta de menor peso.

Aproximation algorithms for clustering to minimize the sum of diameters [20] - 2000: Neste artigo, é tratado o problema da soma do diâmetro mínimo. Esse problema é definido da seguinte forma. Seja um grafo $G = (V, E)$ com pesos nas arestas e um inteiro $k \leq |V|$. O objetivo do problema é encontrar uma partição de V em k subconjuntos $V_1, \ldots, V_k$ que minimize $\sum_{i=1}^{k} dia(V_i)$, onde a função $dia(V_i)$ é o peso da maior aresta no subgrafo induzido por V_i.

Um resultado importante deste artigo é a prova de inaproximabilidade para o problema da soma do diâmetro mínimo quando a atribuição de pesos nas arestas não obedece desigualdade triangular. Nesse caso, é mostrado que, a menos que $P = NP$, para qualquer $\rho \geq 1$, não existe algoritmo de aproximação de tempo polinomial com aproximação ρ, mesmo para o caso em que o número de clusteres é fixo no valor 3.

1.3.3 Algoritmos para Problemas de Partição de Grafos

Balanced Graph Partitioning [3] - 2004: Neste artigo, é tratado o problema de particionar um grafo $G = (V, E)$ em k subconjuntos de vértices, $V_1, V_2, ..., V_k$, tais que o custo das arestas que ligam vértices de diferentes subconjuntos seja mínimo. É investigado o problema da partição $(k, 1+\epsilon)-$balanceado, que é o problema de procurar a partição de custo mínimo de G em k partes tal que cada parte contenha no máximo $(1+\epsilon)\frac{n}{k}$ vértices.

O algoritmo consiste em dividir o grafo em dois subconjuntos de vértices de forma que os subconjuntos sejam balanceados e prosseguir recursivamente com as divisões. Cada decomposição recursiva do grafo corresponderá a uma parte de uma árvore binária. A raiz conterá todos os vértices do grafo, V, e cada folha conterá um vértice $v_t \in V$.

Em cada passo da divisão, o algoritmo tenta encontrar o $\epsilon-$balanceamento mínimo, isto é uma partição do conjunto de entrada V em duas partes com tamanho maior do que $\epsilon|V|$ que corta o número mínimo de arestas. É usado um algoritmo de aproximação que retorna um $\frac{\epsilon}{2}-$balanced cut, i.e. cada parte contém pelo menos $\frac{\epsilon}{2}|V|$ vértices. Isso pode ser feito em tempo $\tilde{\mathrm{O}}(\frac{n^2}{\epsilon})$. O algoritmo é uma $O(\log n^2)$-aproximação para o problema.

Partitioning Graphs into Balanced Components [38] - 2009 : O problema estudado neste artigo é a partição de grafo. Basicamente, esse problema consiste em dividir o grafo em k componentes de tamanhos semelhantes, minimizando o peso das arestas que ligam vértices de componentes distintos. É utilizada programação semidefinida e um algoritmo para arrendondamento. É apresentado um algoritmo que alcança uma $O(\sqrt{\log n \log k})$-aproximação.

Fast Approximate Graph Partitioning Algorithms [21] - 1999 : Neste artigo, é estudado o problema de partição de grafo com peso nos vértices. Podemos definir esse problema da seguinte forma: dado como entrada um grafo $G = (V, E)$ com capacidades nas arestas e peso nos vértices, tem-se, como objetivo, encontrar um subconjunto de arestas cuja remoção particionará o grafo em k componentes conexos, de tal forma que a soma dos pesos dos vértices de cada componente seja no máximo ρ ($\rho < 1$) vezes o somatório do peso de todos os vértices do grafo. É apresentado um algoritmo $O(\log n)$-aproximado. O algoritmo é baseado na técnica chamada *spread metrics*, que pode ser usada para resolver o problema da partição em grafos. Informalmente, um *spread metric* de um grafo é uma atribuição de tamanhos para às arestas (ou vértices) que satisfaz certas condições úteis no particionamento de grafos. Por exemplo, o volume do *spread metric* provê um limite inferior para o custo da solução do problema. O volume do *spread metric* é definido por $\sum_{e \in E} c(e)d(e)$, onde $d(e)$ é a função de *spread metric* que atribui um tamanho não-negativo à aresta e e $c(e)$ é a capacidade da aresta e. O *spread metric* para um *ρ-separator* pode ser obtido por meio de um programa linear, cuja solução atribui um

tamanho $d(e)$ a cada aresta $e \in E$. Após o cálculo do *spread metric*, é aplicado o algoritmo para particionar o grafo. O procedimento para particionamento utiliza a distância atribuída pelo *spread metric* para encontrar os vértices mais próximos dentro de uma esfera com volume de, no mínimo, ρ vezes o somatório do peso de todos os vértices do grafo.

Capítulo 2

Funções de Avaliação

Todos os algoritmos que serão descritos neste trabalho retornam um particionamento $\mathcal{C} = \{C_1, ..., C_k\}$. O que desejamos é poder avaliar os resultados e averiguar se um algoritmo produz um "bom" particionamento. Portanto, funções de avaliação devem ser definidas e, além disso, elas devem corresponder ao paradigma principal deste trabalho – maximizar a densidade intracluster e maximizar a esparsidade intercluster.

Vamos definir nesta seção as funções de avaliação utilizadas para determinar um "bom" particionamento. Antes disso, definiremos a notação utilizada. Assumiremos que $G = (V, E, w)$ é um grafo simples, conexo, não direcionado e com uma função $w : E \to \mathbb{R}^+$. Seja $|V| = n$ e $\mathcal{C} = \{C_1, ..., C_k\}$ uma partição (também chamaremos de particionamento/clusterização) de V, onde cada $C_i \subseteq V$ corresponde a um cluster. Denotaremos por $w(\mathcal{C})$ a soma do peso de todas as arestas intracluster e $\overline{w}(\mathcal{C})$ o peso de todas as arestas intercluster, ou seja,

$$w(\mathcal{C}) = \frac{1}{2} \sum_{i=1}^{k} \sum_{u,v \in C_i} w(u, v),$$

$$\overline{w}(\mathcal{C}) = \frac{1}{2} \sum_{i=1}^{k} \sum_{u \in C_i, v \in V \setminus C_i} w(u, v).$$

Para um subconjunto de arestas $E' \subseteq E$, definimos $w(E') = \sum_{e \in E'} w(e)$. O subgrafo induzido pelo subconjunto C_i será denotado por $G[C_i]$. Sejam X e Y subconjuntos de V. Denotamos por $E(X, Y)$ o conjunto de arestas entre vértices de X e Y. O somatório dos pesos das arestas de $E(X, Y)$ será denotado por $c(X, Y)$. Quando $X = Y$, então também denotamos $E(X, Y)$ por $E(X)$.

A seguir, serão descritas as funções de Condutância, Corte Normalizado, Cobertura e Desempenho.

Condutância - φ: A condutância de um cluster C_i é definida como:

$$\varphi(C_i) = \begin{cases} 1, & \text{se } C_i \in \{\emptyset, V\} \\ 0, & \text{se } C_i \notin \{\emptyset, V\} \text{ e } E(C_i, V\backslash C_i) = 0 \\ \frac{c(C_i, V\backslash C_i)}{\min\{a(C_i), a(V\backslash C_i)\}}, & \text{caso contrário} \end{cases}$$

onde

$$a(C_i) = \sum_{u \in C_i} \sum_{v \in V} w(u,v) = 2 \sum_{e \in E(C_i)} w(e) + \sum_{e \in c(C_i, V\backslash C_i)} w(e)$$

A condutância de um grafo G, denotada por $\varphi(G)$, é o valor da condutância mínima dentre todos os possíveis cortes de G, isto é,

$$\varphi(G) = \min_{C_i \subset V} \varphi(C_i).$$

O problema de encontrar o corte de condutância mínimo de um grafo G é NP-Difícil [5], além disso, existem algoritmos aproximados [34, 41] (um desses algoritmos será discutido mais profundamente neste trabalho).

Note que, um corte de condutância em um grafo será mínimo quando os subconjuntos de vértices gerado pelo corte tenha a soma dos pesos das arestas internas alto e, ao mesmo tempo, o somatório das arestas do corte tenha o valor baixo. Veja na Figura 2.1 uma ilustração de um corte de condutância com subconjuntos de vértices balanceados em relação a soma das arestas internas.

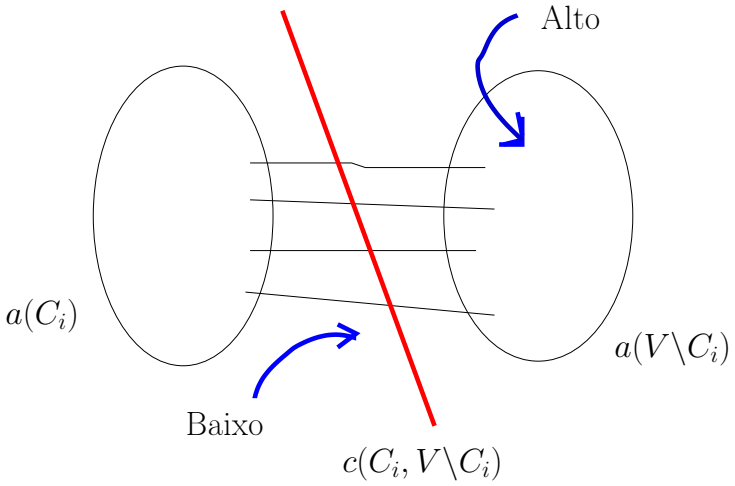


Figura 2.1: Representação de um corte de condutância. Cada circunferência representa um conjunto de vértices separados pelo corte $c(C_i, V\backslash C_i)$.

Para um particionamento $\mathcal{C} = \{C_1, \ldots C_k\}$ de um grafo G, chamamos de condutância intracluster, $\alpha(\mathcal{C})$, o valor de condutância mínimo sobre todos os subgrafos induzidos

$G[C_i]$, enquanto a condutância intercluster, $\delta(\mathcal{C})$, depende do valor de condutância máximo sobre todos os cortes clusteres C_i. Essas diferentes noções de condutância são definidas formalmente a seguir:

$$\alpha(\mathcal{C}) = \min_{i \in \{1,\ldots,k\}} \varphi(G[C_i]),$$

$$\delta(\mathcal{C}) = \begin{cases} 1, & \text{se } \mathcal{C} = \{V\}, \\ 1 - \max\limits_{i \in \{1,\ldots,k\}} \varphi(C_i) & \text{caso contrário.} \end{cases}$$

Podemos avaliar o particionamento $\mathcal{C}$ da mesma forma como foi expressa pela função $index(\mathcal{C})$ na equação (1.1). Para o papel das funções f e g utilizamos respectivamente as funções α e δ. Para a função N, utilizamos o valor constante 1, que também é o valor máximo de $\alpha + \delta$.

Note que um particionamento $\mathcal{C}$ com condutância intracluster muito baixa significa que pelo menos um cluster (um subconjunto C_i de $\mathcal{C}$) é esparso, o que não é uma "boa característica" para um cluster. Por outro lado, um particionamento com condutância intercluster baixa, contém pelo menos dois clusteres com vértices fortemente relacionados entre si, o que também não é uma "boa característica" para o particionamento.

Corte Normalizado - $NCut$**:** Seja um particionamento $\mathcal{C} = \{C_1, \ldots, C_k\}$ com $k > 1$. O custo do corte normalizado de $\mathcal{C}$ é definido como

$$NCut(\mathcal{C}) = \sum_{i=1}^{k} \frac{c(C_i, V \backslash C_i)}{a(C_i)}.$$

Note que, quanto menor for o custo do corte normalizado, melhor será o particionamento.

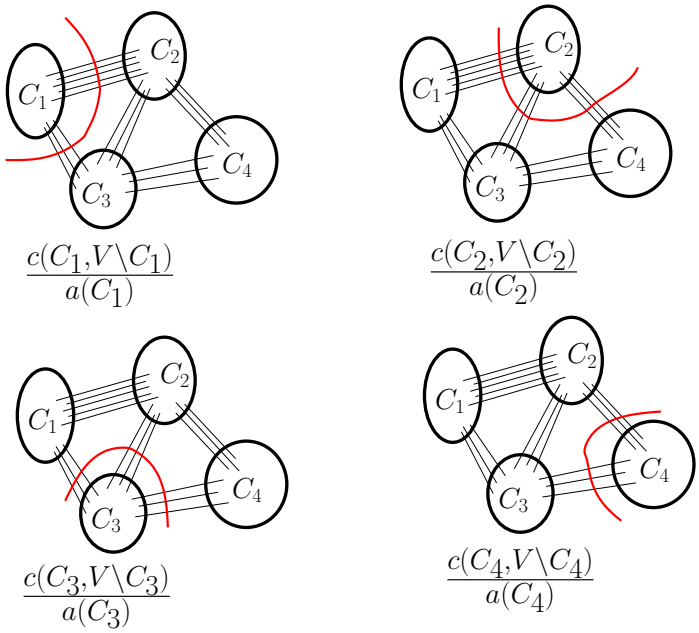
A Figura 2.2 ilustra os sucessivos cortes feitos em um grafo para calcular o custo do corte normalizado no particionamento $\mathcal{C} = \{C_1, C_2, C_3, C_4\}$.

Cobertura - *coverage***:** O valor de cobertura, dado pela função $coverage(\mathcal{C})$, de um particionamento $\mathcal{C}$ com pelo menos três clusteres é a fração entre o peso das arestas intracluster e o peso total de todas as arestas. A função é definida como

$$coverage(\mathcal{C}) = \frac{w(\mathcal{C})}{w(E)} = \frac{w(\mathcal{C})}{w(\mathcal{C}) + \overline{w}(\mathcal{C})}.$$

Quanto maior for a cobertura melhor será a qualidade do particionamento.

Na função de *coverage*, restringimos o número k de clusteres para pelo menos três. Note que, se $k = 2$, o valor de *coverage* será máximo para o particionamento obtido pelo corte mínimo do grafo (obtendo o menor valor para $w(\mathcal{C})$). Em geral, o corte mínimo do grafo não é considerado um "bom" particionamento.



$$NCut(\mathcal{C}) = \frac{c(C_1,V\setminus C_1)}{a(C_1)} + \frac{c(C_2,V\setminus C_2)}{a(C_2)} + \frac{c(C_3,V\setminus C_3)}{a(C_3)} + \frac{c(C_4,V\setminus C_4)}{a(C_4)}$$

Figura 2.2: Sucessivos cortes para o calcular o valor da função $NCut$

Desempenho - *Performance*: A *Performance* de um particionamento $\mathcal{C}$ representa o número de "pares de vértices corretamente interpretados" em um grafo [12]. Informalmente, a *Performance* é uma soma normalizada de contribuições de cada par de vértices. Se os dois vértices estiverem no mesmo cluster, então o par contribui com o peso da aresta entre eles, se houver. Do contrário, o par contribui com um valor fixo, descontado o peso da aresta correspondente, quando existir. Seja M o peso máximo que uma aresta pode ter. Definimos funções f e g como:

$$f(\mathcal{C}) = \sum_{i=1}^{k} w(E(C_i))$$

$$g(\mathcal{C}) = \sum_{i=1}^{k} \sum_{j>i} M|\{(u,v) \notin E | u \in C_i, v \in C_j\}|$$

A função f contará a contribuição dos pares de vértices intracluster, enquanto a função g contará a contribuição M de cada par de vértices não adjacentes em diferentes clusteres. Para descontar o custo das arestas intercluster, é feita a diferença $(M|\overline{E(\mathcal{C})}| - w(\overline{E(\mathcal{C})}))$, onde $\overline{E(\mathcal{C})}$ é o conjunto de arestas intercluster. A fórmula completa da *Performance* é

dada por

$$Performance(\mathcal{C}) = \frac{f(\mathcal{C}) + g(\mathcal{C}) + (M|\overline{E(\mathcal{C})}| - w(\overline{E(\mathcal{C})}))}{\frac{1}{2}n(n-1)M}.$$

Quanto maior for o valor da *Performance*, melhor será o particionamento.

Capítulo 3

Descrição de algoritmos escolhidos

Neste capítulo, apresentamos alguns algoritmos selecionados após a revisão bibliográfica, para um estudo mais profundo, incluindo implementação e comparação prática dos mesmos. Escolhemos algoritmos que não possuem alta complexidade e apresentam boas perspectivas para aplicações práticas. A maioria dos algoritmos selecionados não possuem garantia de aproximação, mas são úteis, pois utilizam heurísticas que podem levar a bons resultados em aplicações específicas.

3.1 *Spectral Clustering*

O método espectral de clusterização é uma técnica baseada em grafos. Ela utiliza os autovalores e os autovetores da matriz de Laplace do grafo para encontrar os clusteres. A matriz utilizada é formada a partir do grafo de entrada. Nas próximas seções são definidas e apresentadas algumas propriedades básicas sobre tais matrizes.

Os algoritmos que utilizam métodos de *clusterização espectral* estão se tornando bastante populares [42]. Isso ocorre devido a simplicidade de implementação, já que existem vários softwares que fornecem funções básicas de álgebra linear. Outra vantagem é o desempenho superior aos tradicionais algoritmos de clusterização, como por exemplo, os algoritmos para o problema *k-means*.

3.1.1 Autovetores e Autovalores

Nesta seção é feita uma pequena revisão de vários conceitos envolvendo álgebra linear. Especificamente, são definidos autovetores e autovalores e são apresentados teoremas importantes, que serão utilizados em alguns dos algoritmos que apresentaremos posteriormente.

Definição 1 (Autovetores e Autovalores) *Se A é uma matriz $n \times n$, então um vetor não-nulo $x \in \mathbb{R}^n$ é chamado um* autovetor *de A se Ax é um múltiplo escalar de x, ou seja,*

$$Ax = \lambda x$$

para algum escalar $\lambda \in \mathbb{R}$. O escalar λ é chamado um autovalor *de A e dizemos que x é um autovetor associado a λ.*

Definição 2 (Matiz Simétrica) *Uma matriz A é simétrica se for igual a própria transposta, ou seja, $A' = A$.*

Definição 3 (Conjunto Linearmente Independente/Dependente) *Se $X = \{x_1, x_2, \ldots, x_r\}$ é um conjunto não-vazio de vetores em $\mathbb{R}^n$ e se a equação vetorial*

$$k_1x_1 + k_2x_2 + \ldots + k_rx_r = 0$$

tem apenas a solução trivial $k_i = 0$ para todo i, $1 \leq i \leq r$, então o conjunto X é chamado linearmente independente. *Se existirem outras soluções, então X é um conjunto* linearmente dependente *[32].*

Teorema 1 *Se $x_1, x_2, \ldots, x_k$ são autovetores de uma matriz A associados aos autovalores distintos $\lambda_1, \lambda_2, \ldots, \lambda_k$, então $X = \{x_1, x_2, \ldots, x_k\}$ é um conjunto linearmente independente.*

Prova: Vamos supor que o conjunto X é formado por vetores linearmente dependentes.

Por definição, o autovetor x_1 é não-nulo, logo o conjunto formado por apenas um vetor $\{x_1\}$ é linearmente independente. Seja r o maior inteiro tal que $\{x_1, x_2, \ldots, x_r\}$ seja linearmente independente. Como foi suposto que X é linearmente dependente, obtemos $1 \leq r < k$. Teremos que o conjunto $\{x_1, \ldots, x_r, x_{r+1}\}$ é, pela maximalidade de r, linearmente dependente. Assim, existem escalares $c_1, c_2, \ldots, c_{r+1}$, não-nulos, que satisfazem

$$c_1x_1 + c_2x_2 + \ldots + c_{r+1}x_{r+1} = 0 \tag{3.1}$$

Multiplicando ambos os lados de (3.1) por A e sabendo que $Ax_i = \lambda_i x_i$, para $1 \leq i \leq r+1$, obtemos

$$c_1\lambda_1x_1 + c_2\lambda_2x_2 + \ldots + c_{r+1}\lambda_{r+1}x_{r+1} = 0 \tag{3.2}$$

Multiplicando ambos os lados de (3.1) por λ_{r+1} e subtraindo a equação resultante de (3.2), teremos

$$c_1(\lambda_1 - \lambda_{r+1})x_1 + c_2(\lambda_2 - \lambda_{r+1})x_2 + \ldots + c_r(\lambda_r - \lambda_{r+1})x_r = 0 \tag{3.3}$$

Como os vetores $\{x_1, \ldots, x_r\}$ formam um conjunto linearmente independente, a equação (3.3) implica

$$c_1(\lambda_1 - \lambda_{r+1}) = c_2(\lambda_2 - \lambda_{r+1}) = \ldots = c_r(\lambda_r - \lambda_{r+1}) = 0$$

e como $\lambda_1, \lambda_2, \ldots, \lambda_r, \lambda_{r+1}$ são distintos, seque que

$$c_1 = c_2 = \ldots = c_r = 0.$$

Substituindo esses valores em (3.1) teremos que

$$c_{r+1}x_{r+1} = 0.$$

Como o autovetor x_{r+1} é não-nulo, isso implica que $c_{r+1} = 0$.

Logo, todos os escalares $c_1, c_2, \ldots, c_{r+1}$ são nulos, o que contraria a suposição inicialmente adotada. Portanto, chegamos a um absurdo, o que completa a prova. ■

Definição 4 (Produto Interno) *Sejam u e v dois vetores, o produto interno denotado por $u' \cdot v$, onde u' é o vetor u na forma transposta, é definido como*

$$u' \cdot v = u_1v_1 + u_2v_2 + \ldots + u_nv_n.$$

Definição 5 (Conjunto Ortogonal) *Um conjunto $\{v_1, v_2, ..., v_p\}$ de vetores em $\mathbb{R}^n$ é dito um* conjunto ortogonal *se cada par de vetores distintos no conjunto é ortogonal, isto é, se $v_i' \cdot v_j = 0$ sempre que $i \neq j$.*

Definição 6 (Conjunto Ortonormal) *Diz-se que o conjunto $S = \{v_1, v_2, ..., v_n\}$ de vetores em $\mathbb{R}^n$ é* conjunto ortonormal *se, para todo $1 \leq i, j \leq n$,*

$$(v_i' \cdot v_j) = \begin{cases} 0, & \text{se } i \neq j, \\ 1, & \text{se } i = j. \end{cases}$$

Isto é, S é um conjunto ortogonal com vetores unitários (normalizados).

Teorema 2 *Se $S = \{v_1, v_2, \ldots, v_p\}$ é um conjunto ortogonal de vetores não nulos em $\mathbb{R}^n$, então S é linearmente independente e, portanto, é uma base para o subespaço gerado por S.*

Prova: Escrevendo 0 como combinação linear dos vetores de S, utilizando os escalares $c_1, c_2, \ldots, c_p$, teremos a seguinte igualdade:

$$0 = c_1v_1 + c_2v_2 + \ldots + c_pv_p.$$

Multiplicando essa igualdade por um vetor v_i, para todo $v_i \in S$, teremos

$$0 \cdot v_i = (c_1 v_1 + c_2 v_2 + \ldots + c_p v_p)' \cdot v_i = 0.$$

Assim,

$$c_1(v_1' \cdot v_i) + c_2(v_2' \cdot v_i) + \ldots + c_i(v_i' \cdot v_i) + \ldots + c_p(v_p' \cdot v_i) = 0.$$

Como S é ortogonal, temos $v_j' \cdot v_i = 0$, quando $i \neq j$. Logo, $c_i(v_i' \cdot v_i) = 0$ e, por v_i ser não-nulo, $c_i = 0$ para cada um dos escalares $c_1, c_2, \ldots, c_p$. Ou seja, S é linearmente independente. ■

Teorema 3 *Os autovetores de uma matriz simétrica A formam uma base ortogonal.*

Prova: Sejam λ_i e λ_j autovalores e x_i e x_j os seus respectivos autovetores associados. Utilizando a definição 1, podemos fazer o seguinte:

$$\begin{array}{ll} Ax_i = \lambda_i x_i & \text{(Multiplicando por } x_j'\text{)} \\ x_j' \cdot (Ax_i) = x_j' \cdot (\lambda_i x_i). & \end{array}$$

Fazendo um procedimento semelhante, multiplicando a equação $Ax_j = \lambda_j x_j$ pelo autovetor x_i' e subtraindo os dois resultados obteremos, pela simetria da matriz A, que

$$(\lambda_i - \lambda_j) x_i' \cdot x_j = 0,$$

no qual segue que $x_i' \cdot x_j = 0$, o que finaliza a prova. ■

Teorema 4 *Se* $S = \{v_1, \ldots, v_n\}$ *é uma* base ortonormal *de um espaço* $\mathcal{V}$ *e* u *é um vetor qualquer de* $\mathcal{V}$*, então*

$$u = (u' \cdot v_1)v_1 + (u' \cdot v_2)v_2 + \ldots + (u' \cdot v_n)v_n.$$

Prova: Como $S = \{v_1, v_2, \ldots, v_n\}$ é uma base, um vetor u pode ser escrito da seguinte forma.

$$u = k_1 v_1 + k_2 v_2 + ... + k_n v_n.$$

Agora, para completar a prova, basta mostrar que para $i = 1, 2, \ldots, n$ teremos $k_i = (u' \cdot v_i)$. Para cada vetor v_i de S nós temos

$$(u' \cdot v_i) = ((k_1 v_1 + k_2 v_2 + ... + k_n v_n)' \cdot v_i)$$

$$= k_1(v_1' \cdot v_i) + k_2(v_2' \cdot v_i) + ... + k_n(v_n' \cdot v_i).$$

Como S é um conjunto ortonormal, temos $(v_i' \cdot v_i) = ||v_i||^2 = 1$ e $(v_j' \cdot v_i) = 0$ se $i \neq j$. Portanto, temos que $(u' \cdot v_i) = k_i$, o que completa a prova. ■

Os escalares

$$(u' \cdot v_1), (u' \cdot v_2), \ldots, (u' \cdot v_n)$$

utilizados na prova do Teorema 4 são as coordenadas do vetor $u = k_1v_1 + k_2v_2 + \ldots + k_nv_n$ em relação à base ortonormal $S = \{v_1, v_2, \ldots, v_n\}$ e

$$(u)_S = ((u' \cdot v_1), (u' \cdot v_2), \ldots, (u' \cdot v_n))$$

é o vetor de coordenadas de u em relação a esta base.

Definição 7 (Matriz Diagonalizável) *Uma matriz quadrada A é dita* diagonalizável *se existir uma matriz inversível P tal que $P^{-1}AP$ é uma matriz diagonal; dizemos, então, que a matriz P* diagonaliza *A.*

Proposição 1 *Se A é uma matriz $n \times n$, então são equivalentes as seguintes afirmações:*

(a) *A é diagonalizável;*

(b) *A tem n autovetores linearmente independentes.*

Prova: Supondo que A é diagonalizável, então existe uma matriz P inversível tal que $P^{-1}AP$ é diagonal, ou seja, $P^{-1}AP = D$, onde $D = diag(\lambda_1, \lambda_2, \ldots, \lambda_n)$.

Segue da fórmula $P^{-1}AP = D$ que $AP = PD$. Se denotarmos os vetores colunas de P por $p_1, p_2, \ldots, p_n$ então as colunas de PD serão $\lambda_1p_1, \ldots, \lambda_np_n$. No entanto, as sucessivas colunas de AP são $Ap_1, Ap_2, \ldots, Ap_n$. Assim, devemos ter

$$Ap_1 = \lambda_1p_1, Ap_2 = \lambda_2p_2, \ldots, Ap_n = \lambda_np_n$$

Como P é inversível, seus vetores colunas são todos não-nulos; assim, por 1 segue que $\lambda_1, \lambda_2, \ldots, \lambda_n$ são autovalores de A e que $p_1, \ldots, p_n$ são os autovetores associados. Como P é inversível, então $p_1, p_2, \ldots, p_n$ são linearmente independentes.

Agora, vamos supor que (b) é verdadeiro e provar (a). Assim, suponha que A tem n autovetores linearmente independentes $p_1, \ldots, p_n$ com autovalores associados $\lambda_1, \ldots, \lambda_n$. Seja a matriz P com vetores colunas $p_1, \ldots, p_n$. Os vetores colunas do produto AP são

$$Ap_1, \ldots, Ap_n.$$

Mas

$$Ap_1 = \lambda_1p_1, \ldots, Ap_n = \lambda_np_n.$$

Com isso, teremos que $AP = PD$, onde $D = diag(\lambda_1, \ldots, \lambda_n)$. Como os vetores colunas de P são linearmente independentes, P é inversível; assim, podemos escrever $D = P^{-1}AP$, ou seja, A é diagonalizável. ■

Definição 8 (Matriz Ortogonalmente Diagonalizável) *Uma matriz A de tamanho $n \times n$ é dita Ortogonalmente Diagonalizável se existir uma matriz ortogonal P tal que $P^{-1}AP = P'AP$ é uma matriz diagonal. Uma matriz ortogonal P é uma matriz cuja inversa coincide com a sua transposta, isto é: $P^{-1} = P'$.*

Teorema 5 *Se A é uma matriz $n \times n$, então as seguintes afirmações são equivalentes.*

(a) *A é simétrica;*

(b) *A tem um conjunto ortonormal de n autovetores;*

(c) *A é ortogonalmente diagonalizável.*

Prova: A prova desse teorema pode ser encontrada em [32], página 251, Teorema 7.3.1.

Os n autovetores $\{x_1, \ldots, x_n\}$ que formam uma base ortonormal correspondem às colunas da matriz P, que diagonaliza A ortogonalmente. Isso pode ser verificado calculando a equação $P^{-1}AP = D$, onde D é uma matriz diagonal. O resultado em cada posição i da diagonal de D terá o valor $x_i'Ax_i$ (lembrando $x_i' \cdot x_j = 0$ para $i \neq j$).

Teorema 6 *Seja A uma matriz $n \times n$ simétrica cujos autovalores em ordem crescente de valor são $\lambda_1 \leq \lambda_2 \leq \ldots \leq \lambda_n$. Seja $x \in \mathbb{R}^n$ um vetor qualquer restrito a $||x|| = 1$, então:*

(a) $\lambda_1 \leq x'Ax \leq \lambda_n$

(b) *se x é o autovetor de A associado a λ_1 então $x'Ax = \lambda_1$ e se x é o autovetor de A associado a λ_n então $x'Ax = \lambda_n$.*

Prova: Como A é simétrico, segue do Teorema 5 que existe uma base ortonormal consistindo de n autovetores de A. Seja $S = \{v_1, v_2, \ldots, v_n\}$ tal base, onde v_i é o autovetor associado ao autovalor λ_i. Como S é uma base ortonormal, cada vetor v_i está restrito a $||v_i|| = 1$.

Segue pelo Teorema 4 que, para qualquer $x \in \mathbb{R}^n$ temos:

$$x = (x' \cdot v_1)v_1 + (x' \cdot v_2)v_2 + \ldots + (v' \cdot v_n)v_n.$$

Assim,

$$\begin{aligned} Ax &= (x' \cdot v_1)Av_1 + (x' \cdot v_2)Av_2 + \ldots + (x' \cdot v_n)Av_n \\ &= (x' \cdot v_1)\lambda_1 v_1 + (x' \cdot v_2)\lambda_2 v_2 + \ldots + (x' \cdot v_n)\lambda_n v_n \\ &= \lambda_1(x' \cdot v_1)v_1 + \lambda_2(x' \cdot v_2)v_2 + \ldots + \lambda_n(x' \cdot v_n)v_n. \end{aligned}$$

Segue que os vetores de coordenadas de x e de Ax em relação à base S são

$$(x)_S = (x' \cdot v_1, x' \cdot v_2, \ldots, x' \cdot v_n)$$

$$(Ax)_S = (\lambda_1(x' \cdot v_1), \lambda_2(x' \cdot v_2), \ldots, \lambda_n(x' \cdot v_n))$$

Usando essas duas equações acima e lembrando que $||x|| = 1$, obteremos

$$||x||^2 = (x' \cdot v_1)^2 + (x' \cdot v_2)^2 + \ldots + (x' \cdot v_n)^2 = 1$$

$$(x' \cdot Ax) = \lambda_1(x' \cdot v_1)^2 + \lambda_2(x' \cdot v_2)^2 + \ldots + \lambda_n(x' \cdot v_n)^2.$$

Usando essas duas equações, podemos provar que $x'Ax \leq \lambda_n$ como segue:

$$\begin{aligned} x'Ax = (x' \cdot Ax) &= \lambda_1(x' \cdot v_1)^2 + \lambda_2(x' \cdot v_2)^2 + \ldots + \lambda_n(x' \cdot v_n)^2 \\ &\leq \lambda_n(x' \cdot v_1)^2 + \lambda_n(x' \cdot v_2)^2 + \ldots + \lambda_n(x' \cdot v_n)^2 \\ &= \lambda_n((x' \cdot v_1)^2 + (x' \cdot v_2)^2 + \ldots + (x' \cdot v_n)^2) \\ &= \lambda_n. \end{aligned}$$

Para provar que $x'Ax \geq \lambda_1$, realizamos o mesmo procedimento. Para isso faremos o seguinte cálculo

$$\begin{aligned} x'Ax = (x' \cdot Ax) &= \lambda_1(x' \cdot v_1)^2 + \lambda_2(x' \cdot v_2)^2 + \ldots + \lambda_n(x' \cdot v_n)^2 \\ &\geq \lambda_1(x' \cdot v_1)^2 + \lambda_1(x' \cdot v_2)^2 + \ldots + \lambda_1(x' \cdot v_n)^2 \\ &= \lambda_1((x' \cdot v_1)^2 + (x' \cdot v_2)^2 + \ldots + (x' \cdot v_n)^2) \\ &= \lambda_1. \end{aligned}$$

Continuemos a prova agora para o item **(b)**. Sabemos que a equação

$$Av_i = \lambda_i v_i \tag{3.4}$$

tem como solução apenas os autovetores v_i, para $1 \leq i \leq n$, de S. Agora, multiplicando a equação (3.4) por v_i em ambos os lados da igualdade, teremos o seguinte:

$$v_i' \cdot (Av_i) = \lambda_i(v_i' \cdot v_i).$$

Note que a restrição $||v_i|| = 1$ implica que $v_i' \cdot v_i = 1$ para todo vetor v_i em S, logo, teremos que $v_i'Av_i = \lambda_i$.

Considerando x o autovetor v_1 de A associado a λ_1 e $||x|| = 1$, teremos

$$x'Ax = x' \cdot Ax = \lambda_1(x' \cdot x) = \lambda_1||x||^2 = \lambda_1.$$

Analogamente, considerando x o autovetor v_n de A associado a λ_n teremos que $x'Ax = \lambda_n$.
■

3.1.2 Matrizes e Grafos

Nesta seção são dadas definições das diferentes matrizes utilizadas para representação de grafos. Além de suas definições, também são vistas várias propriedades das matrizes úteis para o entendimento dos métodos espectrais de clusterização.

Definição 9 (Corte) *Seja um grafo $G = (V, E, w)$, onde w é uma função $w : E \to \mathbb{R}^+$ que atribui peso a cada aresta $e \in E$. Dado $S \subseteq V$, definimos um corte induzido por S como o conjunto de arestas que conectam vértices de S com vértices de $V \backslash S$, ou seja*

$$E(S, V\backslash S) = \{(u, v) \in E | u \in S, v \in V\backslash S\}.$$

O custo do corte será o somatório do peso das arestas do corte. Denotaremos o custo do corte como $c(S, V\backslash S)$.

A matriz para representação de grafo mais conhecida é a *matriz de adjacência*. Dado um grafo $G = (V, E, w)$ com n vértices, não direcionado e w uma função $w : E \to \mathbb{R}^+$ de atribuição de peso a cada aresta $e \in E$, chamaremos de $A(G)$ a *matriz de adjacência* de G. Cada entrada $A_{(u,v)}$ da matriz de adjacência de G será da seguinte forma:

$$A_{(u,v)} = \begin{cases} w(u, v) & \text{se } e = (u, v) \in E, \\ 0 & \text{caso contrário.} \end{cases}$$

Dado o grafo G, denotamos por $D(G)$ a matriz diagonal de G em que cada entrada $D_{(u,v)}$ da matriz D terá o seguinte valor

$$D_{(u,v)} = \begin{cases} 0 & \text{se } u \neq v, \\ d(u) & \text{se } u = v, \end{cases}$$

onde $d(u) = \sum_{(u,v)\in E} w(u, v)$.

Definição 10 *O* espectro *de um grafo G é o conjunto de autovalores da matriz $M(G)$ de algum tipo M (veremos à frente definições de vários tipos de matrizes), junto aos seus respectivos autovetores associados [9].*

A principal ferramenta no estudo e aplicação de métodos espectrais para clusterização em grafos é a matriz de Laplace [42]. Existe um campo dedicado ao estudo dessas matrizes e suas propriedades dentro da teoria dos grafos, chamado *Spectral Graph Theory* (mais detalhes sobre *Spectral Graph Theory* veja em [17]). Abaixo, é dada a definição de matriz de Laplace e, em seguida, algumas propriedades necessárias para o estudo de clusterização espectral.

Definição 11 (Matriz de Laplace) *Seja $G = (V, E, w)$ um grafo não orientado com uma função $w(e)$ que atribui peso a cada aresta $e \in E$. A matriz de Laplace do grafo G é definida como:*

$$\mathcal{L}(G) = D(G) - A(G).$$

Pela definição, nota-se que cada posição (u, v) da matriz de Laplace, que corresponde a linha u e coluna v, tem a seguinte forma:

$$\mathcal{L}(u, v) = \begin{cases} d(v) & \text{se } u = v \\ -w(u, v) & \text{se } u \neq v \end{cases}$$

Proposição 2 *A matriz de Laplace $\mathcal{L}$ contém um conjunto ortonormal de n autovetores.*

Prova: Sendo a matriz de Laplace simétrica, a prova segue diretamente do Teorema 5.

Proposição 3 *Seja um grafo $G = (V, E)$, $n = |V|$ e $\mathcal{L} = \mathcal{L}(G)$ a matriz de Laplace de G. Então $\mathcal{L}$ é simétrica e positiva semidefinida, isto é,*

$$\forall x \in \mathbb{R}^n : x'\mathcal{L}x \geq 0,$$

precisamente temos:

$$x'\mathcal{L}x = \sum_{(u,v)\in E} w_{uv}(x_u - x_v)^2, \tag{3.5}$$

onde x_v denota a coordenada de x indexada pelo vértice v na matriz $\mathcal{L}$.

Prova: Lembre-se que d é função que retorna o somatório dos pesos de todas as arestas incidentes no vértice com índice i $(1 \leq i \leq n)$, definida formalmente como

$$d(i) = \sum_{j=1}^{n} w_{ij}.$$

Usando a definição de matriz de Laplace (definição 11), temos

$$x'\mathcal{L}x = x'Dx - x'Ax = \sum_{i=1}^{n} d(i)x_i^2 - \sum_{j=1}^{n}\sum_{i=1}^{n} x_i x_j w_{ij}$$

$$= \frac{1}{2}\left(\sum_{i=1}^{n} d(i)x_i^2 - 2\sum_{j=1}^{n}\sum_{i=1}^{n} x_i x_j w_{ij} + \sum_{j=1}^{n} d(j)x_j^2\right)$$

$$= \frac{1}{2}\left(\sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij}x_i^2 - 2\sum_{j=1}^{n}\sum_{i=1}^{n} x_i x_j w_{ij} + \sum_{j=1}^{n}\sum_{i=1}^{n} w_{ij}x_j^2\right)$$

$$= \frac{1}{2}\left(\sum_{i=1}^{n}\sum_{j=1}^{n} \left(w_{ij}x_i^2 - 2x_i x_j w_{ij} + w_{ij}x_j^2\right)\right)$$

$$= \frac{1}{2}\left(\sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij}\left(x_i^2 - 2x_i x_j + x_j^2\right)\right) = \frac{1}{2}\sum_{j=1}^{n}\sum_{i=1}^{n} w_{ij}(x_i - x_j)^2 = \sum_{(u,v)\in E} w_{uv}(x_u - x_v)^2.$$

A simetria de $\mathcal{L}$ segue diretamente pela simetria de D e pela simetria da matriz de adjacência A. Uma matriz A qualquer é positiva semi-definida se $x'Ax \geq 0$, assim, como o peso das arestas w_{ij} são valores não negativos, teremos que $\mathcal{L}$ é positiva semidefinida. ■

Definição 12 (Representação de Grafo no Espaço $\mathbb{R}^m$) *Definimos uma representação ρ de um grafo com pesos nas arestas $G = (V, E)$ no espaço $\mathbb{R}^m$, $m \leq n$ ($n = |V|$), como um mapeamento de V em $\mathbb{R}^m$ [27].*

Informalmente, podemos pensar que uma representação é a atribuição de cada vértice a algum ponto no espaço m-dimensional. A representação de um vértice u no espaço corresponderá a um vetor linha $\rho(u)$. Assim, a representação ρ pode ser dada como uma matriz R com tamanho $n \times m$ (cada linha dessa matriz corresponderá a um vértice do grafo G).

Algumas considerações sobre a representação devem ser tomadas. A primeira é que o "centro de gravidade" deve estar na origem. Isso é chamado de representação balanceada [27]. A representação dada pela matriz R é balanceada se $\overline{1}' \cdot R = 0$, onde $\overline{1}$ é um vetor unitário com valores iguais em todas suas coordenadas. Outra consideração é que uma representação será boa se a sua energia for mínima. O valor da energia de uma representação ρ é definido como

$$energia(\rho) = \sum_{(u,v)\in E} w_{uv}||\rho(u) - \rho(v)||^2. \tag{3.6}$$

Note que $||x||$ denota a distância Euclidiana de um vetor x. Assim, é natural pensar que, em uma boa representação, ou em uma representação com pouca energia, a distância entre os pontos que representam os vértices seja mínima. O mínimo possível para a fórmula da energia ocorre quando, para todos $u, v \in V$, vale $\rho(u) = \rho(v)$ ($energia(\rho) = 0$). Entretanto, como cada valor atribuído por ρ deve ser distinto, desconsideramos essa representação.

Proposição 4 *Seja ρ uma representação de um grafo G com peso nas arestas, e seja R a matriz de tamanho $n \times m$ ($m \leq n$) com colunas ortonormais. Se $\mathcal{L}$ é a matriz de Laplace do grafo G, então*

$$energia(\rho) = Tr(R'\mathcal{L}R),$$

onde a função Tr retorna a soma dos valores da diagonal principal.

Prova: Uma representação ρ é denotada na matriz R pelo espaço linha e também sabemos que os vetores colunas r_i, para $1 \leq i \leq m$, de R são ortonormais. Logo, cada valor na diagonal i da matriz $R'\mathcal{L}R$ será $r_i'\mathcal{L}r_i$. Lembrando que as linhas de R são os vetores $\rho(u)$, para todo $u \in V$. Portanto, pela Proposição 3, temos

$$r_i'\mathcal{L}r_i = \sum_{(u,v)\in E} w_{uv}(r_{ui} - r_{vi})^2,$$

onde r_{ui} corresponde a linha u da coluna i de R.

Fazendo a soma da diagonal da matriz $R'\mathcal{L}R$ teremos

$$\sum_{i=1}^{m} r_i\mathcal{L}r_i = \sum_{i=1}^{n} \sum_{(u,v)\in E} w_{uv}(r_{ui} - r_{vi})^2 = \sum_{(u,v)\in E} w_{uv}(\rho(u) - \rho(v))^2$$

que é justamente a equação que define energia (equação 3.6). ■

Prova: A prova pode ser encontrada em [9], página 203. ■

Proposição 5 *Seja G um grafo com n vértices e peso nas arestas e $\mathcal{L}$ sua correspondente matriz de Laplace. Assuma que os autovalores de $\mathcal{L}$ são $\lambda_1 \leq \ldots \leq \lambda_n$ e que $\lambda_2 > 0$. A energia mínima de uma representação de G em $\mathbb{R}^m$ é igual a $\sum_{i=2}^{m+1} \lambda_i$.*

Prova: A prova pode ser encontrada em [27], página 287. ■

Proposição 6 *Seja um grafo $G = (V, E, w)$ com n vértices. A função $\pi : \mathbb{R}^n \to \mathbb{R}$ definida como:*

$$\pi(x) = \frac{\sum_{(u,v)\in E} w_{uv}(x_u - x_v)^2}{\sum_{v\in V}(x_v)^2}$$

tem o vetor unitário como mínimo global com $\pi(\overline{1}) = 0$. Restringindo π ao domínio

$$D = \{x \in \mathbb{R}^n \backslash \{0\} : x' \cdot \overline{1} = 0, ||x|| = 1\},$$

obtemos que π tem em $x_2 \in D$ um mínimo global, onde x_2 é o autovetor da matriz de Laplace do grafo G associado ao menor autovalor não-nulo, λ_2 .

Prova: Pela Proposição 2, sabemos que a matriz de Laplace $\mathcal{L}$ contém n autovetores formando uma base ortonormal $S = \{v_1, \ldots, v_n\}$. Sabemos também, pela Proposição 3, que $\sum_{(u,t)\in E} w_{ut}(v_{ui} - v_{ti})^2 = v_i' \mathcal{L} v_i$. Além disso, podemos reescrever a equação que define um autovalor da matriz de Laplace, definida como $\mathcal{L}v_i = \lambda_i v_i$. Multiplicando por v_i' dos dois lados da igualdade, teremos $v_i' \mathcal{L} v_i = \lambda_i v_i' \cdot v_i$, assim, isolando λ_i teremos

$$\lambda_i = \frac{v_i' \mathcal{L} v_i}{v_i' \cdot v_i} = \frac{\sum_{(u,t)\in E} w_{ut}(v_{ui} - v_{ti})^2}{\sum_{u\in V} v_{ui}^2},$$

onde v_{ui} corresponde ao valor no índice u do autovetor v_i.

Como os autovetores formam uma base ortonormal (pelo Teorema 5), é fácil perceber que o vetor unitário normalizado é o autovetor associado ao autovalor 0 e, portanto, minimiza a função π. Agora vamos mostrar que, restringindo o domínio da função em D, temos v_2 como o mínimo de π.

Primeiramente, note que, para qualquer $x \in \mathbb{R}^n$, $\pi(x) = \pi(\hat{x})$, onde $\hat{x}$ é o vetor x normalizado. Assim, iremos nos restringir a vetores com norma 1. Pela Proposição 5, os autovetores $v_2, \ldots, v_{l+1}$ são uma representação de energia mínima em $\mathbb{R}^l$ quando restringimos vetores a D. Para $\mathbb{R}^l$, a representação de energia mínima consiste no autovetor v_2 e, portanto, v_2 minimiza π.

Proposição 7 *Os autovalores da matriz de Laplace são números reais.*

Prova: A prova pode ser encontrada em [32], Teorema 7.3.2, página 252. ■

Definição 13 (Matriz de Transição de Probabilidade) *Seja $G = (V, E, w)$ um grafo não orientado com uma função $w(e)$ que atribui peso a todas as arestas $e \in E$. Define-se matriz de transição de probabilidade $M(G)$ como*

$$M(G) = D(G)^{-1} \times A(G), \tag{3.7}$$

onde $A(G)$ é a matriz de adjacência do grafo G e $D(G)$ é a matriz diagonal.

A próxima proposição é importante pois relaciona a matriz de Laplace com a matriz de transição.

Proposição 8 *Um autovalor λ da matriz de transição M, obtido pela resolução do problema*

$$Mx = \lambda x,$$

corresponde a

$$\mathcal{L}x = (1 - \lambda)Dx,$$

que é, por sua vez, o problema generalizado dos autovalores/autovetores [44].

Prova: O problema generalizado dos autovalores/autovetores é definido como

$$Ax = \lambda Bx,$$

onde A e B são matrizes quaisquer e queremos encontrar o autovalor x e o autovetor λ. Note que a definição padrão do problema de autovetores, dada na definição 1, é um caso particular, em que B é matriz identidade.

Pela definição da matriz de transição (equação (3.7)) e de matriz de Laplace (definição 11), podemos fazer os seguintes cálculos

$$Mx = D^{-1}Ax = D^{-1}(D - \mathcal{L})x = x - D^{-1}\mathcal{L}x. \tag{3.8}$$

Sendo x um autovetor sabemos que $Mx = \lambda x$, podemos usar a equação (3.8) e obter

$$\begin{array}{l} D^{-1}\mathcal{L}x = (1-\lambda)x \quad \text{(multiplicando por D)} \\ \mathcal{L}x = (1-\lambda)Dx. \quad \blacksquare \end{array} \tag{3.9}$$

3.1.3 *Spectral Clustering Algorithms*

Esta seção tem como objetivo dar uma visão genérica dos algoritmos de clusterização espectral. Questões relativas ao porquê os autovetores dão bons resultados práticos para particionamento em grafos serão apresentados, de forma intuitiva, na próxima seção.

Algoritmos espectrais apresentam uma estrutura comum. Um algoritmo espectral genérico é apresentado a seguir (algoritmo 2).

Algoritmo 2: Algoritmo Genérico para Clusterização Espectral

Entrada: $G = (V, E, w)$

1 Crie a matriz de transição M a partir do grafo G em relação ao função de peso w ;
2 Calcule os maiores autovetores $\{x_1, ..., x_k\}$ de M associados aos maiores autovalores ;
3 Interprete os autovetores $\{x_1, ..., x_k\}$ gerando clusteres ;

No passo 1, alguns algoritmos usam a matriz de Laplace ao invés da matriz de transição. No passo 2, o número de autovetores calculados varia entre os vários algoritmos de particionamento espectral. Se a matriz utilizada for a de Laplace, serão escolhidos os menores autovalores e, caso seja utilizada a matriz de transição, serão escolhidos os maiores. Alguns algoritmos consideram suficiente calcular apenas um autovetor e, a partir desse autovetor, definir um corte que divide o grafo em dois componentes. Os clusteres são obtidos após várias chamadas recursivas para cada componente definido pelo corte. Um algoritmo que utiliza esse método para particionamento será discutido mais detalhadamente na seção 3.3 (veja também em [34] [50]). Outros algoritmos consideram mais

autovetores. Um exemplo desse tipo de algoritmo será tratado mais detalhadamente na seção 3.4.

Na clusterização espectral, todas as informações relevantes para determinar os clusteres são retiradas dos autovetores. Suponha que é conhecida a quantidade k de clusteres. Então os k autovetores associados aos k menores autovalores da matriz de Laplace possuem informações para determinar esses clusteres.

Mais precisamente, cada coordenada de um autovetor $x_i \in \mathbb{R}^n$, para $1 \leq i \leq k$, corresponderá a um vértice do grafo de entrada G. Dado um autovetor x_i e dois vértices u e v, se u e v têm alta similaridade, então os valores de x_{iu} e x_{iv} tendem a ser próximos. De forma semelhante, se u e v possuem baixa similaridade, então x_{iu} e x_{iv} tendem a ter valores distintos.

Vários algoritmos que utilizam técnica de clusterização espectral [48, 55] constroem uma matriz $X = [x_1 x_2 ... x_k] \in \mathbb{R}^{n \times k}$ onde cada autovetor é uma coluna da matriz. Cada vértice está associado a uma linha da matriz X. Note que, dessa forma, os k autovetores formam uma matriz de representação $(n \times k)$, conforme discutido na definição 12. Cada linha é interpretada como um ponto no espaço $\mathbb{R}^k$ e os clusteres são obtidos a partir da disposição destes pontos no espaço. Algoritmos como o *k-means* ou *k-median* (ou qualquer outro algoritmo para clusterização espacial) podem ser utilizados para determinarem os clusteres usando esta matriz X.

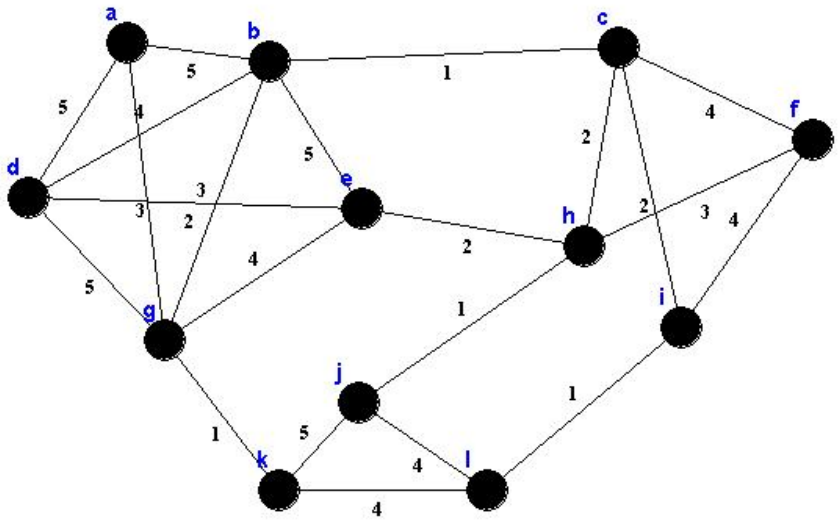


Figura 3.1: Grafo de Exemplo.

Tomemos um exemplo prático para ilustrar as informações que podem ser retiradas dos autovetores. Considere o grafo representado na Figura 3.1. Temos três clusteres:

$\{a, b, e, g, d\}$, $\{c, f, h, i\}$ e $\{i, j, k\}$. Veja na Figura 3.2 a matriz de transição desse grafo.

a	0.00	0.38	0.00	0.38	0.00	0.00	0.23	0.00	0.00	0.00	0.00	0.00
b	0.26	0.00	0.05	0.21	0.32	0.00	0.16	0.00	0.00	0.00	0.00	0.00
c	0.00	0.11	0.00	0.00	0.00	0.44	0.00	0.22	0.22	0.00	0.00	0.00
d	0.31	0.25	0.00	0.00	0.13	0.00	0.31	0.00	0.00	0.00	0.00	0.00
e	0.00	0.43	0.00	0.14	0.00	0.00	0.29	0.14	0.00	0.00	0.00	0.00
f	0.00	0.00	0.36	0.00	0.00	0.00	0.00	0.27	0.36	0.00	0.00	0.00
g	0.19	0.19	0.00	0.31	0.25	0.00	0.00	0.00	0.00	0.00	0.06	0.00
h	0.00	0.00	0.25	0.00	0.25	0.38	0.00	0.00	0.00	0.13	0.00	0.00
i	0.00	0.00	0.29	0.00	0.00	0.57	0.00	0.00	0.00	0.00	0.00	0.14
j	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.10	0.00	0.00	0.50	0.40
k	0.00	0.00	0.00	0.00	0.00	0.00	0.10	0.00	0.00	0.50	0.00	0.40
l	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.11	0.44	0.44	0.00

Figura 3.2: Matriz de transição de probabilidade do grafo da Figura 3.1. Cada vértice do grafo é uma linha da matriz.

Nas figuras 3.3 e 3.4, temos os gráficos que representam os autovetores da matriz de transição do grafo da Figura 3.1. Nestes gráficos, o eixo x corresponde aos vértices. Cada gráfico está associado a um autovetor onde o eixo y corresponde aos valores das coordenadas do autovetor referentes a cada vértice.

Alguns autovetores possuem mais informações para gerar os clusteres. Na Figura 3.3, por exemplo, podemos ver que, em cada cluster natural do grafo, seus vértices possuem valores próximos. Os vértices a, b, d, e, g no gráfico da esquerda possuem valores próximos nas suas respectivas posições do segundo autovetor. O mesmo acontece para o cluster c, f, h e o cluster j, k, l. O quarto e o quinto autovetores, correspondentes a Figura 3.4, já não determinam corretamente a distinção dos clusteres. No trabalho de [57], são propostos algoritmos para seleção de autovetores relevantes. É demonstrado que a seleção de autovetores leva a uma melhor acurácia no número de clusteres resultantes.

Na Seção 3.4, é apresentado um algoritmo que interpreta os autovetores de uma maneira diferente. Ao invés de utilizar os autovetores para determinar a disposição dos vértices no espaço, é construído um grafo, onde cada vértice será um ponto e o peso de uma aresta será a distância entre os pontos correspondentes. Os clusteres são encontrados utilizando uma técnica semelhante àquela proposta no trabalho de Zahn [60].

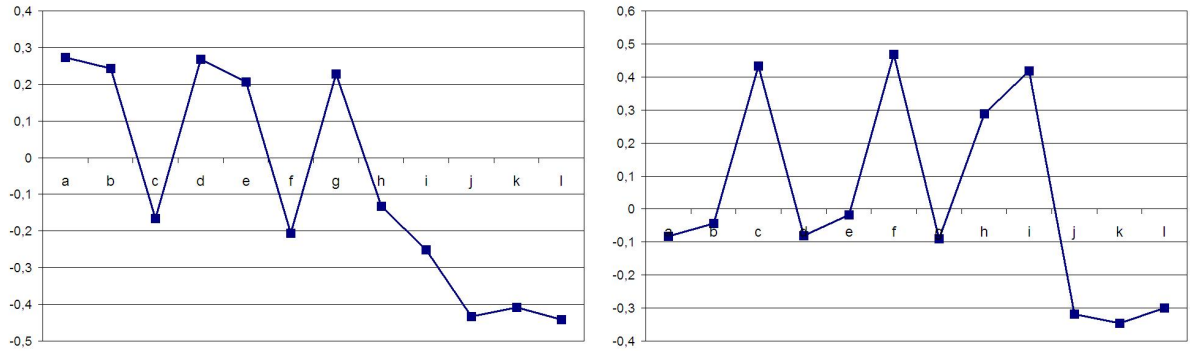


Figura 3.3: Segundo e terceiro autovetores.

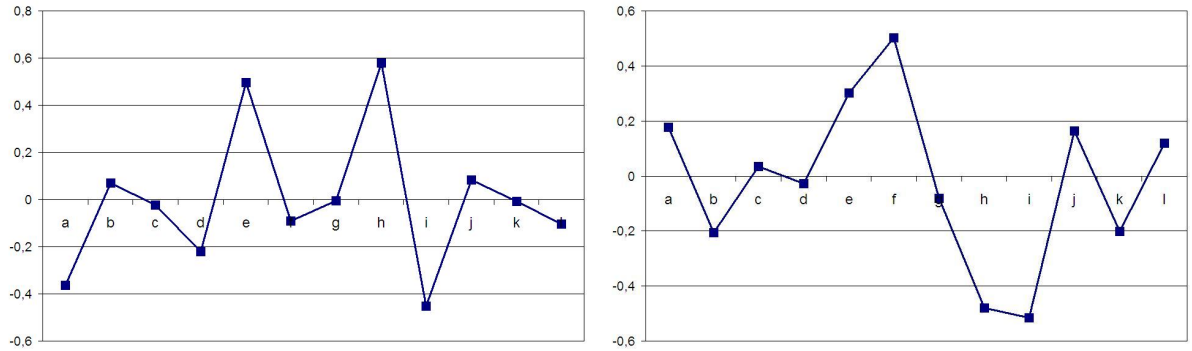


Figura 3.4: Quarto e quinto autovetores.

3.1.4 Por que *Spectral Clustering* funciona?

Intuitivamente, o problema de particionamento/clusterização de um grafo, como descrita na Seção 1.2, tem como objetivo encontrar uma partição do grafo no qual as arestas entre vértices de diferentes partições tenham peso baixo e arestas entre os vértices de uma mesma partição tenham peso alto. A maneira mais simples de construir uma partição/clusterização é resolvendo o problema do corte mínimo (ver definição 9). O problema dessa estratégia é que a solução do corte mínimo pode simplesmente consistir em separar um único vértice do resto do grafo, não constituindo, assim, um "bom cluster". Uma maneira de contornar esse problema é exigir que os subconjuntos de vértices gerados pelo corte sejam "razoavelmente" grandes e com tamanhos (quantidade de vértices ou soma dos pesos das arestas internas) semelhantes. Segundo Luxburg [42], as duas funções que codificam isso são *RatioCut* (primeiramente estudado por Hagen e Kahng [30]) e corte normalizado $Ncut$ (primeiramente estudado por Shi e Malik [50]). A função de condutância também tenta gerar cortes com subconjuntos de vértices balanceados. As funções de avaliação para a condutância e corte normalizado ($NCut$) são definidas

na Seção 2. Dada uma partição $\{C_1, \ldots, C_k\}$ dos vértices, a definição de *RatioCut* é a seguinte:

$$RatioCut(C_1, ..., C_k) = \sum_{i=1}^{k} \frac{c(C_i, C_i \backslash V)}{|C_i|} \tag{3.10}$$

O objetivo é encontrar uma partição que minimize estes valores.

O que queremos demonstrar é a relação do problema de corte normalizado com o problema de encontrar os autovetores da matriz de Laplace. Neste trabalho, não demonstraremos a relação com o problema *RatioCut* (pode ser vista detalhadamente no trabalho de [42]), pois o objetivo do problema *RatioCut* é balancear os subconjuntos C_i pela quantidade de vértices, $|C_i|$, e balancear os clusteres baseando-se apenas na quantidade de vértices não garante que o peso das arestas internas tenham peso alto, ou seja, que pares de vértices de um mesmo cluster sejam similares.

Para facilitar o entendimento, consideremos o caso onde $k = 2$. Nesse caso, nosso objetivo é resolver o seguinte problema de otimização:

$$\min_{C \subset V} Ncut(C, \overline{C}). \tag{3.11}$$

O Teorema 7 a seguir relaciona a forma quadrática da matriz de Laplace com o problema do corte normalizado.

Teorema 7 *Seja D a matriz diagonal e $\mathcal{L}$ a matriz de Laplace de um grafo $G = (V, E, w)$ não-direcionado e com uma função w que atribui peso a cada aresta. Para $k = 2$, o problema de minimizar o corte normalizado ($Ncut$) – ver a equação (3.11) – pode ser reescrito da seguinte forma:*

$$\min_{C \subset V} f' \mathcal{L} f \text{ sujeito a } (Df) \perp \overline{1}, f' D f = a(V), \tag{3.12}$$

onde o vetor f é definido por

$$f_i = \begin{cases} \sqrt{\frac{a(\overline{C})}{a(C)}} & se\ i \in C \\ -\sqrt{\frac{a(C)}{a(\overline{C})}} & se\ i \in \overline{C} \end{cases} \tag{3.13}$$

Prova: Podemos reescrever o problema do corte normalizado ($Ncut$) usando a matriz

de Laplace. Pela Proposição 3, podemos fazer o seguinte cálculo:

$$\begin{aligned}
f'\mathcal{L}f &= \sum_{i,j\in V} w_{ij}(f_i - f_j)^2 \\
&= \sum_{i\in C, j\in \overline{C}} w_{ij}\left(\sqrt{\frac{a(\overline{C})}{a(C)}} + \sqrt{\frac{a(C)}{a(\overline{C})}}\right)^2 + \sum_{i\in \overline{C}, j\in C} w_{ij}\left(-\sqrt{\frac{a(\overline{C})}{a(C)}} - \sqrt{\frac{a(C)}{a(\overline{C})}}\right)^2 \\
&= \sum_{i\in C, j\in \overline{C}} w_{ij}\left(\frac{a(\overline{C})}{a(C)} + 2 + \frac{a(C)}{a(\overline{C})}\right) + \sum_{i\in \overline{C}, j\in C} w_{ij}\left(\frac{a(\overline{C})}{a(C)} + 2 + \frac{a(C)}{a(\overline{C})}\right) \\
&= 2c(C,\overline{C})\left(\frac{a(\overline{C})}{a(C)} + 2 + \frac{a(C)}{a(\overline{C})}\right) \\
&= 2c(C,\overline{C})\left(\frac{a(\overline{C})a(\overline{C})}{a(C)a(\overline{C})} + \frac{a(C)a(C)}{a(C)a(\overline{C})} + 2\frac{a(C)a(\overline{C})}{a(C)a(\overline{C})}\right) \\
&= 2c(C,\overline{C})\left(\frac{a(\overline{C})^2 + a(C)^2 + 2a(C)a(\overline{C})}{a(C)a(\overline{C})}\right) \\
&= 2c(C,\overline{C})\left(\frac{a(\overline{C})(a(\overline{C}) + a(C)) + a(C)(a(\overline{C}) + a(C))}{a(C)a(\overline{C})}\right) \\
&= 2c(C,\overline{C})\left(\frac{a(\overline{C}) + a(C)}{a(C)} + \frac{a(\overline{C}) + a(C)}{a(\overline{C})}\right) \\
&= 2c(C,\overline{C})\left(\frac{a(V)}{a(C)} + \frac{a(V)}{a(\overline{C})}\right) \\
&= 2a(V)Ncut(C,\overline{C}).
\end{aligned} \tag{3.14}$$

Além disso, devemos demonstrar que o vetor $Df \perp \overline{1}$, onde D é a matriz diagonal gerada do grafo e o vetor f é definido como em (3.13). Lembre-se que cada elemento da diagonal da matriz D, denotado por $d(i)$, corresponde a um vértice $i \in V$ e seu valor é a soma dos pesos das arestas incidentes em i. O vetor Df é perpendicular ao vetor constante $\overline{1}$ se, e somente se, $(Df)' \cdot \overline{1} = 0$. Isso é comprovado pelo seguinte cálculo:

$$\begin{aligned}
(Df)'\overline{1} = \sum_{i\in V} d(i)f_i &= \sum_{i\in C} d(i)\sqrt{\frac{a(\overline{C})}{a(C)}} - \sum_{i\in \overline{C}} d(i)\sqrt{\frac{a(C)}{a(\overline{C})}} \\
&= a(C)\sqrt{\frac{a(\overline{C})}{a(C)}} - a(\overline{C})\sqrt{\frac{a(C)}{a(\overline{C})}} = 0.
\end{aligned} \tag{3.15}$$

A igualdade $f'Df = a(V)$ é verdadeira pelo seguinte cálculo:

$$f'Df = \sum_{i\in V} f_i^2 d(i) = \sum_{i\in C} f_i^2 d(i) + \sum_{i\in \overline{C}} f_i^2 d(i) = \sum_{i\in C} \frac{a(\overline{C})}{a(C)} d(i) + \sum_{i\in \overline{C}} \frac{a(C)}{a(\overline{C})} d(i)$$

$$= a(C) + a(\overline{C}) = a(V).$$

Note que para, $C \subseteq V$, temos $a(C) = \sum_{i \in V} d(i)$.

Portanto, demonstramos que resolvendo o problema (3.12) é resolvido o problema do corte normalizado ($Ncut$) para o caso $k = 2$. ■

O problema apresentado no Teorema 7 (problema (3.12)) é um problema discreto de otimização NP-Difícil [42]. Dizemos que o problema (3.12) é discreto pois as coordenadas do vetor solução f apresentam apenas dois valores possíveis, como pode ser notado na definição (3.13) do vetor f.

Abaixo temos uma relaxação para 3.13, onde f assume qualquer valor real.

$$\min_{f \in \mathbb{R}^n} f'\mathcal{L}f \text{ sujeito a } (Df) \perp \overline{1}, f'Df = a(V). \tag{3.16}$$

Pela Proposição 6, é fácil notar que minimizar o problema relaxado (3.16) é semelhante a encontrar o segundo autovetor (associado ao segundo menor autovalor) da matriz de Laplace. Pela equação (3.14), é demonstrado que, resolvendo o problema de minimizar $f'\mathcal{L}f$, encontramos o corte normalizado (consideremos o valor de $a(V)$ constante). Portanto, para $k = 2$, calcular o segundo autovetor fornece uma aproximação para o corte normalizado.

Por fim, é necessário transformar a solução relaxada, cujos valores do vetor f são quaisquer números reais, para valores discretos. O modo mais simples de fazer isso é usar o sinal do vetor f como indicador, ou seja,

$$C = \{v_i | f_i \leq 0\}$$

Vamos considerar agora o caso em que o número de clusteres é maior do que dois ($k > 2$). Seja $n = |V|$, definiremos k vetores, onde o i-ésimo vetor tem a forma $h_i = (h_{i1}, ..., h_{in})$ e, para cada $1 \leq j \leq n$, definimos

$$h_{ij} = \begin{cases} \frac{1}{\sqrt{a(C_i)}} & \text{se } j \in C_i, \\ 0 & \text{caso contrário.} \end{cases} \tag{3.17}$$

Em seguida, construiremos uma matriz H formada pelos k vetores como colunas. A matriz terá as seguintes propriedades, declaradas no lema abaixo.

Lema 1 *Seja a matriz H formada pelos k vetores definidos em (3.17) como colunas e $\mathcal{L}$ a matriz de Laplace. Então valem as seguintes propriedades sobre H e $\mathcal{L}$.*

1. *$H'DH = I$, onde H' é a matriz transposta de H, D a matriz diagonal e I a matriz identidade;*

2. $h_i' \mathcal{L} h_i = 2\frac{c(C,\overline{C})}{a(C_i)}$.

Prova. A parte (1) pode ser verificada notando que todos os vetores formados em (3.17) são ortogonais entre si. A matriz H' terá os vetores de h como linhas e a matriz H como colunas. Dessa forma, cada entrada i e j da matriz resultante da multiplicação $H'DH$ será o produto interno $h_i' D h_j = 0$ para $i \neq j$, o produto $h_i' D h_i = 1$, comprovado pelo seguinte cálculo

$$h_i' D h_i = \sum_{j \in C_i} d_j \left(\frac{1}{\sqrt{a(C_i)}} \right)^2 = \frac{1}{a(C_i)} \sum_{j \in C_i} d_j = 1,$$

lembrando que $\sum_{j \in C_i} d_j = a(C_i)$.

A parte (2) pode ser provada utilizando a Proposição 3 e a definição do vetor h (3.17), fazendo o seguinte cálculo:

$$\begin{aligned} h_i' \mathcal{L} h_i &= \sum_{j=1}^{n} \sum_{k=1}^{n} w_{jk} (h_{ji} - h_{ki})^2 \\ &= \sum_{j \in C} \sum_{k \in \overline{C}} w_{jk} \left(\frac{1}{\sqrt{a(C)}} \right)^2 + \sum_{j \in \overline{C}} \sum_{k \in C} w_{jk} \left(-\frac{1}{\sqrt{a(C)}} \right)^2 \\ &= 2\frac{c(C,\overline{C})}{a(C)}. \end{aligned} \tag{3.18}$$

■

Além disso, temos que

$$h_i' \mathcal{L} h_i = (H' \mathcal{L} H)_{ii}. \tag{3.19}$$

Juntando a definição de corte normalizado (veja no Capítulo 2) e o lema 1 às equações (3.18) e (3.19) teremos

$$Ncut(C_1, ..., C_k) = \sum_{i=1}^{k} \frac{c(C_i, \overline{C_i})}{a(C_i)} = \frac{1}{2} \sum_{i=1}^{k} h_i' \mathcal{L} h_i = \frac{1}{2} \sum_{i=1}^{k} (H' \mathcal{L} H)_{ii} = \frac{1}{2} Tr(H' \mathcal{L} H), \tag{3.20}$$

onde a função Tr retorna a soma dos elementos da diagonal da matriz.

Podemos reescrever o problema de minimizar $Ncut(C_1, ..., C_k)$ como

$$\min_{C_1,...,C_k} Tr(H' \mathcal{L} H) \text{ sujeito a } H'DH = I. \tag{3.21}$$

O problema relaxado permite que os valores da matriz H sejam reais. Então teremos a seguinte forma relaxada:

$$\min_{H \in \mathbb{R}^{n \times k}} Tr(H' \mathcal{L} H) \text{ sujeito a } H'DH = I. \tag{3.22}$$

Fazendo a substituição $U = D^{\frac{1}{2}}H$, obteremos o seguinte problema

$$\min_{U \in \mathbb{R}^{n \times k}} Tr(U' D^{-\frac{1}{2}} \mathcal{L} D^{-\frac{1}{2}} U) \text{ sujeito a } U'U = I. \tag{3.23}$$

Note que o Problema 3.23 corresponde a encontrar os autovetores da matriz simétrica $D^{-\frac{1}{2}} \mathcal{L} D^{-\frac{1}{2}}$. A restrição $U'U = I$ significa que U será formado por vetores ortonormais como colunas. Logo, a matriz U contém os primeiros k autovetores como colunas.

Podemos relacionar os autovetores da matriz $D^{-\frac{1}{2}} \mathcal{L} D^{-\frac{1}{2}}$ aos autovetores da matriz de transição. Essa relação será demonstrada no Teorema 8.

Teorema 8 *Dada a matriz $D^{-\frac{1}{2}} \mathcal{L} D^{-\frac{1}{2}}$ formada a partir de um grafo G, tomemos também a matriz U com os respectivos k menores autovetores de $D^{-\frac{1}{2}} \mathcal{L} D^{-\frac{1}{2}}$. Cada coluna de U corresponde a um autovetor u_i associado ao autovalor λ_i, formando o par (λ_i, u_i), para $1 \leq i \leq k$. O espectro da matriz de transição M do grafo G será formado pelos pares $(1 - \lambda_i, D^{-\frac{1}{2}} u_i)$*

Prova: Inicialmente, escolheremos a i-ésima coluna da matriz U. Essa coluna será o autovetor u_i correspondente ao i-ésimo autovalor λ_i, $1 \leq i \leq k$.

Pela equação fundamental dos autovetores teremos

$$D^{-\frac{1}{2}} \mathcal{L} D^{-\frac{1}{2}} u_i = \lambda_i u_i,$$

multiplicando a igualdade por $D^{-\frac{1}{2}}$, a equação pode ser reescrita como

$$D^{-1} \mathcal{L} D^{-\frac{1}{2}} u_i = \lambda_i D^{-\frac{1}{2}} u_i. \tag{3.24}$$

Pela definição da matriz de Laplace e da matriz de transição, sabemos que $\mathcal{L} = D - A$ e que $M = D^{-1}A$. Assim, teremos a seguinte igualdade

$$D^{-1}\mathcal{L} = D^{-1}(D - A) = I - D^{-1}A = I - M \tag{3.25}$$

Substituindo (3.25) em (3.24) teremos

$$D^{-1} \mathcal{L} D^{-\frac{1}{2}} u_i = (I - M) D^{-\frac{1}{2}} u_i = D^{-\frac{1}{2}} u_i - M D^{-\frac{1}{2}} u_i = \lambda_i D^{-\frac{1}{2}} u_i. \tag{3.26}$$

Reescrevendo a equação (3.26), podemos chegar no seguinte

$$\begin{aligned} & D^{-\frac{1}{2}} u_i - M D^{-\frac{1}{2}} u_i = \lambda_i D^{-\frac{1}{2}} u_i \quad (\text{ subtraindo pelo vetor } -D^{-\frac{1}{2}} u_i) \\ \Rightarrow \quad & -M D^{-\frac{1}{2}} u_i = \lambda_i D^{-\frac{1}{2}} u_i - D^{-\frac{1}{2}} u_i \\ \Rightarrow \quad & M D^{-\frac{1}{2}} u_i = (1 - \lambda_i) D^{-\frac{1}{2}} u_i \end{aligned} \tag{3.27}$$

Portanto, pela equação (3.27), conclui-se que o par $(1 - \lambda_i, D^{-\frac{1}{2}}u_i)$, onde u_i é uma coluna da matriz U, corresponde ao autovalor e autovetor da matriz de transição M. ■

A partir do problema (3.23) e sua relação com o Teorema 8, fica claro que calcular os k autovetores da matriz de transição é semelhante a encontrar uma solução aproximada para o corte normalizado ($Ncut$).

Para transformar os valores reais encontrados pela matriz U em uma solução discreta, interpretemos cada linha de U como pontos no espaço $\mathbb{R}^k$. Cada ponto corresponderá a um vértice do grafo G. Para encontrar as partições, normalmente, é utilizado um algoritmo para *k-means* sobre esses pontos [55].

3.2 *Markov Clustering(MCL)*

Nesta seção, apresentaremos o algoritmo proposto em [54] mas, antes de descrever o algoritmo, daremos uma breve explicação sobre a relação entre cadeia de Markov e grafos.

Uma Cadeia de Markov é definida como um conjunto de estados S e um conjunto T de transições (i, j) entre estados $i, j \in S$. A cada momento ocorre uma transição. Para cada transição há uma probabilidade $p_{i,j}$ tal que, se em um determinado momento estamos no estado i, então com probabilidade $p_{i,j}$ estaremos no estado j no momento seguinte. Todas probabilidades são positivas e para cada estado i vale

$$\sum_{j \in S:(i,j) \in T} p_{i,j} = 1.$$

A Cadeia de Markov pode ser visualizada como um grafo cujo o conjunto de vértices são os estados e o conjunto de arestas são as transições. Cada aresta tem peso $p_{i,j}$.

Quando transitamos de um vértice u para um vértice v, estamos passando de um estado corrente para um estado futuro na cadeia de Markov, o que chamamos de *um passo em um passeio aleatório.*

Dado um grafo de entrada G, o algoritmo MCL considera a matriz de transição $M(G)$ como uma Cadeia de Markov. A ideia chave do algoritmo é que um passeio aleatório começando de um vértice de um cluster denso (subgrafo de vértices altamente similares) com pouca probabilidade deixará o cluster até que vários vértices do cluster sejam visitados. Descrevemos o MCL no Algoritmo 3.

Para simular os passos de um passeio dentro do grafo, a matriz de transição M é elevada a uma potência e, obtendo M^e. Cada entrada $p^e_{u,v}$ de M^e corresponde à probabilidade de ir de u para v em e passos em um passeio aleatório.

Na próxima etapa do algoritmo, chamada de *inflação* (linhas 4 a 11 no algoritmo 1), cada entrada $p^e_{u,v}$ é elevada a um valor constante r e re-normalizada logo em seguida.

Algoritmo 3: Markov Clustering (MCL)

Entrada: $G = (V, E, w)$, parâmetro de expansão e, parâmetro de inflação r

$M \leftarrow M(G)$;
while M *não é idempotente* **do**
  $M \leftarrow M^e$;
  forall $u \in V$ **do**
    forall $v \in V$ **do**
      $p_{uv} \leftarrow p_{uv}^r$
    end
    forall $v \in V$ **do**
      $p_{uv} \leftarrow \frac{p_{uv}}{\sum_{w \in V} p_{uw}}$
    end
  end
end
$H \leftarrow$ grafo induzido pelas entradas diferentes de zero em M ;
$C \leftarrow$ partição induzida por componentes conexos de H ;

O objetivo é que os valores de transições com alta probabilidade sejam intensificados e transições baixas sejam degradadas.

A operação de inflação é a seguinte: Dada uma matriz $M \in \mathbb{R}^{k \times l}$ e um valor real, $r > 0$, a matriz resultante da re-normalização de cada linha de M com suas entradas elevadas a um fator r é denotada por $\Gamma_r M$. Γ_r é chamado de operação de *inflação* com uma potência r. Formalmente esta operação $\Gamma_r : \mathbb{R}^{k \times l} \to \mathbb{R}^{k \times l}$ é definida como:

$$(\Gamma_r M)_{u,v} = (p_{u,v})^r / \sum_{w=1}^{l} (p_{u,w})^r \tag{3.28}$$

para $u = 1, ..., k$ e $v = 1, ..., l$.

Para ilustrar a operação de inflação utilizaremos como exemplo um vetor v formado pelos elementos (0, 0.500, 0, 0.166, 0.333) e uma potência $r = 2$. Veja abaixo o exemplo:

$$\begin{pmatrix} 0 & 0.500 & 0 & 0.166 & 0.333 \end{pmatrix}$$
$$\downarrow inflacao(\Gamma_2 v)$$
$$\begin{pmatrix} 0 & 0.642 & 0 & 0.071 & 0.285 \end{pmatrix}$$

Note que, aplicando a inflação, o maior valor de v intensificará, passando de 0.5 para 0.642. Em contrapartida, os menores valores foram diminuídos.

O fato importante do algoritmo é que, em um passeio aleatório de e passos, partindo de um vértice de um subgrafo denso C_i em G, é mais provável chegarmos a um vértice

de C_i do que a um vértice fora de C_i. Portanto, os passeios que ligam subgrafos densos distintos terão o valor da probabilidade de transição baixo.

O principal processo realizado pelo algoritmo MCL é a alternância entre exponenciação e inflação da matriz de transição M, representado pela operação $\Gamma_r(M^e)$. Dessa forma, em uma iteração k do laço da linha 2 do algoritmo, teremos como resultado uma matriz $M_k = \Gamma_r(M^e_{k-1})$. O laço da linha 2 é repetido até que a matriz M_k seja igual a M_{k+1}, satisfazendo $M_k = M_{k+1} = M_{k+2} = ... = M_\infty$. Em outras palavras, a matriz M_k não se altera após a alternância entre exponenciação e inflação. Quando é alcançado este estado, dizemos que a matriz M_k é idempotente.

Na alternância entre exponenciação e inflação, valores baixos tenderão para zero. Assim, as arestas que ligam vértices de diferentes subgrafos densos serão degradadas de forma que, no resultado final do algoritmo, cada componente conexo representará um particionamento.

Na tese de van Dongen [54], não é apresentada uma prova de que o processo de alternância entre exponenciação e inflação levará a "bons" particionamentos. Entretanto, a heurística que leva para a formulação do algoritmo sugere que particionamentos significativos possam ser determinados.

Na próxima seção será apresentado um exemplo mais detalhado com a execução do algoritmo.

3.2.1 Execução do MCL

Dado o grafo G_1 (Figura 3.5) com suas correspondentes matrizes de adjacência $A[G_1]$ e diagonal $D[G_1]$ (Figura 3.6), a seguinte matriz normalizada $M[G_1] = D[G_1]^{-1} \times A[G_1]$ (Figura 3.7) é construída.

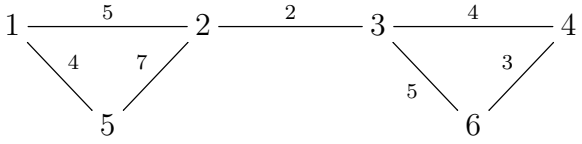


Figura 3.5: Grafo G_1.

Quando a matriz M é elevada a uma constante e, o objetivo é simular um passeio aleatório de e passos. Inicialmente, cada entrada $p_{u,v}$ de M representa a transição inicial (probabilidades referente a um passo). A matriz M^2, por exemplo, corresponde a um passeio de dois passos. Cada posição $p^2_{u,v}$ de M^2 terá o seguinte valor:

$$p^2_{u,v} = \sum_{w \in V} p_{u,w} p_{w,v}.$$

Este valor é a probabilidade de ir de u para v em 2 passos. O valor $p^2_{u,v}$ é a probabilidade de transitar de um vértice u para um vértice qualquer w com probabilidade $p_{u,w}$ e, a partir do vértice w, transitar para o vértice v com probabilidade $p_{w,v}$. Para simular um passeio com e passos, eleva-se a matriz M a uma potência e, sendo que cada valor $p^e_{u,v}$ de M^e corresponde à probabilidade de transição de u para v após e passos.

$$A_{6\times6} = \begin{bmatrix} 0 & 5 & 0 & 0 & 4 & 0 \\ 5 & 0 & 2 & 0 & 7 & 0 \\ 0 & 2 & 0 & 4 & 0 & 5 \\ 0 & 0 & 4 & 0 & 0 & 3 \\ 4 & 7 & 0 & 0 & 0 & 0 \\ 0 & 0 & 5 & 3 & 0 & 0 \end{bmatrix} \quad D_{6\times6} = \begin{bmatrix} 9 & 0 & 0 & 0 & 0 & 0 \\ 0 & 14 & 0 & 0 & 0 & 0 \\ 0 & 0 & 11 & 0 & 0 & 0 \\ 0 & 0 & 0 & 7 & 0 & 0 \\ 0 & 0 & 0 & 0 & 11 & 0 \\ 0 & 0 & 0 & 0 & 0 & 8 \end{bmatrix}$$

Figura 3.6: Matriz de Adjacência $A[G_1]$ e Matriz Diagonal $D[G_1]$.

$$M_{6\times6} = \begin{bmatrix} 0.00000 & 0.55556 & 0.00000 & 0.00000 & 0.44444 & 0.00000 \\ 0.35714 & 0.00000 & 0.14286 & 0.00000 & 0.50000 & 0.00000 \\ 0.00000 & 0.18182 & 0.00000 & 0.36364 & 0.00000 & 0.45455 \\ 0.00000 & 0.00000 & 0.57143 & 0.00000 & 0.00000 & 0.42857 \\ 0.36364 & 0.63636 & 0.00000 & 0.00000 & 0.00000 & 0.00000 \\ 0.00000 & 0.00000 & 0.62500 & 0.37500 & 0.00000 & 0.00000 \end{bmatrix}$$

Figura 3.7: Matriz de Transição $M^1[G_1]$.

Após simular o passeio, a próxima etapa do algoritmo é a operação de inflação. A operação de inflação é responsável por intensificar valores correspondentes com alta probabilidade de transição e enfraquecer aqueles com baixa probabilidade. Esse efeito é obtido ao elevar cada entrada da matriz M^e a um valor r e, em seguida, normalizar com todos os valores da mesma linha.

Após várias iterações de expansão e inflação, a matriz ficará idempotente. Veja na Figura 3.8 (e também no exemplo anterior para o vetor X_6) que, aplicando a inflação na matriz idempotente, não ocorrerá nenhuma alteração.

Os particionamentos são encontrados a partir dos subgrafos induzidos construídos a partir das entradas da matriz idempotente que tenham valores diferente de zero. Cada componente conexo será interpretado como um particionamento.

O resultado do algoritmo aplicado no exemplo da Figura 3.5, com a respectiva matriz de transição da Figura 3.7, resulta na matriz da Figura 3.8. Tem-se na Figura 3.9 o grafo criado a partir da matriz final.

$$A_{6\times 6} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}$$

Figura 3.8: Matriz idempotente.

Figura 3.9: Grafo com o passeio da matriz resultante. (Figura 3.8) .

Os subgrafos induzidos são gerados a partir da matriz resultante. Para cada componente conexo, forma-se um cluster, o que finaliza a execução do algoritmo. Veja na Figura 3.9 os conjuntos de vértices $\{1, 2, 5\}$ e $\{3, 4, 6\}$ formam dois clusteres.

Em outro exemplo, dada pela Figura 3.2.1, é possível perceber os sucessivos estágios realizados pelo algoritmo MCL. Note que, as arestas com cores fortes estão em regiões densas. Essas arestas são intensificadas a cada iteração do MCL. Por outro lado, as arestas de regiões esparsas são apagadas.

3.3 *Iterative Conductance Cutting*

Nesta seção, será apresentado o algoritmo *Iterative Conductance Cutting* (ICC) proposto em [34].

É fácil encontrar um particionamento por meio de $k-1$ sucessivos cortes mínimos. Para isso, basta aplicar um corte $E(S, \overline{S})$ e, em seguida, aplicar outros cortes para os subgrafos induzidos de S e $\overline{S}$. Entretanto, não é garantido que os conjuntos formados pelo corte mínimo sejam bons clusteres [34]. Por exemplo, o corte mínimo poderá separar apenas um único vértice no subconjunto C_i, assim o subconjunto C_i não formará um subgrafo com alta densidade interna.

Um "bom" cluster C_i, com $0 \leq i \leq k$ e pertencente a um particionamento $\mathcal{C} = \{C_1, ..., C_k\}$ de um grafo $G = (V, E)$, possui seu subgrafo induzido denso e, ao mesmo tempo, pouca conectividade entre seus vértices e os vértices de $\overline{C_i}$. A notação $\overline{C_i}$ representa os vértices do subconjunto $V \backslash C_i$.

Uma boa forma de avaliar um "bom" cluster é procurar um conjunto C_i de vértices

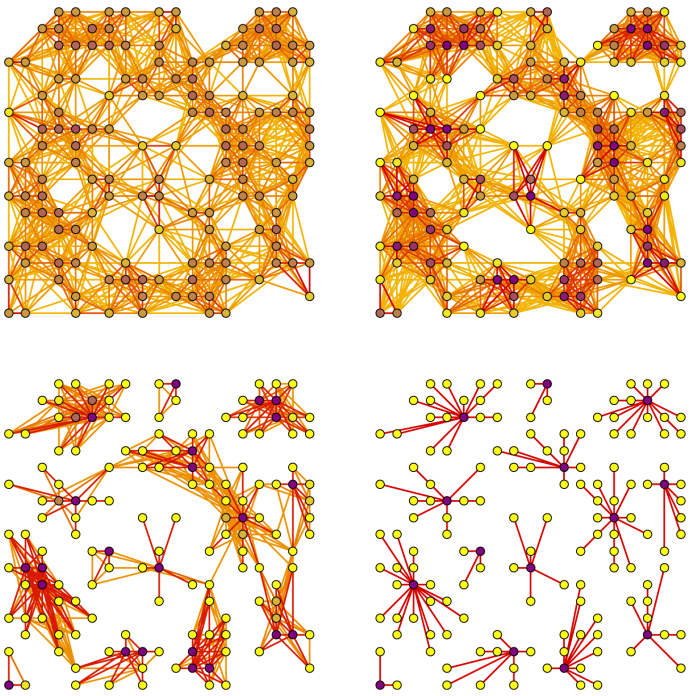


Figura 3.10: Estágios sucessivos do Algoritmo MCL [54]

que minimize o custo do corte $c(C_i, \overline{C_i})$ e que também maximize a densidade interna de C_i e $\overline{C_i}$. Na Seção 2 descrevemos algumas funções que podem ser utilizadas para avaliar um "bom" particionamento. No algoritmo ICC foi utilizada a função de *condutância*. A *condutância*, como visto na Seção 2, compara o custo do corte e os pesos das arestas internas dos dois subgrafos induzidos. A condutância $\varphi(C_i)$ de um corte $c(C_i, V \backslash C_i)$ em G é definida como:

$$\varphi(C_i) = \begin{cases} 1, & \text{se } C_i \in \{\emptyset, V\} \\ 0, & \text{se } C_i \notin \{\emptyset, V\} \text{ e } E(C_i, V \backslash C_i) = 0 \\ \dfrac{c(C_i, V \backslash C_i)}{\min\{a(C_i), a(V \backslash C_i)\}}, & \text{caso contrário} \end{cases}$$

onde

$$a(C_i) = \sum_{v \in C_i} \sum_{w \in V} w(v, w) = 2 \sum_{e \in E(C_i)} w(e) + \sum_{e \in c(C_i, V \backslash C_i)} w(e)$$

A condutância $\varphi(G)$ de um grafo G é o valor da condutância mínima dentre todos possíveis cortes de G:

$$\varphi(G) = \min_{C_i \subset V} \varphi(C_i).$$

A ideia do algoritmo é que, em um bom particionamento $\mathcal{C} = \{C_1, ..., C_k\}$ do grafo, cada C_i tem um valor de condutância baixo e, ao mesmo tempo, o valor de condutância de $G[C_i]$ não é baixo, onde $G[C_i]$ é o subconjunto induzido por C_i.

Inicialmente, o algoritmo considera como particionamento apenas uma partição que contém apenas um conjunto com todos os vértices. A cada iteração, o algoritmo verifica se existe uma parte $C \in \mathcal{C}$ tal que a condutância do subgrafo induzido por esta parte é menor do que um limite. Se existir tal parte, ela será um indicativo de que C não é uma boa parte, pois existem arestas de peso baixo conectando seus vértices. O algoritmo encontra então o corte $S \subset C$ de condutância mínima em $G[C]$ e cria duas novas partes no lugar de C. O algoritmo procede enquanto houver partições que possam ser divididas. Descrevemos o ICC no algoritmo 4.

Algoritmo 4: Iterative Conductance Cutting (ICC)

Entrada: $G = (V, E, w)$, limiar de condutância $0 < \alpha^* < 1$
1 $\mathcal{C} \leftarrow \{V\}$;
2 **while** $\exists\ C \in \mathcal{C}$ *com* $\varphi(G[C]) < \alpha^*$ **do**
3 $\quad x \leftarrow$ autovetor de $M(G[C])$ associado ao segundo maior autovalor ;
4 $\quad S \leftarrow \{S \subset C \mid \max_{v \in S}(x_v) < \min_{w \in C \setminus S}(x_w)\}$;
5 $\quad C' \leftarrow \arg\min_{S \in S}(\varphi(S))$;
6 $\quad \mathcal{C} \leftarrow (\mathcal{C} \setminus \{C\}) \cup \{C', C \setminus C'\}$;
7 **end**

O problema de encontrar o corte de condutância mínima é NP-Difícil, dessa forma, o algoritmo ICC utiliza um algoritmo de aproximação para o corte de condutância mínima (linhas 3-5 do Algoritmo 4). A heurística para aproximar o corte de condutância mínima pode ser descrita da seguinte forma: primeiramente, calcula-se o autovetor $x \in \mathbb{R}^n$ relativo ao segundo maior autovalor λ_2 da matriz $M(G[C])$, onde $G[C]$ é o subgrafo induzido pelo subconjunto de vértices C. Ordene os valores do autovetor $x = (x_1, ..., x_n)$ de forma crescente, tal que $x_{i_1} \leq x_{i_2} \leq ... \leq x_{i_n}$. Em seguida, encontra-se o valor t que minimiza a condutância $\varphi(\{i_1, ..., i_t\})$, $1 \leq t \leq n$.

No trabalho de Kannan *et al.* [34], é provado um limite superior para a aproximação com a seguinte relação:

$$\varphi(G) \geq 1 - \lambda_2 \geq \frac{\phi(G)^2}{2}, \tag{3.29}$$

onde $\phi(G)$ é o valor do corte de condutância encontrado pelo algoritmo aproximado que utiliza a heurística do segundo autovetor λ_2.

Veja no apêndice A a prova da garantia de aproximação do algoritmo ICC para o problema de clusterização. Veja também, no apêndice A.2, a demonstração da garantia

de aproximação, dada em (3.29), para o algoritmo que utiliza a heurística do segundo autovetor.

3.4 *Geometric MST Clustering*

Geometric MST Clustering (GMC) é um algoritmo para clusterização em grafos que combina particionamento espectral com técnicas de clusterização no espaço $\mathbb{R}^n$ [12].

A ideia do algoritmo é imergir um grafo G no espaço $\mathbb{R}^n$ e atribuir peso às arestas. Os valores dos pesos serão dados pela relação de distância entre vértices. Em seguida, é criada uma árvore geradora mínima (*Minimum Spanning Tree* - MST) com os novos pesos das arestas de G. Seja T uma MST gerada com os novos pesos. Para encontrar os clusteres, são removidas arestas de T com peso superior a um limiar τ. Chamaremos de $F(T, \tau)$ a floresta induzida por todas as arestas de T com peso no máximo τ. Para cada limiar τ, os componentes conexos de $F(T, \tau)$ gerarão os clusteres $\mathcal{C}(\tau)$. O limiar τ é aquele escolhido dentre os valores dos pesos das arestas de T que maximiza a função de qualidade (ver Seção 1.2) aplicada sobre $\mathcal{C}(\tau)$. Veja o Algoritmo 5 para a descrição em pseudocódigo do Algoritmo GMC.

Algoritmo 5: Geometric MST Clustering (GMC)

Entrada: $G = (V, E, w)$, número de dimensões para imersão d , função avaliadora de clusteres *quality*

1 $(1, \lambda_2, ..., \lambda_{d+1}) \leftarrow d + 1$ maiores autovalores de $M(G)$;
2 $d' \leftarrow \max\{i | 1 \leq i \leq d, \lambda_i > 0\}$;
3 $x^{(1)}, ..., x^{(d')} \leftarrow$ autovetores de $M(G)$ associado com $\lambda_1, ..., \lambda_{d'}$;
4 **forall** $e = (u, v) \in E$ **do**
5 $\quad w(e) \leftarrow \sum_{i=1}^{d'} |x_i[u] - x_i[v]|$;
6 **end**
7 $T \leftarrow$ árvore geradora mínima de G em relação a w;
8 $C \leftarrow C(\tau)$ tal que a $quality(C(\tau))$ é máxima sobre todo $\tau \in \{w(e) | \in T\}$;

Na primeira linha do algoritmo, são selecionados os autovetores associados aos d maiores autovalores da matriz de transição (ver definição 13) do grafo de entrada. Note que os k maiores autovalores de $M[G]$ correspondem aos k menores autovalores generalizados da matriz de Laplace (Proposição 8).

Nas linhas (2) e (3) do algoritmo 5, são selecionados apenas os autovetores associados a autovalores positivos. Segundo Gaertler [25], essa seleção é feita devido a uma propriedade observada em grafos bipartidos. Nesses grafos, temos $\lambda_2 = -1$ e, ao executarmos o

algoritmo GMC (utilizando autovalores negativos), obtemos a bipartição do grafo como resultado. Isso não leva a um "bom" particionamento, já que em cada parte não haveria arestas ligando os vértices internos.

O algoritmo (GMC) segue a mesma estrutura genérica para clusterização espectral (ver algoritmo 2) discutida na Seção 3.1.3. Após criar a matriz de transição do grafo dado como entrada e selecionar os autovetores úteis, o próximo passo é interpretar os autovetores selecionados como clusteres. O artifício usado pelo algoritmo é imergir o grafo no espaço.

Uma imersão do grafo G no espaço é construída a partir de d autovetores $\{x_1, x_2, ..., x_d\}$ da matriz de transição $M(G)$ que estão associados aos maiores autovalores menores ou iguais a um. Formalmente, podemos enunciar o problema da imersão de um grafo G no espaço d-dimensional da seguinte forma:

Problema 1 (Imersão no Espaço) *[25] Dado como entrada um grafo $G = (V, E)$ com n vértices – sem perda de generalidade assumimos que $V = \{1, 2, ..., n\}$ – e uma função de custo $g : \mathbb{R}^{n \times d} \to \mathbb{R}$, onde d representa o número de dimensões. O objetivo é determinar uma função $\varrho : V \to \mathbb{R}^d$ tal que, se a função de custo g tem um mínimo global, então o valor de $g(A)$ é o mínimo com a matriz A construída como*

$$A = \begin{pmatrix} \varrho(1) \\ \vdots \\ \varrho(n) \end{pmatrix}.$$

No trabalho de Gaertler [25], é usada a equação 3.6 como função de custo. Para dar maior compreensão de como é realizada a imersão, podemos simplificar o problema para apenas uma dimensão. Nesse caso, teremos apenas o posicionamento de cada vértice em uma reta. Como $\varrho : V \to \mathbb{R}$, a função ϱ pode ser interpretada como um vetor indexado pelos vértices. A função de custo será a soma de todas as distâncias entre dois vértices adjacentes. Claramente, o valor mínimo da função de custo (dada pela equação 3.6) corresponde ao posicionamento em que todos os pontos estão na mesma posição. Para evitar que isso ocorra, é postulado que o vetor que define o posicionamento de cada vértice tenha tamanho unitário e seja ortogonal ao vetor $\overline{1}$. A partir da Proposição 6, podemos notar que todos os autovetores relacionados ao autovalor λ_2 são os mínimos globais para a função de custo.

Seja $d' \leq n$. Cada índice de um autovetor x_i $(1 \leq i \leq d')$ corresponderá a um vértice. A seleção de d' autovetores corresponde à imersão do grafo $G = (V, E)$ em d' dimensões. Após a imersão, um vértice $v \in V$ terá um ponto p_u correspondente. As coordenadas do ponto p_u são os índices u de cada autovetor, isto é, $p_u = (x_1[u], x_2[u], ..., x_{d'}[u])$. Ao contrário do peso atribuído a arestas de um grafo, que é diretamente proporcional

à similaridade entre os vértices correspondentes, a distância entre dois pontos de uma imersão é inversamente proporcional a sua similaridade, ou seja, quanto menor for a distância, maior será a similaridade entre os pontos.

Para ilustrarmos o processo de imersão de um grafo e a sua relação com clusteres, tomaremos como exemplo um grafo (ver Figura 3.11) e aplicaremos o processo descrito pelo algoritmo GMC. A matriz de adjacência do grafo G_1 é representada pela Figura 3.12. Para simplificar a ilustração, será considerada a imersão do grafo G_1 em uma reta.

Sejam o grafo $G_1 = (V_1, E_1)$, representado pela Figura 3.11, e x o autovetor associado ao maior autovalor de $M(G_1)$. Suponha que seja aplicada a imersão de G_1 em apenas uma dimensão ($d = 1$), definindo uma função de $\delta : E_1 \to \mathbb{R}$ por

$$\delta(e) = |x[u] - x[v]|, \tag{3.30}$$

onde $e = (u, v) \in E_1$. Podemos interpretar x como uma função que atribui um valor real a cada vértice, sendo δ o módulo da diferença entre dois valores reais.

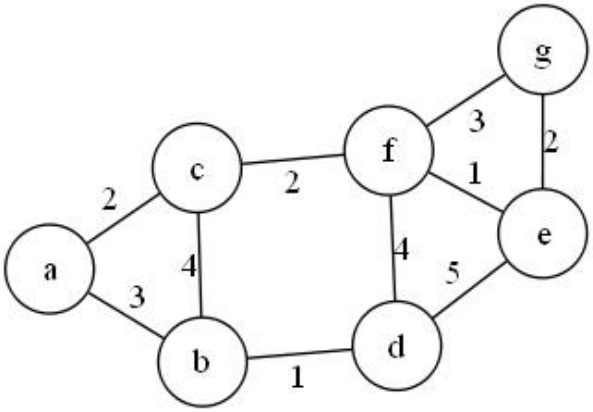


Figura 3.11: Grafo G_1 dado como entrada para o exemplo de imersão no espaço $\mathbb{R}^n$.

$$A = \begin{bmatrix} 0 & 3 & 2 & 0 & 0 & 0 & 0 \\ 3 & 0 & 4 & 1 & 0 & 0 & 0 \\ 2 & 4 & 0 & 0 & 0 & 2 & 0 \\ 0 & 1 & 0 & 0 & 5 & 4 & 0 \\ 0 & 0 & 0 & 5 & 0 & 1 & 2 \\ 0 & 0 & 2 & 4 & 1 & 0 & 3 \\ 0 & 0 & 0 & 0 & 2 & 3 & 0 \end{bmatrix}$$

Figura 3.12: Matriz de Adjacência do grafo G_1. Cada linha e coluna obedecem à ordem lexicográfica dos rótulos dos vértices.

$$M = \begin{bmatrix} 0.377 & 0.535 & 0.054 & 0.461 & 0.739 & 0.421 & -0.083 \\ 0.377 & 0.452 & -0.244 & -0.686 & -0.120 & 0.078 & -0.111 \\ 0.377 & 0.385 & 0.251 & 0.417 & -0.505 & -0.277 & 0.144 \\ 0.377 & -0.272 & 0.490 & -0.137 & 0.159 & -0.251 & -0.422 \\ 0.377 & -0.354 & -0.449 & 0.263 & -0.294 & 0.388 & -0.362 \\ 0.377 & -0.212 & -0.417 & 0.025 & 0.250 & -0.410 & 0.368 \\ 0.377 & -0.338 & 0.507 & -0.227 & -0.087 & 0.597 & 0.717 \end{bmatrix}$$

Figura 3.13: Matriz com os autovetores da matriz de adjacência do grafo G_1

Na Figura 3.13, temos, em cada coluna, os autovetores da matriz de transição do grafo G_1. Na primeira coluna, temos o autovetor correspondente ao primeiro maior autovalor λ_1, na segunda coluna, o autovetor correspondente ao segundo maior autovalor λ_2 e assim sucessivamente para cada autovalor λ_i $(1 \leq i \leq n)$. Considerando apenas uma dimensão para a imersão $(d = 1)$, será escolhido apenas o segundo autovetor (correspondente ao autovalor λ_2) para representar a posição de cada vértice na reta. A representação é dada pela Figura 3.14.

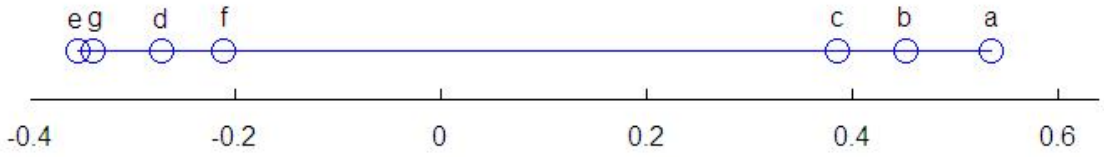


Figura 3.14: Reta com a imersão dos vértices do grafo G_1.

Note que os pontos $\{e, g, d, f\}$ estão posicionados próximos um dos outros e distantes dos pontos $\{c, b, a\}$. Para o grafo G_1, que é um grafo com poucos vértices, é possível perceber que os clusteres são os conjuntos de vértices $\{e, g, d, f\}$ e $\{c, b, a\}$ apenas com o auxílio dessa imersão.

Para grafos não tão simples, como grafos com número maior de vértices, é necessário determinar quais os conjuntos de pontos correspondem a cada cluster. Para determinar os clusteres, o algoritmo GMC modifica os pesos das arestas para o valor dado pela função δ definida na equação (3.30). Em seguida, é criada uma árvore geradora mínima do grafo imerso. Em seguida, o algoritmo (GMC) utiliza uma técnica semelhante àquela descrita no trabalho de Zahn [60] – constrói uma árvore geradora mínima do grafo e remove as arestas com peso superior a um limiar. Diferentemente do algoritmo de Zahn, onde o limiar é dado como entrada, no algoritmo GMC, o limiar é escolhido entre todos os valores de pesos da aresta da árvore geradora mínima tal que uma função de qualidade dos clusteres seja maximizada. As funções de qualidade dos clusteres são aquelas que apresentam

a estrutura que relaciona a densidade intraclusteres *versus* a esparsidade interclusteres. Podemos usar qualquer uma das funções de custo definidas na Seção 1.2.

Uma vez que a função de qualidade seja calculada rapidamente, o tempo de execução do algoritmo GMC é dominado pelo tempo de calcular os autovetores da matriz de transição e o tempo da criação da árvore geradora mínima.

3.5 K-Centros

Nesta seção, será discutido o problema dos k-centros. O algoritmo de aproximação descrito para o problema foi proposto por Teofilo Gonzales [28]. Esse algoritmo é uma $2-$aproximação e utiliza uma estratégia gulosa para encontrar a solução aproximada.

Uma instância do problema dos *k-centros* consiste em um grafo completo $G = (V, E)$ com uma função w que atribui peso a cada aresta $e = (u, v) \in E$, com $w_e \geq 0$. O objetivo do problema é encontrar um subconjunto de vértices $S \subseteq V$ de tamanho no máximo k tal que $w(S) = \max_{v \in V} \min_{u \in S} w_{(v,u)}$ seja minimizado, ou seja, o valor máximo da distância de um vértice até o seu centro mais próximo seja minimizada.

É importante salientar que, nesta seção, será considerada a versão métrica do problema dos *k-centros*. Considere uma função $d : V \times V \to \mathbb{R}^+$ que retorna a distância mínima entre dois vértices quaisquer de V. Na versão simétrica, a função d satisfaz as seguintes propriedades:

- $\forall v \in V, d(v, v) = 0$;
- **Desigualdade Triangular:** $\forall u, v, t \in V, d(u, v) + d(v, t) \geq d(u, t)$;
- **Simetria:** $\forall u, v \in V, d(u, v) = d(v, u)$.

O problema dos *k-centros* métrico é um problema $NP-$Difícil [36].

O problema sem a desigualdade triangular é $NP-$Difícil de aproximar dentro de qualquer fator constante [36]. O problema considerando a desigualdade triangular e desconsiderando a simetria, chamado de problema dos *k-centros* assimétrico, foi mostrado que é $\Omega(\log n)$ difícil de aproximar [18]. A versão métrica (obedecendo também a desigualdade triangular), considerada neste trabalho, tem o fator de aproximação 2 como o melhor possível para o problema [31], a menos que $P = NP$.

O algoritmo de Gonzalez [28] é bastante simples. Inicialmente, é selecionado um vértice arbitrário como o primeiro centro, esse centro é atribuído ao conjunto S. Então, em cada iteração, o novo centro será o vértice mais distante de algum centro pertecente ao conjunto S. Essa escolha gulosa será feita até que a quantidades de centros atribuídos a S seja do tamanho k. Abaixo é descrito o algoritmo em pseudocódigo.

Algoritmo 6: Algoritmo de Gonzalez para o problema K-Centros

Entrada: $G(V, E)$ grafo completo, k quantidade de centros d função que retorna a distancia entre dois vértice do grafo G

1 $V' \leftarrow V$;
2 seja u um vértice escolhido aleatoriamene de V ;
3 $S \leftarrow S \cup \{u\}$;
4 $V' \leftarrow V' - \{u\}$;
5 **for** $j \leftarrow 1$ *até* k **do**
6 $\quad S \leftarrow S \cup \left\{ \arg\max_{v \in V'} d(v, S) \right\}$;
7 $\quad V' \leftarrow V' - \{u\}$;
8 **end**
9 Retorne S;

Teorema 9 *O algoritmo de Gonzalez calcula uma* $2-$*aproximação para o problema dos* k-centros.

Prova: Seja S^* uma solução ótima para o problema dos *k-centros* e S uma solução do algoritmo de Gonzales. Considere também o valor r^* para o maior raio da solução ótima, ou seja, r^* será a distância máxima de um vértice até um dos k centros de S^* mais próximo. Suponha por absurdo que o valor r, distância de um vértice até o centro em S mais próximo, seja $r > 2r^*$. Dessa forma, todos os k centros selecionados pelo algoritmo de Gonzalez e também um ponto mútuo entre os centros (ponto que esteja sendo coberto por esferas de raio r de dois centros), estão distantes pelo menos r entre si. Assim, pela desigualdade triangular, para cobrir esses $k+1$ vértices dentro de uma distância r^* (com o raio de cobertura r^*), cada cobertura de raio r^* conterá no máximo um desses $k+1$ centros. Isso implica que pelo menos $k+1$ coberturas de raio r^* serão requeridas para cobrir todos os vértices com uma distância de no máximo r^*, o que é uma contradição. ■

Capítulo 4

GRASP para o problema de Particionamento

Nesta seção, é apresentada uma nova abordagem que utiliza a meta-heurística GRASP para tratar o problema de particionamento em grafos. Na Seção 4.1 daremos uma visão geral sobre a meta-heurística GRASP e na Seção 4.2 falaremos sobre a aplicação do GRASP para tratar o problema de particionamento em grafos.

4.1 A Meta-heurística GRASP

Meta-heurísticas são métodos computacionais que buscam gerar soluções boas fazendo uma varredura pelo espaço de soluções, usando uma combinação de heurísticas com componentes aleatórios. Elas possuem uma estrutura com componentes genéricos que são adaptados ao problema de otimização específico. Apesar de esses métodos sempre procurarem uma boa solução, não há garantia que a solução ótima será encontrada.

Meta-heurísticas estão entre os métodos mais efetivos para resolver problemas de otimização combinatória. Entre as meta-heurísticas mais conhecidas, podemos citar *simulated annealing*, busca tabu, GRASP, algoritmos genéticos, colônia de formigas e várias outras. Geralmente, esses métodos são baseados em diferentes paradigmas e oferecem diferentes mecanismos para escapar de soluções ótimas locais.

Consideraremos problemas de otimização combinatória cujo o objetivo é minimizar o valor de uma função $f(S)$ sobre todas as soluções S do domínio. Cada problema específico pode ser interpretado da seguinte forma: o problema é composto por um conjunto de elementos $E = \{e_1, \ldots, e_n\}$ onde uma solução viável X para o problema é um subconjunto de E. Seja $D \subseteq 2^E$, o domínio do problema, logo $X \in D$. Há uma função $f : D \to \mathbb{R}$ que atribui um custo para cada solução. O objetivo é encontrar uma solução $S^* \in D$ tal que $f(S^*) \leq f(S), \forall S \in D$.

A meta-heurística utilizada neste trabalho será o GRASP (*Greedy Randomized Adaptive Search Procedures*). GRASP é uma meta-heurística iterativa, desenvolvida por Feo e Resende [22], em que cada iteração consiste em duas fases: construção e busca local.

Na fase de construção, uma solução viável é criada progressivamente a partir do zero. Em cada iteração, apenas um elemento é adicionado a solução. A seleção do próximo elemento é guiado por uma função gulosa que mede o benefício que o mais recente elemento escolhido concede para a solução até então construída. O pseudo-código para construção de uma solução inicial gulosa é dada pelo Algoritmo 7.

Algoritmo 7: **AlgGuloso** (Algoritmo guloso para construção da solução inicial do GRASP.)

Entrada: Conjunto de elementos $E = \{e_1, e_2, \ldots, e_n\}$, função $c : E \to \mathbb{R}$ que valida o custo de todos os elementos candidatos

```
1  S ← ∅;
2  Inicie o conjunto de candidatos: C ← E;
3  Valida o custo c(e) para todo e ∈ C;
4  while C ≠ ∅ do
5  |  Selecione um elemento s ∈ C com o menor custo c(s);
6  |  Adicione s na solução corrente: S ← S ∪ {s} ;
7  |  Atualize o conjunto de candidatos C ;
8  |  Revalida o custo c(e) para todo e ∈ C;
9  end
10 Retorne S;
```

A técnica de busca local tenta melhorar a solução inicial S de maneira iterativa. A cada iteração, é gerado um conjunto de soluções vizinhas de S, denotado por $N(S)$. Este conjunto é gerado através de pequenas modificações na solução atual S. A melhor solução em $N(S)$ é escolhida para ser a solução atual, se esta melhorar o custo da função objetivo em relação a S.

Algoritmo 8: **BuscaLocal** (Algoritmo de busca local para problemas de minimização.)

Entrada: Solução inicial S

```
1 while S não é o ótimo local do
2 |  Procure S' ∈ N(S) com f(S') < f(S) ;
3 |  S ← S' ;
4 end
5 Retorne S;
```

Uma solução S^* é um ótimo local em relação a vizinhança N se $f(S^*) \leq f(S), \forall S \in$

$N(S^*)$. Métodos de busca local são baseados na exploração das soluções vizinhas, procurando melhorar a solução até que o ótimo local seja encontrado. O pseudo-código da busca local para um problema de minimização é dado pelo algoritmo 8.

O Algoritmo 9 ilustra o procedimento principal do GRASP. O número máximo de iterações é dado como entrada – definida pela constante $MaxIteracao$.

Algoritmo 9: **GRASP**

Entrada: constante $MaxIteracao$, problema a ser resolvido Z
1 $f^* \leftarrow \infty$;
2 **for** $i = 1$ *até* $MaxIteracao$ **do**
3 $S \leftarrow AlgGuloso(Z)$;
4 $S \leftarrow BuscaLocal(S)$;
5 **if** $f(S) < f^*$ **then**
6 $S^* \leftarrow S$;
7 $f^* \leftarrow f(S)$;
8 **end**
9 **end**
10 Retorne S;

Neste trabalho, propomos uma heurística baseada em GRASP para tratar o problema de particionamento em grafos. As fases componentes desse algoritmo, fase de construção e fase de busca local, serão descritas na próxima seção de maneira que sejam identificadas suas peculiaridades.

4.2 GRASP para o problema de Particionamento

Inicialmente, descreveremos como construir uma solução inicial viável para o problema de particionamento em grafo e, em seguida, como gerar os vizinhos para realizar a busca local.

4.2.1 Construção da Solução Inicial

Foram elaboradas várias estratégias para a fase de construção. Para cada estratégia, considera-se um grafo $G = (V, E, w)$ dado como entrada. Abaixo são descritas as estratégias:

Classificação gulosa utilizando centroides: Seja R um subconjunto com k vértices de V. Chamaremos cada vértice $r_i \in R$ de centro. Cada centro r_i corresponderá a um cluster C_i do particionamento $\mathcal{C}$, onde $1 < i \leq k$. A estratégia é atribuir cada vértice ao

cluster mais similar de forma que a soma dos pesos das arestas para outros clusteres seja baixa. Em outras palavras, um vértice $v \in V$ será atribuído ao cluster C_i (correspondente ao centro r_i) que tenha o menor valor para

$$q(v, C_i) = \frac{1}{w(v, r_i) + \epsilon} + \sum_{\{u \in C_j | i \neq j\}} w(v, u), \tag{4.1}$$

onde ϵ é uma constante, tal que $0 < \epsilon \leq 1$.

Para encontrar o conjunto R com os centros iniciais resolvemos o problema do *k-centro* ou *k-means* (veja a descrição desses problemas na Seção 1.3.2). Embora ambos sejam NP-Difíceis, existem algoritmos aproximados rápidos com bom fator de aproximação e heurísticas que retornam bons resultados.

Antes de buscar o conjunto R de centros de um grafo G (utilizando os algoritmos para os problemas *k-center* ou *k-means*), devemos construir uma representação do grafo no espaço d-dimensional (veja a Definição 12). Para construir a representação, é feito o processo de imersão no espaço. Esse processo constrói uma função $\varrho : V \to \mathbb{R}^d$ que atribui cada vértice a um ponto no espaço. Veja o Algoritmo 10 para a construção da imersão.

Inicialmente, o Algoritmo 10 cria uma matriz de transição de probabilidade M (veja a definição em 13) onde cada linha e coluna corresponderá a um dos n vértices do grafo de entrada G. Em seguida, calcula os $d + 1$ maiores autovalores $1, \lambda_2, ..., \lambda_{d+1}$ e seus respectivos autovetores $x^{(1)}, ..., x^{(d+1)}$ de M. Obtém uma matriz R com tamanho $(n \times (d + 1))$ formada pelos autovetores como colunas. Uma representação do vértice $u \in V$ no espaço corresponderá a um vetor linha $\varrho(u)$ de R.

Uma representação ϱ é boa quanto menor o valor de sua energia $energia(\varrho)$ (ver a definição da função de *energia* em (3.6)). O que faremos agora, é demonstrar que a representação construída pelo Algoritmo 10 é boa. Para isso, consideramos a matriz de Laplace $\mathcal{L}$ do grafo $G = (V, E, w)$ e a matriz R^* formada pelos autovetores $y_1, \ldots, y_n$ correspondente aos menores autovalores $\lambda_1^*, \ldots, \lambda_n^*$ de $\mathcal{L}$ como colunas. Pela Proposição 4 sabemos que

$$energia(\varrho) = Tr(R'\mathcal{L}R) = \sum_{i=1}^{d+1} y^{(i)'} \mathcal{L} y^{(i)}. \tag{4.2}$$

Pela definição de autovetor,

$$\mathcal{L}x = \lambda^* x \tag{4.3}$$

$$\Rightarrow x'\mathcal{L}x = \lambda^*. \tag{4.4}$$

Unindo as equações 3.6 e 4.4, obtemos $energia(\varrho) = \sum_{i=1}^{n} \lambda^{(i)*}$. Pela Proposição 5 sabemos que a energia mínima de uma representação de G em $\mathbb{R}^d$ é igual a $\sum_{i=2}^{d+1} \lambda^{(i)*}$. Portanto a representação R^* é boa.

Pela Proposição 8, se λ é autovalor de M (matriz de transição de probabilidade) então $1-\lambda$ é autovalor de $\mathcal{L}$. Com isso, sabendo que a soma dos menores autovalores de $\mathcal{L}$, $\sum_{i=2}^{d+1}(1-\lambda)$, define uma boa representação, então, a soma dos maiores autovalores de M, $\sum_{i=2}^{d+1}\lambda$, também define uma boa representação. Portanto, podemos utilizar uma matriz R formada pelos autovetores dos maiores autovalores de M como colunas para obter uma boa representação com energia mínima.

Algoritmo 10: *imersao* – Algoritmo para imersão no espaço

Entrada: $G = (V, E, w)$ com n vértices, d é o número de dimensões para imersão

1 Represente cada vértice de V como um inteiro de 1 até n ;
2 Construa a matriz de transição de probabilidade $M(G)$;
3 $(1, \lambda_2, ..., \lambda_{d+1}) \leftarrow d+1$ maiores autovalores de $M(G)$;
4 $x^{(1)}, ..., x^{(d+1)} \leftarrow$ autovetores de $M(G)$ associado com $\lambda_1, ..., \lambda_{d+1}$;
5 **forall** *i de* 1 *até* n **do**
6 $\quad \varrho(i) \leftarrow (x^{(1)}[i], ..., x^{(d+1)}[i])$;
7 **end**
8 retorna a representação ϱ ;

O algoritmo guloso utilizado pelo GRASP para a construção de uma solução inicial viável é descrito no Algoritmo 11. Inicialmente, o algoritmo cria uma representação dos vértices do grafo no espaço e busca o conjunto de centros da representação. Na descrição do Algoritmo 11, utilizamos o método *k-centros* para encontrar esse conjunto. Entretanto, nos testes computacionais, utilizamos também o algoritmo *k-means*. Cada centro será atribuído a um cluster diferente. Em seguida, escolhemos aleatoriamente um conjunto T' com um número x vértices. Cada vértice desse conjunto é atribuído ao cluster que minimiza a função q definida em 4.1. O algoritmo para quando todos os vértices forem atribuídos a algum dos k clusteres.

Classificação Gulosa Aleatória: Elaboramos uma outra forma mais simples de construir uma solução inicial viável. Seja R um subconjunto de V com k centros escolhidos aleatoriamente. Cada cluster será formado incrementalmente, escolhendo a aresta de maior peso que sai de um vértice pertencente a um cluster e unindo esse cluster ao vértice da aresta. Inicialmente, serão criados k clusteres $C_1, \ldots, C_k$ com apenas um vértice de R em cada um. A cada passo do algoritmo, escolhemos a aresta de maior peso que liga um vértice v ainda não associado a um vértice de algum cluster C_i, $1 \leq i \leq k$, e adicionamos v a C_i.

Soluções iniciais de heurísticas: Outra forma de construir soluções iniciais é aproveitar o particionamento retornado por outras heurísticas. Os algoritmos como o ICC e MCL, que implementam heurísticas para achar o particionamento, poderão ser usados

Algoritmo 11: Algoritmo Aleatório Guloso para construção de uma solução do problema de particionamento.

Entrada: grafo $G = (V, E, w)$, valor constante x, número k de clusteres

$\varrho \leftarrow imersao(G)$;
$R \leftarrow k - center(\varrho)$;
Atribua cada vértice de R a um diferente cluster C_i, onde $1 \leq i \leq k$;
$T \leftarrow V \backslash R$;
while $T \neq \emptyset$ **do**
    Escolha aleatoriamente um subconjunto $T' \subseteq T$ tal que $|T'| \leq x$;
    for $p \in T'$ **do**
        $\{p\} \cup argmin_{r_i \in R} q(C_i)$;
    end
    $T \leftarrow T - T'$;
end
Retorne o particionamento $\{C_1, \ldots, C_k\}$;

para gerar a solução inicial.

4.2.2 Busca Local

Nesta seção, descrevemos como gerar uma vizinhança para uma dada solução. A partir de uma solução inicial $\mathcal{C}$, uma solução vizinha $\mathcal{C}' \in N(\mathcal{C})$ é obtida pela operação de "movimento". Essa operação corresponde a mover alguns vértices de um cluster $C_i \in \mathcal{C}$ para outro cluster $C_j \in \mathcal{C}$.

Seja $G' = (C_i, E', w)$ o subgrafo induzido pelos vértices de C_i; aplicamos o corte mínimo em G' e obtemos os subconjuntos S e $\overline{S}$, separados pelo corte. Dentre S e $\overline{S}$ considere o de menor tamanho, digamos S. Um movimento corresponde a atribuição de todos os vértices de S para um outro cluster C_j. Para cada atribuição teremos uma uma nova solução que será uma solução vizinha de C.

Seja k o número de clusteres. Em cada iteração da busca local, serão feitos k cortes mínimos e $k(k-1)$ movimentos. As operações de movimentos tem complexidade $O(n)$, restando o corte mínimo como a operação mais onerosa.

Introduziremos o algoritmo de Stoer e Wagner [52] (algoritmo 13) para resolver o problema do corte mínimo.

A forma tradicional de resolver o problema de corte mínimo é por meio de sua relação com o problema de fluxo máximo. Ford e Fulkerson [24] mostraram a dualidade entre o fluxo máximo de s para t e o corte (s,t) mínimo (*minimun* (s,t)-*cut*). No problema de corte (s,t) mínimo, s e t são dois vértices, fonte e destino em um problema de fluxo, que devem ser separados por um corte. Encontrar um corte mínimo de um grafo pode ser feito

procurando o corte (s,t) mínimo entre um vértice fixo s e cada outro vértice $t \in V \backslash \{s\}$. O algoritmo mais rápido para se resolver o problema do corte mínimo usando fluxo tem complexidade de tempo $O(n^3)$ [1].

Neste trabalho, utilizamos um algoritmo determinístico simples e rápido, com complexidade de tempo $O(n^3)$ [52]. A ideia é obter o corte (s,t) mínimo de algum par de vértices s e t e aplicar o seguinte teorema para encontrar o corte mínimo do grafo.

Teorema 10 *Sejam s e t dois vértices de um grafo G. Seja $G \backslash \{s,t\}$ o grafo obtido pela união de s e t. Então o corte mínimo de G pode ser obtido pela escolha do menor entre o corte (s,t) mínimo de G e o corte mínimo de $G \backslash \{s,t\}$.*

O teorema é verdadeiro, pois, se existe um corte mínimo de G que separa s e t, então o corte (s,t) mínimo de G é o corte mínimo de G e, se não existe, então o corte mínimo está em $G \backslash \{s,t\}$.

Com isso, utilizaremos o procedimento *arbitrary-minimum* (s,t)*-cut* para construir um algoritmo que resolva o corte mínimo de G. Seja $w(A,y)$ a soma dos pesos de todas as arestas ligando y com vértices em A. O procedimento é descrito no algoritmo 12.

Algoritmo 12: *arbitrary-minimum* (s,t)*-cut* – Procedimento para encontrar o corte (s,t) mínimo, com os vértices s e t quaisquer.

Entrada: grafo $G = (V, E, w)$, vértice a
1 $A \leftarrow \{a\}$;
2 **while** $A \neq V$ **do**
3 $\quad A \leftarrow A \cup \{z\}$, onde $z = arg\max_{y \in V \backslash A} w(A,y)$;
4 **end**
5 Unir em G os dois últimos vértices adicionados em A ;
6 Retornar o valor de $w(A,z)$ onde z é o último vértice adicionado em A;

No procedimento *arbitrary-minimum* (s,t)*-cut*, um subconjunto A de vértices cresce, começando com um vértice arbitrário qualquer, até que A contenha todos os vértices de V. Em cada iteração do laço *while*, adicionamos a A o vértice cuja soma dos pesos de suas arestas que incidem em vértices de A seja máxima. No final do laço, os dois vértices que foram adicionados por último são unidos. A união é obtida contraindo-se esses dois vértices e substituindo cada conjunto de arestas múltiplas resultantes nesse novo vértice por uma aresta simples cujo o peso é a soma das arestas substituídas. No final do algoritmo é encontrado o corte (s,t) mínimo, onde s e t são os dois vértices adicionados por último [52].

O procedimento usado para o corte mínimo é descrito no algoritmo 13. A cada passo, o algoritmo encontra um par (s,t) com seu respectivo corte (s,t) mínimo e une s e t, usando o teorema 10 para encontrar o corte mínimo.

Algoritmo 13: *corte-minimo* – Procedimento para encontrar o corte mínimo

Entrada: grafo $G = (V, E, w)$
while $|V| \neq 2$ **do**
    corte-corrente $\leftarrow$ *minimum* (s,t)-*cut*(G, a);
    if corte-corrente $<$ menor-corte **then**
        menor-corte $\leftarrow$ *corte-corrente*;
    end
end
Seja S o subconjunto de vértices que se uniram em s;
Seja T o subconjunto de vértices que se uniram em t;
Retornar o corte (S, T);

Capítulo 5

Resultados Computacionais

Neste capítulo, apresentamos uma comparação prática entre os algoritmos escolhidos, analisando os resultados aplicando as funções *NCut* e *Performance*. Inicialmente, será descrita a forma com que as instâncias foram apropriadamente criadas para os experimentos. Em seguida, serão apresentados os resultados obtidos.

5.1 As Instâncias

Não foram encontradas referências que indiquem instâncias públicas disponíveis que possam ser usadas para testar algoritmos de clusterização em grafos. Entretanto, alguns trabalhos relacionados descrevem métodos para gerar instâncias aleatórias computacionalmente.

Seguindo o processo de geração aleatória de grafos descrita por Brandes et al [12, 11], criamos um gerador aleatório de grafos com n vértices e com clusteres de tamanho quase uniformes. Um gerador $P(n, s, v)$ determina uma partição $\mathcal{C} = (C_1, \ldots, C_k)$ de $\{1, \ldots, n\}$ com $|C_i|$ sendo uma variável aleatória com valor esperado s e desvio padrão $\frac{s}{v}$. O valor de k depende da escolha de n, s e v. Dada uma partição $\mathcal{C} = P(n, s, v)$ e probabilidades p_{in} e p_{out}, um grafo G é gerado. Os pesos das arestas do grafo G são definidos da seguinte forma: para uma aresta cujos vértices estão em um mesmo cluster C_i, o peso é um valor escolhido aleatoriamente no intervalo $[p_{in}, 1]$; para uma aresta cujos os vértices estão em clusteres diferentes, o peso é um valor escolhido aleatoriamente no intervalo $[0, p_{out}]$. Caso o grafo gerado não seja conexo, arestas adicionais com peso p_{out} são adicionadas.

Para nosso experimento, os parâmetros (n, s, v), p_{in} e p_{out} são definidos a seguir. Tomamos $v = 4$ e escolhemos s por meio de uma distribuição uniforme aleatória de $\{\frac{n}{l} | \log n \leq l \leq \sqrt{n}\}$. Experimentos são executados para $n = 500$. Todas as combinações de probabilidades de p_{in} e p_{out} que distam de 0.05 são consideradas desde que o valor de probabilidade p_{in} seja maior que o valor de p_{out} ($p_{in} > p_{out}$). Com isso, foram criadas 171

instâncias.

Como discutido na Seção 1.2, o que indica um "bom" particionamento é a alta densidade intraclusteres e esparsidade interclusteres. Quanto às instâncias geradas, para aquelas com probabilidade p_{in} alta e p_{out} baixa são "fáceis" de se encontrar um "bom" particionamento. Chamaremos essas instâncias "fáceis" de instâncias esparsas. Chamaremos de instâncias "densas" aquelas com probabilidade p_{out} alta.

O particionamento gerado por $P(n, s, v)$, p_{in}, p_{out} e o próprio grafo (representado por uma matriz de adjacência) são armazenados. Com isso, o particionamento gerado por $P(n, s, v)$ será comparado com outros particionamentos retornados pelos algoritmos.

5.2 Os Testes e seus Resultados

Todos os experimentos foram executados em um computador Intel XEON $2.40GHz$ com $1GB$ de RAM utilizando o sistema operacional LINUX versão 2.6. As implementações dos algoritmos foram escritas em Java (1.6.0). Foi utilizada a biblioteca *LAPACK* que provê rotinas para resolver problemas de álgebra linear (principalmente decomposição em autovetores e autovalores) [2].

Foram implementados os algoritmos ICC, GMC, MCL e GRASP com as funções objetivo *NCut* e *Performance*. Para os algoritmos ICC e GMC, utilizamos a biblioteca LAPACK para encontrar os autovalores e autovetores de uma matriz. Apesar dessa biblioteca ser escrita em Fortran90, foi feita uma interface para comunicação em Java. Outras bibliotecas escritas em Java para cálculo de autovalores e autovetores, como COLT e JAMA, não apresentaram bons desempenhos em relação ao tempo de processamento.

Todos os algoritmos foram implementados usando sofisticadas estruturas de dados. Entretanto, existem certas limitações, especialmente em relação à medida do tempo de execução em Java. Apesar das implementações não serem especialmente otimizadas em relação ao tempo de execução, algumas tendências puderam ser identificadas. Estas tendências são discutidas ao longo desta seção.

Os parâmetros do algoritmo MCL foram definidos para $e = 2$ e $r = 2$, como sugerido nos trabalhos [25, 12]. No algoritmo ICC, realizamos testes com os valores do parâmetro α de 0.2, 0.4 (como sugeridos em [12]) e 0.6. No algoritmo GMC, os melhores resultados foram encontrados para o parâmetro $d = 5$.

O processo realizado pelo ICC particiona o grafo somente se o valor do corte de condutância for menor que α. Para instâncias densas, o ICC com $\alpha = 0.2, 0.4$ não encontrou particionamentos com mais de um cluster. Com $\alpha = 0.2$, os "bons" particionamentos foram encontrados apenas para instâncias com $p_{out} = 0.05, 0.1$. Para $\alpha = 0.4$, "bons" particionamentos foram encontrados com $p_{out} \leq 0.3$. Com $\alpha = 0.6$, o algoritmo ICC encontrou, em todas as instâncias, particionamentos com mais de um cluster. Entretanto,

em cada particionamento o número de clusteres extrapolou o valor k esperado.

Para que o algoritmo ICC pudesse encontrar o particionamento em todas as instâncias (e que o tamanho do particionamento não extrapolasse a quantidade k de clusteres), iniciamos os testes com $\alpha = 0.4$. Caso o algoritmo ICC tenha encontrado um particionamento com apenas um cluster (particionamento trivial), incrementamos α em 0.01 e aplicamos o ICC novamente. Sucessivas execuções do ICC com α incrementado foram feitas até que o resultado fosse um particionamento com mais de um cluster.

O algoritmo MCL também não encontrou particionamento (com mais de um cluster) para as instâncias densas. As instâncias em que o MCL encontrou particionamento foram aquelas com p_{out} menor que 0.2, quem correspondem a 50 das 171 instâncias criadas. Não é claro como incorporar restrições no MCL para que este gere um particionamento com um número mínimo de clusteres. Mesmo assim, utilizamos os resultados do MCL para as comparações.

Modificamos o algoritmo GMC para que ele gerasse particionamentos cujo número de clusteres estivesse entre um limite inferior e um limite superior. No processo realizado pelo algoritmo GMC modificado, após a construção da árvore geradora mínima T (ver algoritmo 5), o particionamento é calculado somente se o número de componentes desconexos da floresta induzida $F(T, \tau)$ for maior que 2 e menor que 40. Usamos o limite inferior 2 para que nunca fosse gerado o particionamento trivial e o limite superior 40 para que o algoritmo não gerasse particionamentos que extrapolam a quantidade de clusteres esperados.

No Capítulo 2, listamos várias funções de avaliação que podem indicar um "bom" particionamento. Entre as que foram listadas, escolhemos as funções de *NCut* e *Performance* para os testes. A condutância não foi escolhida como função de avaliação, porque, no algoritmo ICC, o valor de condutância de um particionamento cresce com o aumento do parâmetro α, tornando os resultados diretamente dependentes do parâmetro de entrada. A função de *coverage* também não foi utilizada devido a restrição no número mínimo de clusteres, requerendo $k > 2$.

A constante $MaxIteracao$, que é o número de iterações que o GRASP faz a busca local, foi definida nos testes com o valor 15. A busca local pode ser interrompida se, após várias buscas nas soluções vizinhas, não ocorrer uma melhora significativa na solução. Isso foi feito na implementação para diminuir o tempo de convergência. Foram feitos vários testes para encontrar qual a porcentagem de melhora considerada significativa. Notamos que, se em 10 iterações da busca local, em cada iteração a solução vizinha melhora menos de $0,5\%$ a solução corrente, a convergência para o ótimo local será bastante lenta.

Também com o objetivo de melhorar a convergência da busca local, utilizamos uma abordagem baseada em corte mínimo para geração de vizinhos. Outras abordagens também foram testadas, entretanto, não apresentaram bons desempenhos na convergência

da busca local. Temos os seguintes exemplos de abordagens que foram testadas: selecionar aleatoriamente um vértice v e em seguida atribuí-lo para os outros clusteres; selecionar um conjunto de vértices U adjacentes de um vértice escolhido aleatoriamente v e que, ao mesmo tempo, o peso das arestas que ligam vértices de U ao vértice v seja alta; escolher aleatoriamente um vértice v e um conjunto de vértices U tal que, para um vértice $u \in U$, o peso da aresta (v, u) é mínimo e cada vértice u seja de clusteres diferentes. Apesar de algumas dessas técnicas testadas serem mais baratas do que escolher vértices baseados no corte mínimo, a qualidade da vizinhança gerada não foi significantemente melhor.

Foram aplicadas as funções de avaliação *NCut* e *Performance* sobre o particionamento gerador, $P(n, s, v)$. Ressaltamos que não há garantia de que o valor dessas funções, aplicadas sobre o particionamento gerador, seja o ótimo global. Todavia, os particionamentos resultantes dos algoritmos testados não revelaram valores melhores do que aqueles obtidos pelo particionamento gerador.

As funções de avaliação *NCut* e *Performance* foram aplicadas sobre os particionamentos resultantes dos algoritmos. Lembramos que um "bom" particionamento tem um valor baixo da função *NCut* e um valor alto de *Performance*. Os resultados são apresentados para cada função de avaliação. Para mostrar de forma resumida os resultados dos testes sobre as 171 instâncias, fizemos um agrupamento dos resultados. São dois grupos: um em relação ao valor de p_{out} e outro em relação a p_{in}, lembrando que cada par de valores p_{in} e p_{out} corresponde a uma instância. No grupo em relação a p_{in}, para cada valor fixo de p_{in} é mostrada a média dos resultados para todos os valores de p_{out}. E para o grupo em relação a p_{out}, para cada valor fixo de p_{out} é mostrada a média dos resultados para todos os valores de p_{in}.

Além disso, para cada grupo (p_{in} ou p_{out}) foram construídos quatro gráficos, dois para apresentarem os valores das funções de avaliação e outros dois para os tempos de processamento correspondente às funções de avaliação. Em cada gráfico, estão os valores das funções de avaliação aplicados sobre os resultados obtidos pelos algoritmos escolhidos para os testes. E ainda, são mostrados nos gráficos os valores das funções de avaliação aplicadas sobre os particionamentos utilizados para criação das instâncias.

Os itens que são apresentados nos gráficos são os seguintes:

- ***grasp (icc)***: Corresponde aos resultados da metaheurística GRASP. Para gerar a solução inicial utilizamos: soluções dos algoritmos ICC, MCL, *k-centro* e *k-means*; classificação gulosa utilizando centroides e classificação gulosa aleatória (veja na Seção 4.2.1 sobre o processo de geração inicial).

- ***grasp***: Corresponde aos resultados da metaheurística GRASP sem a utilização do resultado do algoritmo ICC para solução inicial. Para gerar a solução inicial utilizamos: soluções dos algoritmos MCL, *k-center* e *k-means*; classificação gulosa

utilizando centroides e classificação gulosa aleatória (veja na Seção 4.2.1 sobre o processo de geração inicial);

- ***icc***: Algoritmo ICC (*Iterative Conductance Cutting* – ver Seção 3.3);
- ***mcl***: Algoritmo MCL (*Markov Clustering Algorithm* – ver Seção 3.2);
- ***gmc***: Algoritmo GMC (*Geometric MST Clustering* – ver Seção 3.4).
- ***result***: O item *result* corresponde aos particionamentos gerados por $P(n, s, v)$ e utilizados nas construções das instâncias de testes. Aplicamos as funções de avaliação sobre o particionamento gerado por $P(n, s, v)$ e apresentamos os valores no gráfico junto aos valores obtidos pelos resultados dos algoritmos.

É importante salientar que o particionamento $P(n, s, v)$ utilizado na construção de uma instância de teste não é necessariamente o particionamento ótimo para as funções de $NCut$ e *Performance*. Entretanto, pela forma com que as instâncias foram construídas, os particionamentos gerados por $P(n, s, v)$ representam os "bons" clusteres.

Como $p_{out} < p_{in}$, para o primeiro valor de $p_{out} = 0.05$, temos o grupo de 18 instâncias com valores de p_{in} em $\{0.1 \ldots 0.95\}$, para $p_{out} = 0.1$, temos 17 instâncias com valores de p_{in} em $\{0.15 \ldots 0.95\}$ e assim sucessivamente, até $p_{out} = 0.9$ e $p_{in} = 0.95$.

A Figura 5.1 mostra os resultados obtidos pelos algoritmos considerando a função de $NCut$ e os resultados agrupados pelo valor de probabilidade p_{out} (o eixo x representa p_{out}). Os resultados obtidos pelos algoritmos ***grasp*** e ***grasp (icc)*** são praticamente idênticos. Isso mostra que, mesmo sem uma solução inicial dada pelo ICC, a busca local ainda converge de forma eficiente. Para instâncias com p_{out} menor que 1.5, os algoritmos ***grasp***, ***grasp (icc)*** e ***mcl*** encontram particionamentos com o número de clusteres menor do que aquele esperado. Consequentemente, os resultados foram melhores do que aqueles definidos por ***result***. Note que o algoritmo ***mcl*** não encontrou nas instâncias densas (com $p_{out} > 0.2$) particionamentos com mais de dois clusteres. Por outro lado, o algoritmo ***gmc*** encontrou bons particionamentos para todas as instâncias. O algoritmo ***icc*** encontrou bons particionamentos apenas para instâncias com grafos esparsos. Se o parâmetro α do algoritmo ***icc*** fosse diminuído, os cortes também diminuiriam, o número de clustereres seria menor e, consequentemente, poderíamos encontrar os mesmos resultados obtidos pelos algoritmos ***grasp***, ***grasp (icc)*** e ***mcl***. Entretanto, o incremento de α impediria que o algoritmo encontrasse cortes nas instâncias densas.

A Figura 5.2 apresenta o tempo de execução dos algoritmos para a função de $NCut$. Podemos notar que os algoritmos ***mcl*** e ***gmc*** apresentam poucas variações no tempo de execução. Temos que o tempo de execução do ***gmc*** é significantemente menor do que qualquer outro algoritmo testado. Esse baixo tempo do ***gmc*** é obtido graças ao limiar

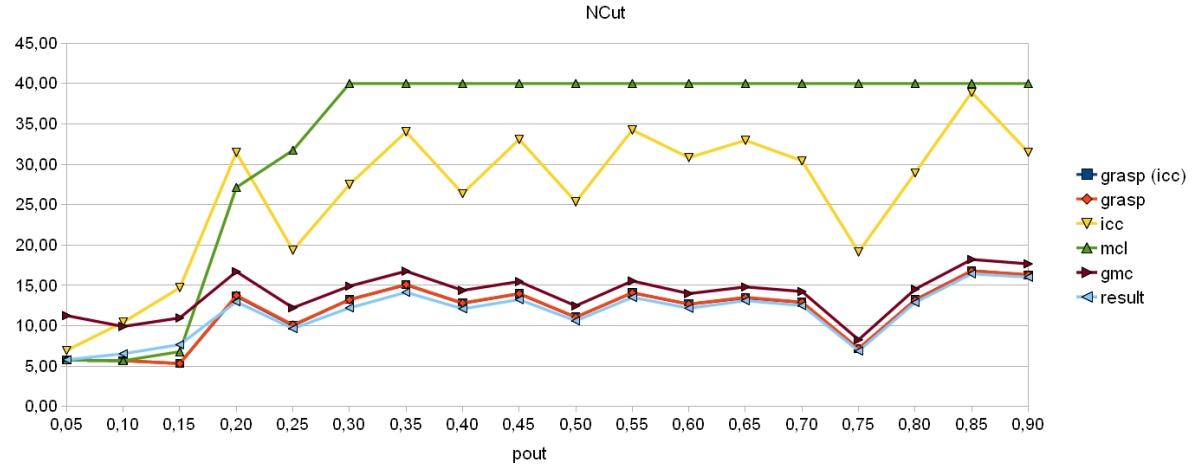


Figura 5.1: Comparação dos particionamentos obtidos com o particionamento gerador (Avaliados por $NCut$ e agrupados por p_{out}).

imposto para o número mínimo e máximo de componentes desconexos que serão aceitos como um particionamento criado a partir da remoção de uma aresta da árvore geradora mínima. O algoritmo ***icc*** apresenta uma piora no tempo a medida que o grafo se torna mais denso, que é justificado pelos sucessivos incrementos no parâmetro α. Os algoritmos ***grasp*** e ***grasp (icc)*** possuem tempo bastante variável, porque a busca local depende da qualidade da solução inicial para chegar a um ótimo local.

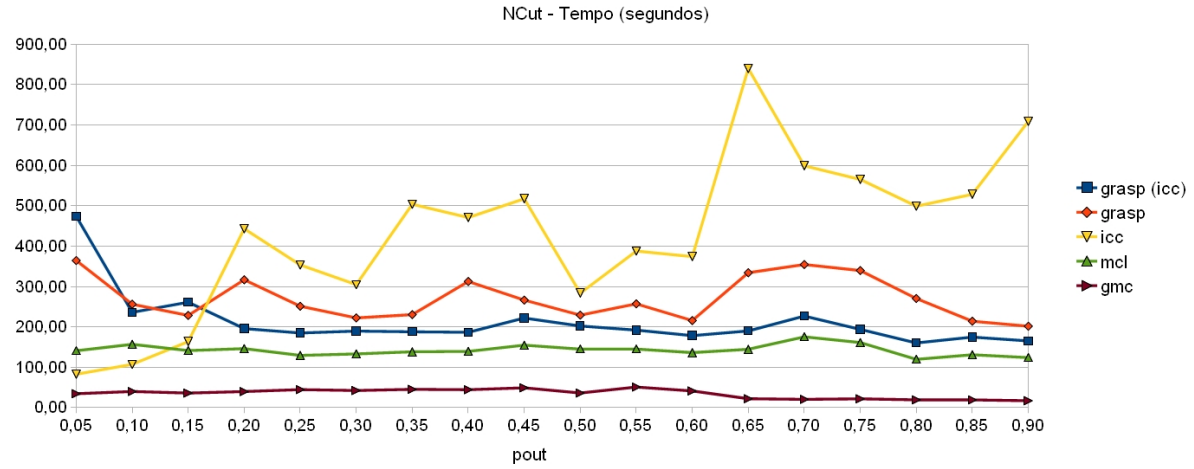


Figura 5.2: Comparação dos tempos de execução dos algoritmos (GRASP com o $NCut$ como função objetivo e os dados agrupados por p_{out}).

A Figura 5.3 apresenta os resultados obtidos pelos algoritmos considerando a função

de *Performance* e agrupamento pelo valor de p_{out}. Os algoritmos ***grasp***, ***grasp (icc)*** e ***icc*** apresentaram bons resultados para todas as instâncias. Os resultados obtidos por esses algoritmos se compararam com os particionamentos de ***result***. Como esperado, o algoritmo ***mcl*** não conseguiu encontrar um particionamento não trivial para todas as instâncias densas (com $p_{out} > 0.15$). O algoritmo ***gmc*** encontrou particionamento para todas as instâncias, mas com o valor de *Performance* menor do que o do particionamento gerador *result*.

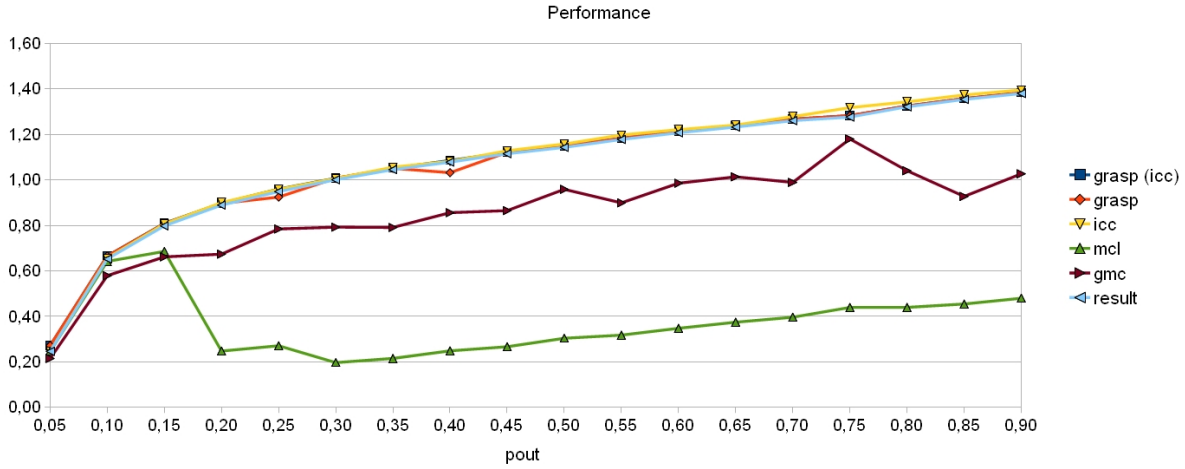


Figura 5.3: Comparação dos particionamentos obtidos com o particionamento gerador (Avaliados por *Performance* e agrupados por p_{out}).

A Figura 5.4 apresenta o tempo de execução dos algoritmos para a função de *Performance*. Na figura, notamos que os algoritmos ***mcl*** e ***gmc*** apresentaram tempo praticamente constante, enquanto os outros algoritmos apresentaram tempo com grande variação. Esta variação nos algoritmos ***grasp*** e ***grasp (icc)*** se deve ao tempo de convergência de uma solução inicial gerada aleatoriamente para seu ótimo local. Dependendo da qualidade da solução inicial e independente da instância de teste, os algoritmos podem convergir de forma lenta.

As Figuras 5.5 e 5.7 apresentam os resultados obtidos pelos algoritmos para as funções *NCut* e *Performance* agrupados pelo valor de p_{in}. Definimos os valores de p_{in} sempre maiores que p_{out}. Assim, visualizando os resultados em relação a p_{in}, podemos perceber a qualidade dos resultados para instâncias esparsas. Veja que os particionamentos obtidos por todos os algoritmos para as instâncias com $p_{in} \leq 0.15$ possuem valores para ambas funções *NCut* e *Performance* quase iguais àquele esperado em ***result***.

As Figuras 5.6 e 5.8 apresentam o tempo de execução dos algoritmos para as funções *NCut* e *Performance* agrupados pelo valor de p_{in}. Como esperado, os algoritmos ***icc***,

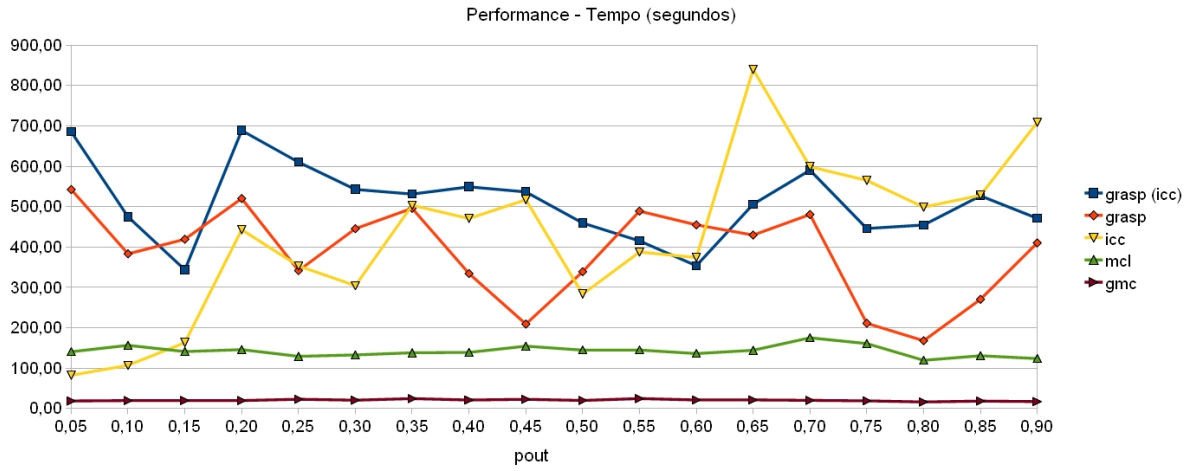


Figura 5.4: Comparação dos tempos de execução dos algoritmos (GRASP com o *Performance* como função objetivo e os dados agrupados por p_{out}).

gmc e ***mcl*** foram significantemente mais rápidos do que ***grasp*** e ***grasp (icc)*** para instâncias esparsas (com $p_{in} < 0.15$).

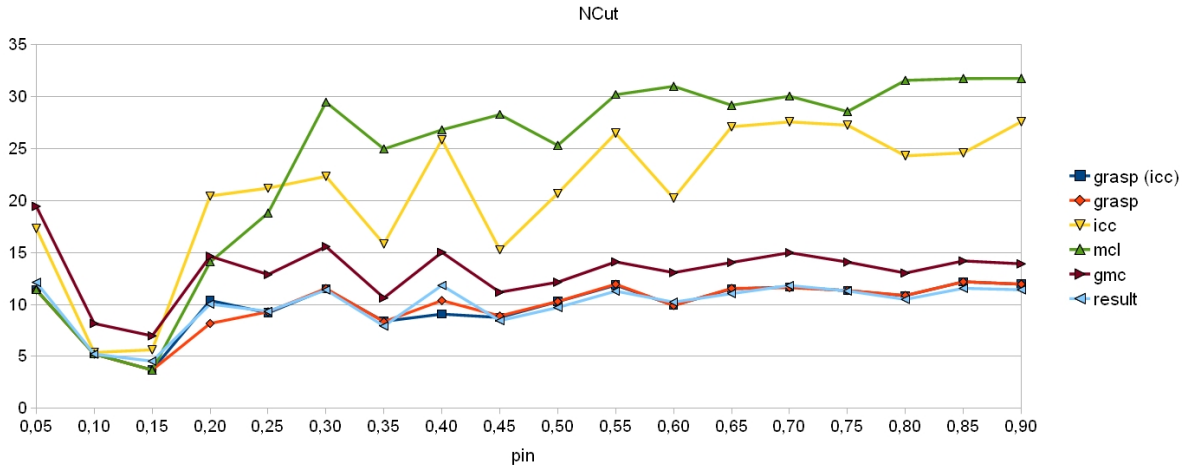


Figura 5.5: Comparação dos particionamentos obtidos com o particionamento gerador (Avaliados por $NCut$ e agrupados por p_{in}).

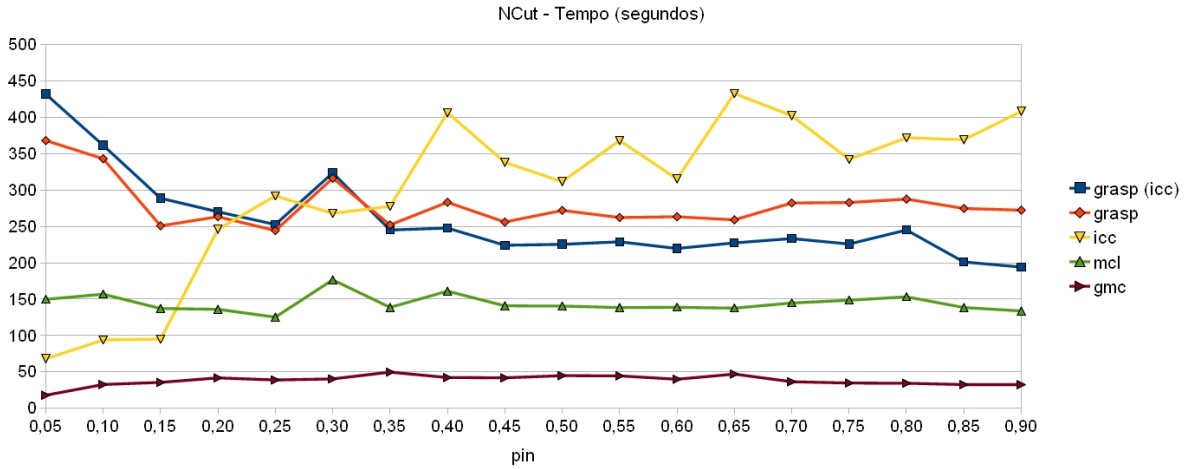


Figura 5.6: Comparação dos tempos de execução dos algoritmos (GRASP com o $NCut$ como função objetivo e os dados agrupados por p_{in}).

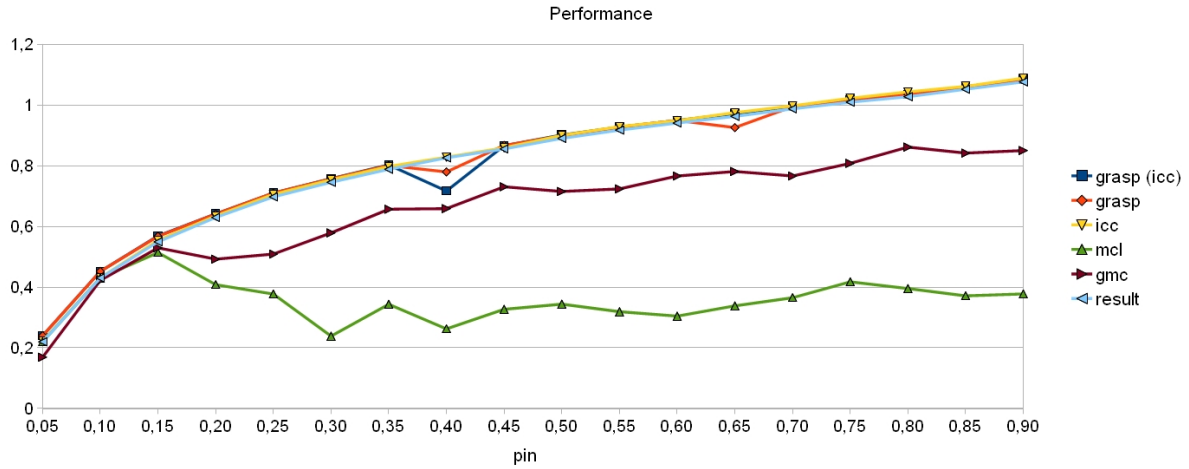


Figura 5.7: Comparação dos particionamentos obtidos com o particionamento gerador (Avaliados por *Performance* e agrupados por p_{in}).

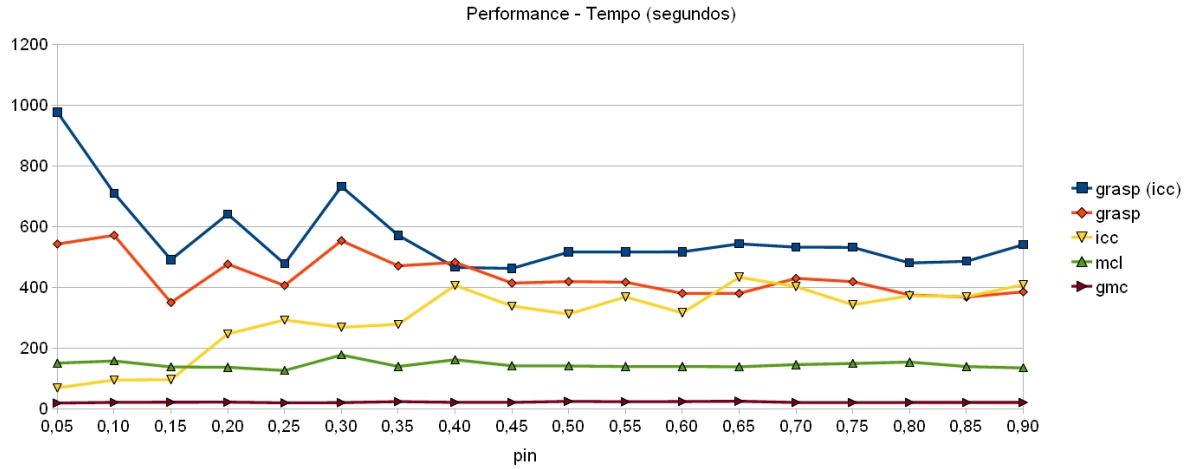


Figura 5.8: Comparação dos tempos de execução dos algoritmos (GRASP com o *Performance* como função objetivo e os dados agrupados por p_{in}).

Capítulo 6

Considerações Finais

Depois dos estudos feitos para a construção desta dissertação, verificamos que os algoritmos para resolver o problema de particionamento ainda merecem mais atenção da comunidade científica. Principalmente no que se refere à qualidade do particionamento. Além disso, seria promissor fazer estudos teóricos que estabelecesse a correlação entre o particionamento obtido por funções de otimização e aqueles obtidos por heurísticas.

Motivados pela má qualidade dos particionamentos obtidos em grafos densos, desenvolvemos uma nova estratégia, utilizando a metaheurística GRASP, para o problema de particionamento em grafos. Apesar do tempo de execução para encontrar um particionamento utilizando o GRASP ser superior à maioria dos algoritmos aqui estudados, obtivemos uma significativa melhora na qualidade dos resultados.

A teoria envolvendo clusterização espectral se mostrou bastante útil para o problema estudado. Graças à técnica de imersão do grafo no espaço, fomos capazes de aplicar algoritmos de clusterização, como o *k-means* e *k-center*, em dados representados por grafos. Além disso, a simplicidade dos algoritmos de clusterização espectral motiva a sua utilização em aplicações práticas. Os resultados práticos obtidos por esses algoritmos foram promissores, em especial, o algoritmo GMC. O algoritmo ICC, apesar de não retornar bons resultados para todas as instâncias com vários níveis de densidade, pode ser utilizado em aplicações práticas bastando apenas a definição do correto parâmetro α. Além de termos realizado um estudo prático com implementação dos algoritmos selecionados, também fizemos um estudo teórico com a preocupação de entender como o uso do espectro da matriz de um grafo poderia resultar em "bons" particionamentos.

Não obtivemos sucesso na busca por instâncias reais. A solução encontrada foi gerar instâncias computacionalmente. Sabemos que as instâncias utilizadas nos testes podem não refletir a realidade, mas elas serviram para o nosso propósito de comparar a eficácia dos vários algoritmos estudados.

6.0.1 Trabalhos Futuros

Apresentamos a seguir alguns pontos de pesquisas que consideramos promissores:

- Buscar instâncias práticas e interessantes para o problema, para que possamos analisar melhor o funcionamento do algoritmo em instâncias reais.
- Utilização da imersão do grafo no espaço dimensional para resolver problemas correlatos, como por exemplo o problema de classificação.
- Investigar a técnica de clusterização espectral afim de obter resultados teóricos. Principalmente a busca por garantias de aproximação para os algoritmos.

Apêndice A

Análise do Algoritmo ICC

Apresentamos nessa sessão a demonstração feita por R. Kannan, S. Vempala e A. Vetta [34] da garantia de aproximação dada pelo algoritmo ICC. Esta é a primeira análise de pior caso para o problema de clusterização utilizando técnica de clusterização espectral.

Outro ponto importante dessa análise é a forma com que a qualidade do particionamento é medido. Foi definido uma métrica bi-criteriosa chamada (α, ϵ)-clustering (veja a definição 14) que relacionará a densidade interna de um conjunto de vértice, chamada de densidade intra-cluster, versus a densidade inter-cluster, medida pelas arestas que saem do subconjunto de vértices. Veja na sessão 1.2 uma discussão sobre métricas para determinar "bons" particionamentos.

Definição 14 ((α, ϵ)-clustering) *Seja $G = (V, E)$ um grafo com peso nas arestas, chamaremos uma partição $\{C_1, C_2, ..., C_l\}$ de V uma (α, ϵ)-clustering se:*

1. *A condutância de cada C_i é pelo menos α;*
2. *O peso total das arestas inter-clusteres é no máximo uma fração ϵ do peso total das arestas do grafo - o valor de ϵ é*

$$\epsilon = \frac{W}{\sum_{(u,v)\in E} w(u,v)}, \tag{A.1}$$

onde W é o peso das arestas inter-clusteres.

Logo, pela definição 14, obtemos uma métrica bi-criteriosa de qualidade do particionamento. Associado com essa métrica teremos o seguinte problema de otimização:

Problema 2 *Dado α, encontre um (α, ϵ)-clustering que minimize ϵ (alternativamente, dado ϵ, encontre um (α, ϵ)-clustering que maximize α).*

O problema 2 é NP-Difícil. Para ver isso, considere como objetivo maximizar α enquanto o valor de ϵ tende a zero. Esse problema é equivalente a encontrar o corte de condutância mínima de um grafo, que é, por sua vez, um problema NP-Difícil [26].

A.1 Garantia de Aproximação do Algoritmo ICC

Para fins de análise, utilizaremos uma versão recursiva do ICC (chamada de *Recursive Conductance Cutting* – RCC) e o problema a ser tratado é aquele definido em 2. O algoritmo RCC trabalha da seguinte forma: dado um grafo $G = (V, E)$, $n = |V|$, o algoritmo procura um corte $c(S, \overline{S})$ de condutância mínima. Então prossegue com os cortes recursivamente sobre os subgrafos induzidos de S e $\overline{S}$. A descrição do algoritmo segue abaixo.

Algoritmo 14: RCC – *Recursive Conductance Cutting*

Input: $G = (V, E)$

1 Encontre o corte $c(S, \overline{S})$ de condutância mínima aproximado em G;

2 Recursivamente aplique os cortes nos subgrafos induzidos $G[S]$ e $G[\overline{S}]$;

É sabido que o corte de condutância mínimo de um grafo é um problema NP-Difícil, logo, é necessário utilizar uma solução rápida e que dê um corte aproximado. Pode-se utilizar a mesma solução proposta no algoritmo ICC – utilizar o segundo autovetor da matriz de transição.

O fator de aproximação que será apresentado independe de qual aproximação para o corte de condutância mínima será utilizada. Isso porque a análise considerará um algoritmo A qualquer que dê um corte de condutância mínimo aproximado.

Seja A um algoritmo de aproximação que produz um corte de condutância no máximo Kx^v se a condutância mínima é x, onde K é independente de x (K pode ser em função do número de vértices) e v uma constante fixa entre 0 e 1. O seguinte teorema provê uma garantia de aproximação para o algoritmo 14 usando A como sub-rotina.

Teorema 11 *Se $G = (V, E)$ tem um (α, ϵ)-clustering, então o algoritmo 14 irá encontrar um particionamento com qualidade*

$$\left(\left(\frac{\alpha}{6K \log \frac{n}{\epsilon}} \right)^{\frac{1}{v}}, (12K + 2)\epsilon^v \log \frac{n}{\epsilon} \right).$$

Prova: Seja os cortes $c(S_1, T_1), c(S_2, T_2), \ldots,$ gerados pelo algoritmo 14 onde convencionamos que S_j é o menor lado (i.e., $a(S_j) \leq a(T_j)$). Seja $C_1, C_2, \ldots, C_l$ um (α, ϵ)-clustering. Assumimos também que os cortes recursivos serão aplicados apenas se a condutância do grafo induzido pelo subconjunto de vértices for menor que $\alpha^* = \frac{\alpha}{6 \log \frac{n}{\epsilon}}$. Uma outra modificação que será feita no algoritmo, apenas para auxiliar na análise, é definir que, em qualquer momento da execução, se for encontrado um cluster $\{C_t\}$ com $a(C_t) < \frac{\epsilon}{n} a(V)$ então cada vértice de C_t será um cluster. A condutância de cada vértice será 1.

O algoritmo A produz um corte de condutância mínimo aproximado de no máximo kx^v. Esse algoritmo realizará o corte somente se $kx^v < \alpha^*$, como foi definido pela condição de parada α^*. Dessa forma, cada subconjunto de vértice (chamaremos de cluster) do particionamento resultante – particionamento retornado pelo algoritmo 14 – terá o corte aproximado de condutância mínimo maior ou igual a α^*, ou seja

$$Kx^v \geq \alpha^*,$$

logo, isolando a variável x (lembrando que x é o valor corte de condutância mínima) teremos que cada cluster terá condutância pelo menos

$$x \geq \left(\frac{\alpha^*}{K}\right)^{\frac{1}{v}} = \left(\frac{\alpha}{6K\log\frac{n}{\epsilon}}\right)^{\frac{1}{v}}.$$

Resta agora limitar o peso das arestas inter-cluster. Para isso dividiremos os cortes em dois grupos. O primeiro grupo, H, serão os cortes com "alta" condutância dentro dos clusteres. O outro grupo serão os cortes restantes. Utilizaremos a notação $w(S_j, T_j) = \sum_{u\in S_j, v\in T_j} w(u,v)$ para denotar o custo do corte $c(S_j, T_j)$. Além disso, denotaremos por $w_I(S_j, T_j)$ a soma do peso das arestas intra-cluster do corte (S_j, T_j), isto é, $w_I(S_j, T_j) = \sum_{i=1}^{l} w(S_j \cap C_i, T_j \cap C_i)$. O grupo H será formado da seguinte forma:

$$H = \left\{ j : w_I(S_j, T_j) \geq 2\alpha^* \sum_{i=1}^{l} min(a(S_j \cap C_i), a(T_j \cap C_i)) \right\}.$$

Agora limitaremos o custo do grupo de "alta" condutância. Sabemos, pela condição de parada α^* que

$$\alpha^* \geq \frac{w(S_j, T_j)}{a(S_j)},$$

assim, chegaremos nas seguintes desigualdades

$$\alpha^* a(S_j) \geq w(S_j, T_j) \geq w_I(S_j, T_j) \geq 2\alpha^* \sum_{i=1}^{l} min(a(S_j \cap C_i), a(T_j \cap C_i)).$$

Consequentemente, observamos que

$$\sum_{i=1}^{l} min(a(S_j \cap C_i), a(T_j \cap C_i)) \leq \frac{1}{2} a(S_j) \tag{A.2}$$

Dados os cortes realizados pelo algoritmo RCC, formando o conjunto $\{c(S_j, T_j)\}$, e também o conjunto de clusteres ótimo, $\{C_i\}$, definiremos um novo conjunto de cortes $\{c(S'_j, T'_j)\}$ como segue. Para cada $j \in H$, chamaremos de *corte que evita cluster* um corte $c(S'_j, T'_j)$ em $S_j \cup T_j$ da seguinte maneira:

para cada i, $1 \leq i \leq l$ faça

$$\text{se } a\left(S_j \cap C_i\right) \geq a\left(T_j \cap C_i\right) \rightarrow S'_j \text{ recebe } \left(S_j \cup T_j\right) \cap C_i$$
$$\text{se } a\left(S_j \cap C_i\right) < a\left(T_j \cap C_i\right) \rightarrow T'_j \text{ recebe } \left(S_j \cup T_j\right) \cap C_i$$

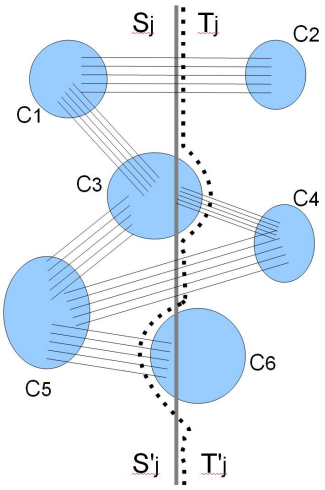


Figura A.1: Ilustração do corte $c(S_j, T_j)$ e do corte que evita cluster $c(S'_j, T'_j)$. Cada circunferência é um cluster C_i.

Um exemplo desses cortes é representado pela figura A.1, onde o corte original – corte feito pelo algoritmo RCC – é representado pela linha contínua e o corte que evita cluster é representado pela linha tracejada. Intuitivamente, podemos interpretar os cortes definidos por $c(S'_j, T'_j)$ como aqueles que retornariam o particionamento ótimo, $\{C_i\}$. Consequentemente, o próximo passo dessa demonstração é encontrar um limite superior do valor do corte dado por $w(S_j, T_j)$ em relação a $w(S'_j, T'_j)$.

Perceba que,

$$|a(S_j) - a(S'_j)| = |a(T_j) - a(T'_j)| = \sum_{i=1}^{l} min(a(S_j \cap C_i), a(T_j \cap C_i)).$$

Utilizando a desigualdade (A.2), teremos:

$$|a(S_j) - a(S'_j)| = |a(T_j) - a(T'_j)| = \sum_{i=1}^{l} min(a(S_j \cap C_i), a(T_j \cap C_i)) \leq \frac{1}{2} a(S_j).$$

Assim, tendo verdade que $|a(S_j) - a(S'_j)| = |a(T_j) - a(T'_j)| \leq \frac{1}{2} a(S_j)$ e $a(S_j) \leq a(T_j)$ teremos que $min(a(S'_j), a(T'_j)) \geq \frac{1}{2} a(S_j)$. Agora utilizaremos a garantia de aproximação

do corte de condutância dada pelo algoritmo A para obter um limite superior no corte $w(S_j, T_j)$ em termos de $w(S'_j, T'_j)$:

$$\frac{w(S_j, T_j)}{a(S_j)} \leq K \left(\frac{w(S'_j, T'_j)}{min(a(S'_j), a(T'_j))} \right)^v \leq K \left(\frac{2w(S'_j, T'_j)}{a(S_j)} \right)^v,$$

que implica em

$$w(S_j, T_j) \leq K(2w(S'_j, T'_j))^v a(S_j)^{1-v}. \tag{A.3}$$

Seja $P(S)$ denotar o conjunto de arestas inter-cluster incidente a algum vértice em S, para qualquer $S \in V$, e seja também, definido como $w(P(S))$, a soma dos pesos das arestas do conjunto $P(S)$. Então, $w(S'_j, T'_j) \leq w(P(S'_j))$, desde que cada aresta do corte (S'_j, T'_j) é uma aresta inter-cluster. Assim, podemos reescrever a desigualdade (A.3) da seguinte forma

$$w(S_j, T_j) \leq K(2w(P(S'_j)))^v a(S_j)^{1-v}.$$

Os lemas apresentados a seguir são importantes para o prosseguimento na demonstração.

Lema 2 *Para cada vértice $u \in V$, existe no máximo $\log \frac{n}{\epsilon}$ valores de j tal que u pertença a S_j. Além disso, existe no máximo $2 \log \frac{n}{\epsilon}$ valores de j tal que u pertença a S'_j.*

Prova: Lembrando que $a(S_j) < a(T_j)$, para qualquer j, nós temos que (para $k > j$) $a(S_k) \leq \frac{1}{2} a(S_k \cup T_k) \leq \frac{1}{2} a(S_j)$. Então $a(S_j)$ reduz por um fator de 2. Note que o valor máximo de $a(S_j)$ é $a(V)$, considerando todos os vértices V, e o valor mínimo definido por convenção é $\frac{\epsilon}{n} a(V)$. Dessa forma, o valor x de divisões feitas no conjunto S_j pode ser calculado como

$$\frac{a(V)}{2^x} = \frac{\epsilon}{n} a(V).$$

Isolando a variável x teremos que $x = \log \frac{n}{\epsilon}$, o que finaliza a prova da primeira sentença do lema.

Para auxiliar na provar da segunda sentença definiremos dois conjuntos de índices. Seja os conjuntos

$$I_u = \{j : u \in S_j\} \qquad J_u = \{j : u \in S'_j \backslash S_j\}.$$

Sabemos que $|I_u| \leq \log \frac{n}{\epsilon}$. Agora, suponha que $j, k \in J_u; j < k$. Suponha também que $u \in C_i$. Então $u \in T_j \cap C_i$. Posteriormente, teremos o subconjunto T_j, com $j < \log \frac{n}{\epsilon}$, particionado pelo corte $c(S_k, T_k)$ e, desde que $u \in S'_k \backslash S_k$, teremos que $a(T_k \cap C_i) \leq a(S_k \cap C_i)$. Assim $a(T_k \cap C_i) \leq \frac{1}{2} a(S_k \cup T_k) \leq \frac{1}{2} a(T_j \cap C_i)$. Logo $a(T_j \cap C_i)$ reduz por um fator de 2. Então, $|J_u| \leq \log \frac{n}{\epsilon}$. Portanto, para $u \in S'_j$, existe no máximo $\log \frac{n}{\epsilon}$ valores de $j \in I_u$ mais $\log \frac{n}{\epsilon}$ valores de $k \in J_u$, que dá o valor de $2 \log \frac{n}{\epsilon}$. ■

Lema 3 *Seja $G = (V, E)$ um grafo com peso nas arestas e W o peso das arestas inter-clusteres do particionamento ótimo do problema 2, então $W = \frac{\epsilon}{2}a(V)$.*

Prova: Note que $a(V) = 2\sum_{(u,v)\in E} w(u,v)$ e que $\epsilon = \frac{W}{\sum_{(u,v)\in E} w(u,v)}$, consequentemente, com a simples substituição, verificamos que $W = \frac{\epsilon}{2}a(V)$. ■

Utilizando o lema 2, percebemos que uma mesma aresta inter-cluster incidente a um vértice $u \in V$ é somada no máximo $2\log\frac{n}{\epsilon}$ vezes no valor de $w(P(S'_j))$. Além disso, utilizando o fato de que $w(P(S'_j)) \leq W$ e o lema 3 ($W = \frac{\epsilon}{2}a(V)$), teremos

$$\sum_{\text{para todo } j} w(P(S'_j)) \leq 2\log\left(\frac{n}{\epsilon}\right)W = 2\log\left(\frac{n}{\epsilon}\right)\frac{\epsilon}{n}a(V) = \epsilon\log\left(\frac{n}{\epsilon}\right)a(V). \tag{A.4}$$

Com isso, podemos encontrar um limite para o corte de "alta" condutância em relação ao custo do conjunto de clusteres ótimo fazendo o seguinte cálculo

$$\begin{aligned}
\sum_{j\in H} w(S_j, T_j) &\leq \sum_{\text{para todo } j} K(kw(P(S'_j)))^v a(S_j)^{1-v} \\
&\leq K\left(2\sum_{\text{para todo } j} w(P(S'_j))\right)^v \left(\sum_{\text{para todo } j} a(S_j)\right)^{1-v} \\
&\leq K\left(2\epsilon\log\left(\frac{n}{\epsilon}\right)a(V)\right)^v \left(2\log\left(\frac{n}{\epsilon}\right)a(V)\right)^{1-v} \text{ (usando (A.4))} \\
&\leq 2K\epsilon^v \log\left(\frac{n}{\epsilon}\right)a(V)
\end{aligned} \tag{A.5}$$

Agora trataremos dos clusteres com baixa condutância, ou seja, aqueles $j \notin H$. Inicialmente, vamos supor que todos os cortes realizados pelo algoritmo 14 induz uma partição de C_i em $P^i_1, P^i_2, \ldots, P^i_{r_i}$. Cada aresta entre dois vértice de C_i que pertencem a diferentes subconjuntos de partições devem pertencer a algum corte $c(S_j, T_j)$ e, consequentemente, cada aresta $e = (u, v)$ de todo corte $c(S_j \cap C_i, T_j \cap C_i)$ possuem os dois vértices u e v em diferentes subconjuntos da partição. Então, lembrando que a partição ótima, definida pelo conjunto $\{C_i\}$, é um (α, ϵ)-clustering, teremos que C_i tem condutância α, com isso, observe que

$$\frac{w(P^i_s, C_i \backslash P^i_s)}{min(a(P^i_s), a(C_i \backslash P^i_s))} \geq \alpha. \tag{A.6}$$

Note também que a seguinte igualdade é verdadeira

$$\sum_{\text{para todo j}} w_I(S_i \cap C_i, T_j \cap C_i) = \frac{1}{2}\sum_{s=1}^{r_i} w(P^i_s, C_i \backslash P^i_s). \tag{A.7}$$

Logo, unindo as expressões descritas em (A.6) e (A.7) teremos

$$\sum_{\text{para todo j}} w_I(S_j \cap C_i, T_j \cap C_i) = \frac{1}{2}\sum_{s=1}^{r_i} w(P_s^i, C_i \backslash P_s^i) \geq \frac{1}{2}\alpha \sum_{s=1}^{r_i} min(a(P_s^i), a(C_i \backslash P_s^i)) \quad \text{(A.8)}$$

Para cada vértice $u \in C_i$ pode existir no máximo $\log \frac{n}{\epsilon}$ valores de j tal que u pertença a algum subconjunto formado por $S_j \cap C_i$ ou $T_j \cap C_i$. Então, teremos que:

$$\sum_{s=1}^{r_i} min(a(P_s^i), a(C_i \backslash P_s^i)) \geq \frac{1}{\log \frac{n}{\epsilon}} \sum_{\text{para todo j}} min(a(S_j \cap C_i), a(T_j \cap C_i)). \quad \text{(A.9)}$$

Unindo as desigualdades (A.8) e (A.9) obteremos

$$\begin{aligned} \sum_{\text{para todo j}} w_I(S_j \cap C_i, T_j \cap C_i) &\geq \frac{1}{2}\alpha \sum_{s=1}^{r_i} min(a(P_s^i), a(C_i \backslash P_s^i)) \\ &\geq \frac{\alpha}{2\log \frac{n}{\epsilon}} \sum_{\text{para todo j}} min(a(S_j \cap C_i), a(T_j \cap C_i)) \end{aligned} \quad \text{(A.10)}$$

Como é verdadeiro que

$$\sum_{\text{para todo j}} \sum_{i=1}^{l} w_I(S_j \cap C_i, T_j \cap C_i) = \sum_{\text{para todo j}} w_I(S_j, T_j)$$

utilizando a desigualdade (A.10) teremos que

$$\sum_{\text{para todo j}} w_I(S_j, T_j) \geq \frac{\alpha}{2\log \frac{n}{\epsilon}} \sum_{\text{para todo j}} \sum_{i=1}^{l} min(a(S_j \cap C_i), a(T_j \cap C_i)).$$

Pela definição de H (considerando os índices $j \notin H$), temos que

$$\begin{aligned} \sum_{j \notin H} w_I(S_j, T_j) &\leq 2\alpha^* \sum_{\text{para todo j}} \sum_{i=1}^{l} min(a(S_i \cap C_i), a(T_j \cap C_i)) \\ &= 2\frac{\alpha}{6\log \frac{n}{\epsilon}} \sum_{\text{para todo j}} \sum_{i=1}^{l} min(a(S_i \cap C_i), a(T_j \cap C_i)) \\ &= \frac{\alpha}{3\log \frac{n}{\epsilon}} \sum_{\text{para todo j}} \sum_{i=1}^{l} min(a(S_i \cap C_i), a(T_j \cap C_i)) \\ &\leq \frac{2}{3} \sum_{\text{para todo j}} w_I(S_j, T_j). \end{aligned} \quad \text{(A.11)}$$

Podemos estender a desigualdade $\sum_{j\notin H} w_I(S_j,T_j) \leq \frac{2}{3}\sum_{\text{para todo j}} w_I(S_j,T_j)$ da seguinte forma:

$$\begin{aligned}
\sum_{j\notin H} w_I(S_j,T_j) &\leq \frac{2}{3}\sum_{\text{para todo j}} w_I(S_j,T_j) \\
&\quad \text{(somando } \sum_{j\in H} w_I(S_j,T_j) \text{ em ambos os lados)} \\
\Rightarrow \sum_{j\notin H} w_I(S_j,T_j) + \sum_{j\in H} w_I(S_j,T_j) &\leq \frac{2}{3}\sum_{\text{para todo j}} w_I(S_j,T_j) + \sum_{j\in H} w_I(S_j,T_j) \\
\Rightarrow \sum_{\text{para todo j}} w_I(S_j,T_j) &\leq \frac{2}{3}\sum_{\text{para todo j}} w_I(S_j,T_j) + \sum_{j\in H} w_I(S_j,T_j) \\
\Rightarrow \sum_{\text{para todo j}} w_I(S_j,T_j) - \frac{2}{3}\sum_{\text{para todo j}} w_I(S_j,T_j) &\leq \sum_{j\in H} w_I(S_j,T_j) \\
\Rightarrow \sum_{\text{para todo j}} w_I(S_j,T_j) &\leq 3\sum_{j\in H} w_I(S_j,T_j)
\end{aligned}$$

Pela desigualdade (A.11) sabemos que $\sum_{j\notin H} w_I(S_j,T_j) \leq \frac{2}{3}\sum_{allj} w_I(S_j,T_j)$, logo

$$\begin{aligned}
\Rightarrow \sum_{j\notin H} w_I(S_j,T_j) &\leq \frac{2}{3}3\sum_{j\in H} w_I(S_j,T_j) \\
\Rightarrow \sum_{j\notin H} w_I(S_j,T_j) &\leq 2\sum_{j\in H} w_I(S_j,T_j) \qquad \text{(A.12)}
\end{aligned}$$

Assim, somos capazes de limitar o custo das arestas inter-clusteres do grupo de condutância "baixa" em termos do custo das arestas de condutância "alta". Aplicando (A.5) em (A.12) teremos

$$\begin{aligned}
\sum_{\notin H} w_I(S_j,T_j) &\leq 2\sum_{j\in H} w_I(S_j,T_j) \\
&\leq 4K\epsilon^v \log\left(\frac{n}{\epsilon}\right) a(V). \qquad \text{(A.13)}
\end{aligned}$$

O somatório dos pesos das arestas de todos os cortes $c(S_j,T_j)$ com índices $j \notin H$ é $\sum_{j\notin H} w(S_j,T_j)$ e a soma dos pesos das arestas intra-cluster dos cortes é $\sum_{j\notin H} w_I(S_j,T_j)$. Logo, se fizermos a diferença $\sum_{j\notin H} w(S_j,T_j) - \sum_{j\notin H} w_I(S_j,T_j)$ teremos a soma das arestas inter-clusteres como um limitante superior, ou seja,

$$\sum_{j\notin H} w(S_j,T_j) - \sum_{j\notin H} w_I(S_j,T_j) \leq W,$$

onde W é a soma das arestas intra-clusteres. Sabemos que $\epsilon = \frac{W}{\sum_{(u,v)\in E} w(u,v)}$ (veja a definição de ϵ em (A.1)) e que $a(V) = 2\sum_{(u,v)\in E} w(u,v)$, logo

$$\frac{\epsilon}{2}a(V) = \frac{W}{2\sum_{(u,v)\in E} w(u,v)} 2 \sum_{(u,v)\in E} w(u,v) = W.$$

Portanto, podemos definir um novo limite superior para a diferença substituindo W por $\frac{\epsilon}{2}a(V)$

$$\sum_{j\notin H}(w(S_j,T_j) - w_I(S_j,T_j)) \leq \frac{\epsilon}{2}a(V). \tag{A.14}$$

O que resta para terminar a prova do teorema 11 é somar (A.5), (A.13) e (A.14). Fazendo a soma de (A.5) e (A.13)

$$\sum_{j\in H} w(S_j,T_j) + \sum_{j\notin H} w_I(S_j,T_j) \leq 6k\epsilon^v \log\frac{n}{\epsilon}a(V).$$

Somando essa desigualdade com a desigualdade (A.14)

$$\sum_{j\in H} w(S_j,T_j) + \sum_{j\notin H} w_I(S_j,T_j) = \sum_{\text{para todo j}} w(S_j,T_j) \leq 6k\epsilon^v \log\frac{n}{\epsilon}a(V) + \frac{\epsilon}{2}a(V).$$

Sabendo que $a(V) = 2\sum_{(u,v)\in E} w(u,v)$ obteremos a seguinte desigualdade

$$\begin{aligned}
\sum_{j\in H} w(S_j,T_j) + \sum_{j\notin H} w_I(S_j,T_j) &= \sum_{\text{para todo j}} w(S_j,T_j) \\
&\leq \sum_{(u,v)\in E} w(u,v)(12k\epsilon^v \log\frac{n}{\epsilon} + \epsilon). \\
\Rightarrow \epsilon^* = \frac{\sum_{\text{para todo j}} w(S_j,T_j)}{\sum_{(u,v)\in E} w(u,v)} &\leq 12k\epsilon^v \log\frac{n}{\epsilon} + \epsilon \\
&\leq (12k+2)\epsilon^v \log\frac{n}{\epsilon}
\end{aligned} \tag{A.15}$$

Note que ϵ^* é a fração entre o peso das arestas inter-clusteres dos cortes $c(S_j,T_j)$ feito pelo algoritmo *RCC* e o peso total das arestas. Isso completa a prova do teorema 11. ■

A.2 Aproximação para o Problema do Corte de Condutância Mínimo

Apresentaremos o teorema (teorema 12) que dá uma garantia de aproximação para o algoritmo que utiliza os autovetores da matriz de transição de probabilidade para encontrar o corte de condutância mínimo. O teorema e sua demonstração foram apresentados

no trabalho de R. Kannan, S. Vempala e A. Vetta [34]. Nesta seção, apresentaremos a demonstração detalhadamente.

Antes de analisar a prova do teorema, sugerimos ao leitor que leia a seção 3.1, onde são apresentados alguns conceitos básicos para o entendimento dessa demonstração. Inicialmente, apresentamos algumas definições e lemas importantes.

Definição 15 (Distribuição Estacionária) *Chama-se distribuição estacionária a distribuição de probabilidade dos estados, tal que, a distribuição no instante $n+1$ é igual a distribuição no instante n.*

Lema 4 *Seja M uma matriz de transição de probabilidade de um grafo $G = (V, E)$ com n vértices. Se o grafo G é conectado e não bipartido, então o caminho aleatório possui apenas uma distribuição estacionária $\pi = \{\pi_1, \pi_2, ..., \pi_n\}$, que é dado por*

$$\pi_i = \frac{\sum_{j \in V} a_{ij}}{a(V)},$$

onde a_{ij} refere-se ao valor na posição i e j da matriz M.

Prova:

Para o vetor $\pi \in \mathbb{R}^n$ ser estacionário, deve-se obedecer a seguinte propriedade $\pi M = \pi$.

Assim, seja Π um vetor em $\mathbb{R}^n$ com cada coordenada $\Pi_i = \frac{\sum_{j \in V} a_{ij}}{a(V)}$. Fazendo a multiplicação de Π pela matriz de transição M obteremos:

$$\Pi M = \begin{bmatrix} \Pi_1 & \Pi_2 & \Pi_3 & ... & \Pi_n \end{bmatrix} \begin{bmatrix} a_{11} & a_{12} & ... & a_{1n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^{n} \Pi_i a_{i1} \\ \sum_{i=1}^{n} \Pi_i a_{i2} \\ \vdots \\ \sum_{i=1}^{n} \Pi_i a_{in} \end{bmatrix}$$

para alguma linha j do produto de ΠM, temos:

$$\sum_{i=1}^{n} \Pi_i a_{ij} = \sum_{i=1}^{n} \left(\frac{\sum_{i=1}^{n} a_{ij}}{a(V)} \frac{a_{ij}}{\sum_{j=1}^{n} a_{ij}} \right) = \sum_{i=1}^{n} \frac{a_{ij}}{a(V)} = \frac{1}{a(V)} \sum_{i=1}^{n} a_{ij} = \Pi_j$$

Portanto $\Pi M = \Pi$, ou seja, Π é um vetor estacionário para M. Isso conclui a prova do lema 4. ■

Definição 16 (Condutância na Cadeia de Markov) *Seja M uma matriz de transição de probabilidade com o valor na coluna i e linha j representado por a_{ij}. Definiremos a condutância de um conjunto de estados S em uma cadeia de Markov como:*

$$\phi(S) = \frac{\sum_{i\in S, j\notin S} \pi_i a_{ij}}{\min(\pi(S), \pi(\overline{S}))}$$

Lema 5 *A condutância em grafo corresponde a condutância na Cadeia de Markov. Logo, a seguinte igualdade é verdadeira*

$$\phi(S) = \frac{\sum_{i\in S, j\notin S} a_{ij}}{min(a(S), a(\overline{S}))} = \frac{\sum_{i\in S, j\notin S} \pi_i a_{ij}}{\min(\pi(S), \pi(\overline{S}))},$$

onde π é a distribuição estacionária da cadeia de Markov.

Prova:
Sem perda de generalidade, suponha que $\min(a(S), a(\overline{S})) = a(S)$, então:

$$\phi(S) = \frac{\sum_{i\in S, j\notin S} a_{ij}}{\min(a(S), a(\overline{S}))} = \frac{\sum_{i\in S, j\notin S} a_{ij}}{a(S)} = \sum_{i\in S, j\notin S} a_{ij} \frac{1}{a(V)} \frac{a(V)}{a(S)}$$

$$= \sum_{i\in S, j\notin S} \left(\frac{\sum_{k=1}^{n} a_{ik}}{a(V)} \frac{a_{ij}}{\sum_{i=1}^{n} a_{ik}} \frac{a(V)}{a(S)} \right) = \sum_{i\in S, j\notin S} \pi_i b_{ij} \frac{a(V)}{a(S)}$$

Sabemos, pelo lema 4, que $\pi_i = \frac{\sum_{i\in V} a_{ij}}{a(V)}$, logo

$$\sum_{i\in S} \pi_i = \frac{\sum_{i\in S} \sum_{j\in V} a_{ij}}{a(V)} = \frac{a(S)}{a(V)} = \pi(S)$$

assim,

$$\sum_{i\in S} \pi_i b_{ij} \frac{a(V)}{a(S)} = \frac{\sum_{i\in S, j\notin S} \pi_i b_{ij}}{\pi(S)}$$

portanto, concluimos que

$$\sum_{i\in S, j\notin S} \pi_i b_{ij} \frac{a(V)}{a(S)} = \frac{\sum_{i\in S, j\notin S} \pi_i b_{ij}}{\pi(S)}$$

onde o $\min(\pi(S), \pi(\overline{S})) = \pi(S)$.

Teorema 12 *Suponha M uma matriz $n \times n$ não negativa e a soma de cada linha igual a 1, suponha também que exista números reais positivos $\pi_1, \pi_2, \pi_3, ..., \pi_n$ somando 1 tal que $\pi_i a_{ij} = \pi_j a_{ji}$ para todo i e j. Se v é o autovetor de M correspondente ao segundo maior*

autovalor λ_2, *e* $i_1, i_2, ..., i_n$ *é uma ordenação de* $1, 2, ..., n$ *tal que* $v_{i_1} \geq v_{i_2} \geq ... \geq v_{i_n}$, *então:*

$$\begin{aligned}\min_{S\subseteq\{1,2,...,n\}} \frac{\sum_{i\in S, j\notin S} \pi_i a_{ij}}{\min(\sum_{i\in S}\pi_i, \sum_{j\in S}\pi_j)} &\geq 1-\lambda_2 \\ &\geq \frac{1}{2}\left(\min_{l,1\leq l\leq n} \frac{\sum_{1\leq u\leq l; l+1\leq v\leq n}\pi_{i_u}a_{i_u i_v}}{\min(\sum_{1\leq u\leq l}\pi_{i_u}, \sum_{l+1\leq v\leq n}\pi_{i_v})}\right)^2\end{aligned}$$

Prova:

Inicialmente validamos o segundo maior autovalor. Seja uma matriz $D^2 = diag(\pi)$. Sabendo que $\pi_i a_{ij} = \pi_j a_{ji}$, teremos que $D^2 M = M^T D^2$ é verdade, logo, a matriz $Q = DMD^{-1}$ é simétrico. Além disso, teremos que os autovalores de M e Q são os mesmos, com o maior autovalor igual a 1.

Para mostrar que os autovalores de M e Q são iguais, faremos os seguintes cálculos:

$$\begin{aligned} Qv &= \lambda v \\ \Rightarrow DMD^{-1}v &= \lambda v \quad \text{multiplicando por } D^{-1}\text{teremos} \\ \Rightarrow D^{-1}DMD^{-1}v &= D^{-1}\lambda v = MD^{-1}v \\ \Rightarrow MD^{-1}v &= \lambda D^{-1}v \end{aligned}$$

Note que $D^{-1}v$ é o autovetor de M com o autovalor λ. Logo, λ é autovetor de M e Q.

Como $\pi_i a_{ij} = \pi_j a_{ji}$, temos que π é a distribuição estacionária de M. Além disso, notemos que o vetor πD^{-1} é o autovetor estacionário de Q satisfazendo

$$\pi D^{-1} Q = \pi D^{-1}.$$

Para provar isso, faremos os seguintes cálculos:

$$\pi D^{-1} Q = \pi D^{-1} DMD^{-1} = \pi M D^{-1} = \pi D^{-1},$$

lembre-se que $\pi M = \pi$.

A matriz Q é simétrica, logo, $Q\pi^T D^{-1} = \pi^T D^{-1}$. Com isso, podemos concluir que $\pi^T D^{-1}$ é autovetor de Q correspondente ao autovalor 1.

O maior autovalor de Q é 1. Lembrando que todos os autovetores são linearmente independentes, assim, o segundo maior autovalor λ_2 será aquele cujo o seu autovetor x obedeça a igualdade $\pi D^{-1} \cdot x = 0$. O segundo autovetor pode ser encontrado da seguinte forma

$$\lambda_2 = \max_{\pi D^{-1} \cdot x = 0} \frac{x^T D M D^{-1} x}{x^T x} \tag{A.16}$$

O que faremos agora é reescrever (A.16) substituindo por $y = D^{-1}x$.

$$1 - \lambda_2 = \min_{\pi^T D^{-1} x = 0} \frac{x^T D(I - M)D^{-1}x}{x^T x} = \min_{\pi^T y = 0} \frac{y^T D^2 (I - M) y}{y^T D^2 y} \tag{A.17}$$

O cálculo a seguir mostra como foi transformada a equação (A.16) em (A.17). Cada posição i e j da matriz Q será denotada por q_{ij}. Os cálculos são apresentados abaixo.

$$x^T Q x = \begin{bmatrix} x_1 & x_2 & \dots & x_n \end{bmatrix} \begin{bmatrix} q_{11} & \dots & q_{1n} \\ \vdots & \ddots & \vdots \\ q_{n1} & \dots & q_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} =$$

$$\begin{bmatrix} \sum_{i=1}^n x_i q_{i1} & \dots & \sum_{i=1}^n x_i q_{in} \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} = \sum_{j=1}^n \left(x_j \sum_{i=1}^n x_i q_{ij} \right)$$

Agora, fazendo para a matriz $(I - Q)$, vamos ter o seguinte

$$x^T (I - Q) x = \begin{bmatrix} x_1 & x_2 & \dots & x_n \end{bmatrix} \begin{bmatrix} 1 - q_{11} & \dots & -q_{1n} \\ \vdots & \ddots & \vdots \\ -q_{n1} & \dots & 1 - q_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} =$$

$$\begin{bmatrix} x_1 - \sum_{i=1}^n x_i q_{i1} & \dots & x_n - \sum_{i=1}^n x_i q_{in} \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} = \sum_{i=1}^n x_i^2 - \sum_{j=1}^n \left(x_j \sum_{i=1}^n x_i q_{ij} \right)$$

fazendo a divisão por $x^T x$, teremos

$$\frac{x^T (I - Q) x}{x^T x} = \frac{\sum_{i=1}^n x_i^2}{\sum_{i=1}^n x_i^2} - \frac{\sum_{j=1}^n (x_j \sum_{i=1}^n x_i q_{ij})}{\sum_{i=1}^n x_i^2} = 1 - \frac{\sum_{j=1}^n (x_j \sum_{i=1}^n x_i q_{ij})}{\sum_{i=1}^n x_i^2} = 1 - \frac{x^T A x}{x^T x}$$

sabendo que $\lambda_2 = \max_{\pi^T D^{-1} x = 0} \frac{x^T Q x}{x^T x}$ e que, o maior autovalor é 1, teremos o seguinte

$$1 - \max_{\pi^T D^{-1} x = 0} \frac{x^T Q x}{x^T x} = 1 - \lambda_2 = \min_{\pi^T D^{-1} x = 0} \frac{x^T (I - Q) x}{x^T x}$$

substituindo $y = D^{-1}x$ ($x = Dy$ e $x^T = y^T D$), lembrando que $QD^{-1}v = \lambda D^{-1}v$. Note que y é autovetor de Q e M.

$$1 - \lambda_2 = \min_{\pi^T D^{-1} x = 0} \frac{x^T D(I - M)D^{-1}x}{x^T x} = \min_{\pi^T y = 0} \frac{y^T D^2 (I - M)y}{y^T D^2 y}$$

Reescrevendo o numerador:

$$\begin{aligned} y^T D^2(I - M)y &= -\sum_{i \neq j} y_i y_j \pi_i a_{ij} + \sum_i \pi_i (1 - a_{ii}) y_i^2 \\ &= -\sum_{i \neq j} y_i y_j \pi_i a_{ij} + \sum_i \pi_i a_{ij} \left(\frac{y_i^2 + y_j^2}{2} \right) \\ &= \sum_{i<j} \pi_i a_{ij} \left(y_i - y_j \right)^2 \end{aligned}$$

Denote $\sum_{i<j} \pi_i b_{ij} \left(y_i - y_j \right)^2$ por $\varepsilon(y, y)$. Então

$$1 - \lambda_2 = \min_{\pi^T y = 0} \frac{\varepsilon(y, y)}{\sum_i \pi_i y_i^2}$$

Seja $E(S, \overline{S})$ um corte com a condutância mínima. Definiremos um vetor w como

$$w_i = \begin{cases} \sqrt{\frac{1}{\sum_{u \in V} a(u)} \frac{\pi(\overline{S})}{\pi(S)}} & \text{se } i \in S \\ -\sqrt{\frac{1}{\sum_{u \in V} a(u)} \frac{\pi(S)}{\pi(\overline{S})}} & \text{se } i \in \overline{S} \end{cases}$$

Vamos verificar que $\sum_i \pi_i w_i = 0$. Sabendo que $\pi_i = \frac{\sum_{j=1}^n a_{ij}}{a(V)}$, teremos

$$\sum_i \pi_i w_i = \sum_{i \in S} \pi_i w_i + \sum_{i \in \overline{S}} \pi_i w_i$$

$$= \sum_{i \in S} \left(\sum_{j=1}^n a_{ij} \frac{1}{a(V)} \sqrt{\frac{1}{\sum_{u \in V} a(u)} \frac{\pi(\overline{S})}{\pi(S)}} \right) - \sum_{i \in \overline{S}} \left(\sum_{j=1}^n a_{ij} \frac{1}{a(V)} \sqrt{\frac{1}{\sum_{u \in V} a(u)} \frac{\pi(S)}{\pi(\overline{S})}} \right)$$

(Sabemos que $\frac{\sum_{i \in S} \sum_{j=1}^n a_{ij}}{a(V)} = \frac{a(S)}{a(V)} = \pi(S)$)

$$= \sqrt{\frac{1}{\sum_{u \in V} a(u)} \frac{\pi(\overline{S})}{\pi(S)}} \pi(S) - \sqrt{\frac{1}{\sum_{u \in V} a(u)} \frac{\pi(S)}{\pi(\overline{S})}} \pi(\overline{S})$$

(fazendo a multiplicação por $\frac{\sqrt{\sum_{u\in V} a(u)\pi(S)}}{\sqrt{\sum_{u\in V} a(u)\pi(S)}}$).

$$= \frac{\sqrt{\pi(\overline{S})}}{\sum_{u\in V} a(u)} \frac{\pi(S)}{\pi(S)} \sqrt{\sum_{u\in V} a(u)\pi(S)} - \frac{\sqrt{\pi(S)}}{\sum_{u\in V} a(u)} \frac{\pi(\overline{S})}{\pi(\overline{S})} \sqrt{\sum_{u\in V} a(u)\pi(S)}$$

$$= \frac{\sqrt{\pi(\overline{S})\pi(S)\sum_{u\in V} a(u)}}{\sum_{u\in V} a(u)} - \frac{\sqrt{\pi(\overline{S})\pi(S)\sum_{u\in V} a(u)}}{\sum_{u} a(u)} = 0$$

Agora, resta provar a desigualdade

$$\phi(S) \geq \frac{\varepsilon(w,w)}{\sum_i \pi_i w_i^2} \geq 1 - \lambda_2$$

O que queremos provar é que

$$\phi(S) = \frac{\sum_{i\in S, j\in \overline{S}} \pi_i b_{ij}}{min(\pi(S), \pi(\overline{S}))} \geq \frac{\varepsilon(w,w)}{\sum_i \pi_i w_i^2} = \frac{\sum_{i<j} \pi_i b_{ij}(y_i - y_j)^2}{\sum_i \pi_i y_i^2} \geq 1 - \lambda_2$$

Antes, vamos apresentar uma lema importante que será usado na prova.

Lema 6 *Escrevendo $\varepsilon(w,w)$ teremos o seguinte: $\varepsilon(w,w) = \sum_{i<j} \pi_i b_{ij}(w_i - w_j)^2$. Observe que $(w_i - w_j)$ é diferente de zero se e somente se $i \in S$ e $j \notin S$, logo, podemos escrever $(w_i - w_j)$ da seguinte maneira:*

$$(w_i - w_j) = \sqrt{\frac{1}{\sum_u a(u)} \frac{\pi(\overline{S})}{\pi(S)}} + \sqrt{\frac{1}{\sum_u a(u)} \frac{\pi(S)}{\pi(\overline{S}}} = (\pi(S) - \pi(\overline{S}))\sqrt{\frac{1}{\sum_u a(u)\pi(S)\pi(\overline{S})}}$$

Prova:

$$(w_i - w_j) = \sqrt{\frac{1}{\sum_u a(u)} \frac{\pi(\overline{S})}{\pi(S)}} + \sqrt{\frac{1}{\sum_u a(u)} \frac{\pi(S)}{\pi(\overline{S}}}$$

($\sqrt{x} - \sqrt{y} = \sqrt{x + y - 2\sqrt{xy}}$ para $x > y$)

$$= \sqrt{\frac{1}{\sum_u a(u)} \frac{\pi(\overline{S})}{\pi(S)} + \frac{1}{\sum_u a(u)} \frac{\pi(S)}{\pi(\overline{S})} - 2\sqrt{\frac{1}{\sum_u a(u)^2}}}$$

$$= \sqrt{\frac{\sum_u a(u)\pi(S)^2 + \sum_u a(u)\pi(\overline{S})^2}{\sum_u a(u)^2\pi(S)\pi(\overline{S})} - \frac{2}{\sum_u a(u)}}$$

$$= \sqrt{\frac{\pi(S)^2 + \pi(\overline{S})^2}{\sum_u a(u)\pi(S)\pi(\overline{S})} - \frac{2}{\sum_u a(u)}} = \sqrt{\frac{1}{\sum_u a(u)}\left(\frac{\pi(S)^2 + \pi(\overline{S})^2}{\pi(S)\pi(\overline{S})} - 2\right)}$$

$$= \sqrt{\frac{1}{\sum_u a(u)}\left(\frac{(\pi(S) - \pi(\overline{S}))^2}{\pi(S)\pi(\overline{S})}\right)} = (\pi(S) - \pi(\overline{S}))\sqrt{\frac{1}{\sum_u a(u)\pi(S)\pi(\overline{S})}}$$

■

Agora podemos continuar a prova da primeira desigualdade:

$$\frac{\sum_{i<j} \pi_i b_{ij}(y_i - y_j)^2}{\sum_i \pi_i y_i^2}$$

(utilizando o lema 6)

$$= \sum_{i<j} \pi_i b_{ij}(\pi(S) + \pi(\overline{S}))^2 \frac{1}{\sum_u a(u)\pi(S)\pi(\overline{S})}$$

$$= \sum_{i<j} \pi_i b_{ij}\left(\frac{\sum_{u\in S}\sum_{v\in V} a_{uv}}{a(V)} + \frac{\sum_{u\in \overline{S}}\sum_{v\in V} a_{uv}}{a(V)}\right)^2 \frac{1}{a(V)\pi(S)\pi(\overline{S})}$$

$$= \sum_{i<j} \pi_i b_{ij}\left(\frac{\sum_{u\in V}\sum_{v\in V} a_{uv}}{a(V)}\right)^2 \frac{1}{a(V)\pi(S)\pi(\overline{S})} = \sum_{i<j} \pi_i b_{ij}\left(\frac{\sum_{u\in V}\sum_{v\in V} a_{uv}}{a(V)}\right)^2 \frac{1}{a(V)\pi(S)\pi(\overline{S})}$$

$$= \sum_{i<j} \pi_i b_{ij} \frac{1}{a(V)\pi(S)\pi(\overline{S})} \leq \sum_{i\in S, j\in \overline{S}} \pi_i b_{ij}$$

Isso finaliza a prova da primeira desigualdade do teorema 12.

Iremos provar a segunda desigualdade:

$$1 - \lambda_2 \geq \frac{1}{2}\left(\min_{l,1\leq l\leq n} \frac{\sum_{1\leq u\leq l; l+1\leq v\leq n} \pi_{i_u} a_{i_u i_v}}{\min(\sum_{1\leq u\leq l} \pi_{i_u}, \sum_{l+1\leq v\leq n} \pi_{i_v})}\right)^2$$

Suponha que o valor de

$$1 - \lambda_2 = min_{\pi^T y=0} \frac{\varepsilon(y,y)}{\sum_i \pi_i y_i^2}$$

é alcançado quando y é igual a v. Então Dv é um autovetor de Q correspondente ao autovalor λ_2 e v é o autovetor de M correspondente a λ_2.

Assuma que os índices são ordenados tal que $v_1 \geq v_2 \geq ... \geq v_n$. Agora defina r satisfazendo $\pi_1 + \pi_2 + ... + \pi_{r-1} \leq \frac{1}{2} < \pi_1 + \pi_2 + ... + \pi_r$ e seja $z_i = v_i - v_r$ para $i = 1, ..., n$. Então $z_1 \geq z_2 \geq ... \geq z_r = 0 \geq z_{r+1} \geq ... \geq z_n$, e

$$\frac{\varepsilon(v,v)}{\sum_i \pi_i v_i^2} = \frac{\varepsilon(z,z)}{-v_r^2 + \sum_i \pi_i z_i^2} \tag{A.18}$$

A igualdade (A.18) é verdadeira e pode ser comprovada com os seguintes cálculos.

$$\varepsilon(v,v) = \sum_{i<j} \pi_i b_{ij}(v_i - v_j)^2 = \sum_{i<j} \pi_i b_{ij}(z_i + v_r - (z_j + v_r))^2$$

$$= \sum_{i<j} \pi_i b_{ij}(z_i - z_j + v_r - v_r)^2 = \sum_{i<j} \pi_i b_{ij}(z_i - z_j)^2 = \varepsilon(z,z)$$

Para comprovar o denominador de (A.18) faremos

$$\sum_i \pi_i v_i^2 = \sum_i \pi_i (z_i + v_r)^2 = \sum_i \pi_i (z_i^2 + 2 z_i v_r + v_r^2) = \sum_i \pi_i z_i^2 + \sum_i \pi_i (2 z_i v_r + v_r^2) = -v_r^2 + \sum_i \pi_i z_i^2$$

Assim,

$$\begin{aligned} \frac{\varepsilon(v,v)}{\sum_i \pi_i v_i^2} = \frac{\varepsilon(z,z)}{-v_r^2 + \sum_i \pi_i z_i^2} &\geq \frac{\varepsilon(z,z)}{\sum_i \pi_i z_i^2} \\ &= \frac{\left(\sum_{i<j} \pi_i a_{ij}(z_i - z_j)^2\right)\left(\sum_{i<j} \pi_i a_{ij}(|z_i| - |z_j|)^2\right)}{\left(\sum_i \pi_i z_i^2\right)\left(\sum_{i<j} \pi_i a_{ij}(|z_i| - |z_j|)^2\right)} \end{aligned}$$

Considerando o numerador do termo final. Por Cauchy-Schwartz

$$\begin{aligned} \left(\sum_{i<j} \pi_i a_{ij}(z_i - z_j)^2\right)\left(\sum_{i<j} \pi_i a_{ij}(|z_i| - |z_j|)^2\right) &\geq \left(\sum_{i<j} \pi_i a_{ij}|z_i - z_j|(|z_i| - |z_j|)\right) \\ &\geq \left(\sum_{i<j} \pi_i a_{ij} \sum_{k=i}^{j-1} |z_{k+1}^2 - z_k^2|\right) \end{aligned} \tag{A.19}$$

A segunda desigualdade da inequação (A.19) segue do fato de que, se $i < j$ então $|z_i - z_j|(|z_i| + |z_j|) \geq \sum_{k=1}^{j-1} |z_{k+1}^2 - z_k^2|$. Isso segue da observação de que

i) Se z_i e z_j tem o mesmo sinal (i.e. $r \notin \{i, i+1, ..., j\}$), então $|z_i - z_j|(|z_i| + |z_j|) = |z_i^2 - z_j^2|$.

ii) Caso contrário, se z_i e z_j tem sinais diferentes, então $|z_i - z_j|(|z_i| + |z_j|) = (|z_i| + |z_j|)^2 > z_i^2 + z_j^2$.

Também teremos,

$$\sum_{i<j} \pi_i a_{ij} (|z_i| + |z_j|)^2 \leq 2 \sum_{i<j} \pi_i a_{ij} (z_i^2 + z_j^2) \leq 2 \sum_i \pi_i z_i^2.$$

Como resultado, teremos

$$\begin{aligned} \frac{\varepsilon(v,v)}{\sum_i \pi_i v_i^2} &\geq \frac{\left(\sum_{i<j} \pi_i a_{ij} (z_i - z_j)^2\right) \left(\sum_{i<j} \pi_i a_{ij} (|z_i| - |z_j|)^2\right)}{\left(\sum_i \pi_i z_i^2\right) \left(\sum_{i<j} \pi_i a_{ij} (|z_i| - |z_j|)^2\right)} \\ &\geq \frac{\left(\sum_{i<j} \pi_i a_{ij} \sum_{k=i}^{j-1} |z_{k+1}^2 - z_k^2|\right)}{2 \left(\sum_i \pi_i z_i^2\right)^2} \end{aligned} \tag{A.20}$$

Seja $S_k = \{1, 2, ..., k\}$, $C_k = \{(i,j) : i \leq k < j\}$ e

$$\varpi = \min_{k, 1 \leq k \leq n} \frac{\sum_{(i,j) \in C_k} \pi_i a_{ij}}{\min\left(\sum_{i:i \leq k} \pi_i, \sum_{i:i>k} \pi_i\right)}.$$

Desde que $z_r = 0$, obtemos

$$\begin{aligned} \sum_{i<j} \pi_i a_{ij} \sum_{k=i}^{j-1} |z_{k+1}^2 - z_k^2| &= \sum_{k=1}^{n-1} |z_{k+1}^2 - z_k^2| \sum_{(i,j) \in C_k} \pi_i a_{ij} \\ &\geq \varpi \left(\sum_{k=1}^{r-1} (z_k^2 - z_{k+1}^2) \pi(S_k) + \sum_{k=r}^{n-1} \left(z_{k+1}^2 - z_k^2\right) (1 - \pi(S_k)) \right) \\ &= \varpi \left(\sum_{k=1}^{n-1} (z_k^2 - z_{k+1}^2) \pi(S_k) + (z_n^2 - z_r^2) \right) \\ &= \varpi \left(\sum_{k=1}^{n} \pi_k z_k^2 \right). \end{aligned} \tag{A.21}$$

Consequentemente, se $\pi^T y = 0$, então

$$1 - \lambda_2 = \frac{\varepsilon(v,v)}{\sum_i \pi_i v_i^2} \geq \frac{\varpi^2}{2}.$$

O que conclui a prova da segunda desigualdade do teorema 12. ■

Apêndice B

Algoritmos para Problemas de Classificação

Abaixo segue a descrição dos vários trabalhos levantados durante a revisão bibliográfica, em especial, apenas trabalhos que tratam do problema de classificação são descritos.

Approximation Algorithms for Classification Problems with Pairwise Relationships: Metric Labeling and Markov Random Fields - [37] - 1999 : Neste trabalho é apresentado um algoritmo para o problema de Classificação. Os principais resultados são um algoritmo $O(\log |L| \log \log |L|)$-aproximado para o caso métrico geral e uma $2-$aproximação para o caso de classificação com *métrica uniforme*.

Para o caso com *métrica uniforme* o algoritmo resolve uma formulação em Programação Linear (PL) do problema, e depois realiza arredondamento da solução fracionária encontrada.

Na formulação da PL é usada a variável não negativa x_{pa} para cada objeto $p \in P$ e um *label* $a \in L$ tal que $\sum_{a \in L} x_{pa} = 1$; i.e. para o valor inteiro da variável x é usado $x_{pa} = 1$ para denotar que $f(p) = a$ – atribui ao objeto p o *label* a. O custo de atribuição de um objeto p é expressado como $\sum_{a \in L} c(p, a) x_{pa}$. Para dois objetos p e q, uma variável z_e corresponderá a distância entre os *labels* $f(p)$ e $f(q)$, no caso com métrica uniforme a variável $z_e = 1$ se esses dois objetos são associados a *labels* diferentes e $z_e = 0$ caso contrário. Uma outra variável, z_{ea}, é utilizada para expressar o valor absoluto para um *label* $a \in L$, ou seja, para uma aresta $e = \{p, q\} \in E$ a variável z_{ea} expressa o valor absoluto de $x_{pa} - x_{qa}$ ($z_{ea} = |x_{pa} - x_{qa}|$).

Quanto as restrições, aquela que garante que cada objeto seja associado a exatamente uma classe é dada por

$$\sum_{a \in L} x_{pa} = 1.$$

Para realizar a associação da variável z_e (para uma aresta $e = \{p, q\}$) com a distância dos *labels* associados a dois objetos p e q é feita a seguinte restrição:

$$z_e = \frac{1}{2} \sum_{a \in L} z_{ea}.$$

A distância dos *labels* associados a dois objetos p e q, que corresponde ao valor absoluto de $x_{pa} - x_{qa}$, é representado no programa linear pela restrição

$$z_{ea} \geq x_{pa} - x_{qa} \quad \text{e} \quad z_{ea} \geq x_{qa} - x_{pa}.$$

Assim, com todas as restrições descritas acima e também adicionando a restrição $x_{pa} \geq 0$ para a relaxação do programa linear, obtemos a seguinte formulação para o problema de classificação com métrica uniforme:

$$\begin{array}{lll} Min & \sum_{e \in E} w_e z_e + \sum_{p \in P, a \in L} c(p,a) x_{pa} & \\ s.t. & \sum_{a \in L} x_{pa} = 1 & p \in P \\ & z_e = \frac{1}{2} \sum_{a \in L} z_{ea} & e \in E \\ & z_{ea} \geq x_{pa} - x_{qa} & e = \{p, q\}, a \in L \\ & z_{ea} \geq x_{qa} - x_{pa} & e = \{p, q\}, a \in L \\ & x_{pa} \geq 0 & p \in P, a \in L. \end{array}$$

Para realizar o arredondamento, é utilizado um algoritmo iterativo. Em cada iteração é escolhido aleatoriamente um valor θ no intervalo $[0, 1]$ e um *label* $a \in L$. Caso o valor da variável x_{pa}, onde p seja um objeto qualquer ainda não classificado, for maior que θ, o objeto p é atribuído ao *label* a. As iterações prosseguem até que todos os objetos sejam atribuídos a exatamente um *label*. O algoritmo é uma 2-aproximação para o caso métrico uniforme.

Para o caso em que a distância não é uniforme o algoritmo utiliza dos resultados de Bartal [6] [7] em que é utilizado árvore métrica para aproximar um espaço métrico finito. Semelhante ao caso métrico uniforme, o caso geral utiliza programação linear relaxada e arredondamento para dar a solução aproximada. O algoritmo possui um fator de aproximação de $O(\log |L| \log \log |L|)$ para o caso métrico geral.

Segundo o trabalho de Bracht e Miyazawa [10], a formulação para o caso métrico uniforme possui $O(|L|n^2)$ restrições e $O(|L|n^2)$ variáveis (n corresponde ao número de objetos e $|L|$ o número de *labels*), sendo assim, ruim para instâncias grandes.

A Constant Factor Approximation Algorithm for a Class of Classification Problems [29] - 2000: No artigo é dado um algoritmo 4-aproximado para o caso em que a distância entre dois *labels* i e j é no máximo um valor constante M. É assumido que a distância métrica entre dois *labels* i e j é dada pela *métrica linear truncada* ($d(i, j) =$

$min\{M, |i-j|\}$). O algoritmo utiliza busca local, sendo que a cada iteração do algoritmo é construído uma rede de fluxo e o corte mínimo da rede define a atribuição de classe para os objetos. Na rede de fluxo são relacionados os custos de associação e os valores pagos para a separação das classes.

É mostrado que o laço principal do algoritmo é repetido $O((\frac{l}{M})(\log Q_0 + \log \epsilon^{-1})$ (Q_0 é o custo da atribuição inicial) para alcançar uma $(4+\epsilon)-$aproximação.

Para o caso de *classificação métrica uniforme*: onde a função $d(i,j)$ assume valor 1, se $i \neq j$, e 0 caso contrário, é mostrado que o algoritmo é uma $2-$aproximação.

Talvez o algoritmo não apresente bons resultados práticos devido a necessidade de construir, em cada iteração, um grafo correspondente a rede de fluxo com $O(|L|^2 n)$ arestas (n é o número de vértices).

Approximation Algorithms for the Metric Labeling Problem via a New Linear Programming Formulation [15] - 2001: Neste trabalho é apresentado uma formulação de programação linear para o problema de Classificação. É dado um algoritmo que apresenta uma $(2+\sqrt{2})$-aproximação para o *truncated linear norm* melhorando a $4-$aproximação dada em [29] (resumo da seção anterior). Para distância uniforme o algoritmo mantém a $2-$aproximação (resumo da seção anterior).

Devido a necessidade da resolução do programa linear relaxado formado por $O(n^2|L|^2)$ restrições e $O(n^2|L|^2)$ variáveis (n é o número de vértices), o algoritmo pode ter tempo de execução muito alto na prática. [10].

Approximating a Class of Classification Problems [46] - 2006: O artigo apresenta uma visão geral do problema de classificação mostrando vários resultados de outros trabalhos.

Na seção 2 do artigo é apresentada a relação entre o problema de classificação (ou *labeling problem*) (LaP) com o problema de *multiterminal or multiway cut problem (MCP)*. O problema MCP é definido da seguinte forma: dado um grafo $G = (V, E)$ com pesos positivos nas arestas $w(e)$, $e \in E$ e um conjunto T de k nodos terminais, o problema consiste em minimizar o peso do conjunto de arestas ϕ tal que a remoção do conjunto ϕ de E desconecta cada nodo terminal dos outros.

Obviamente, a remoção do conjunto de arestas ϕ de E cria exatamente k componentes conexos. Cada componente contém um nodo terminal, em outras palavras, cada vértice não terminal em um componente conexo será uma atribuição a um vértice (nodo) terminal. Assim, o problema MCP pode ser reescrito de maneira semelhante ao problema LaP da seguinte maneira: dado um conjunto de vértices terminais $T \subset V$, e uma função f que atribui cada vértice a um terminal $f : V \to T$, tal que $f(t) = t$ para todo $t \in T$ busca-se minimizar $\sum_{(u,v)\in E} w(u,v)d(f(u), f(v))$, onde (T, d) é uma métrica uniforme. Lembrando

que uma métrica uniforme define para cada label $a, b \in L$, $d(a, b) = 1$ se $a \neq b$ e zero caso contrário.

Dahlhaus [19] apresenta uma $(2-\frac{2}{k})-$aproximação para o problema MCP. O algoritmo funciona da seguinte forma:

Algoritmo 15: Algoritmo de Dahlhaus

1 **for** *cada nodo terminal* $t \in T$ **do**
2 unir todos os terminais em um novo terminal $\bar{t}$ (colocando $w(u, \bar{t}) = \sum_{t' \in T \setminus t} w(u, t')$) ;
3 procurar o corte mínimo entre t e $\bar{t}$;
4 retornar a união dos $(k-1)$ cortes mais baratos.

A remoção de todas as arestas que fazem parte dos $(k-1)$ cortes mais baratos desconectará cada componente de todos os outros e então formará uma solução viável para MCP.

Algoritmos de Aproximação para o Problema de Classificação Métrica [10] - 2005: Neste artigo é apresentado um algoritmo $8 \log n-$aproximado para o problema de classificação métrica. O algoritmo é guloso e utiliza a mesma idéia de um algoritmo para a resolução do problema de localização de recursos. O algoritmo consiste em encontrar a estrela de menor custo ponderado; uma estrela é um grafo conexo onde apenas um vértice (o vértice central) possui grau maior que um. O vértice central da estrela representa uma classe e os vértices restantes são os objetos. A estrela de menor custo é escolhida, e os objetos dessa estrela são atribuidos a sua respectiva classe.

O algoritmo ganha em tempo em relação a outros algoritmos, entretanto possui a desvantagem de possuir um alto fator de aproximação.

Referências Bibliográficas

[1] R. K. Ahuja, J. B. Orlin, and R. E. Tarjan. Improved time bounds for the maximum flow problem. *SIAM J. Comput.*, 18(5):939–954, 1989.

[2] E. Anderson, Z. Bai, J. Dongarra, A. Greenbaum, A. McKenney, J. Du Croz, S. Hammerling, J. Demmel, C. Bischof, and D. Sorensen. Lapack: a portable linear algebra library for high-performance computers. In *Supercomputing '90: Proceedings of the 1990 ACM/IEEE conference on Supercomputing*, pages 2–11, Los Alamitos, CA, USA, 1990. IEEE Computer Society Press.

[3] K. Andreev and H. Räcke. Balanced graph partitioning. In *SPAA '04: Proceedings of the sixteenth annual ACM symposium on Parallelism in algorithms and architectures*, pages 120–124, New York, NY, USA, 2004. ACM.

[4] V. Arya, N. Garg, R. Khandekar, A. Meyerson, K. Munagala, and Vinayaka Pandit. Local search heuristics for k-median and facility location problems, 2002.

[5] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchett-Spaccamela, and M. Protasi. *Complexity and Approximation (Combinatorial Optimization Problems and Their Approximability Properties)*. Springer-Verlag, 1999.

[6] Y. Bartal. Probabilistic approximations of metric spaces and its algorithmic applications. *Proc. ACM FOCS*, 1996.

[7] Y. Bartal. On approximating arbitrary metrics by tree metrics. *Proc. ACM FOCS*, 1998.

[8] A. Ben-Dor, R. Shamir, and Z. Yakhini. Clustering gene expression patterns. *Journal of Computational Biology*, 6(3/4):281–297, 1999.

[9] N. Biggs. *Algebraic Graph Theory*. Cambridge Tracts in Mathematics 67. Cambridge University Press, 1974.

[10] E. C. Bracht, L. A. A. Meira, and F. K. Miyazawa. A greedy approximation algorithm for the uniform metric labeling problem analyzed by a primal-dual technique. *J. Exp. Algorithmics*, 10:2.11, 2005.

[11] U. Brandes, M. Gaertler, and D. Wagner. Experiments on graph clustering algorithms. In *In 11th Europ. Symp. Algorithms*, pages 568–579. Springer-Verlag, 2003.

[12] U. Brandes, M. Gaertler, and D. Wagner. Engineering graph clustering: Models and experimental evaluation. *ACM Journal on Experimental Algorithm*, 12(1):1–26, June 2008.

[13] A. Broder and M. Henzinger. *Algorithmic aspects of information retrieval on the web*. Kluwer Academic Publishers, Norwell, MA, USA, 2002.

[14] Moses Charikar, Sudipto Guha, Éva Tardos, Eva Tardos, and David B. Shmoys. A constant-factor approximation algorithm for the k-median problem, 1999.

[15] C. Chekuri, S. Khanna, J. Naor, and L. Zosin. Approximation algorithms for the metric labeling problem via a new linear programming formulation. In *Symposium on Discrete Algorithms*, pages 109–118, 2001.

[16] M. Chrobak, C. Kenyon, and N. Young. The reverse greedy algorithm for the metric k-median problem. *Inf. Process. Lett.*, 97(2):68–72, 2006.

[17] F. Chung. *Spectral Graph Theory (CBMS Regional Conference Series in Mathematics, No. 92) (Cbms Regional Conference Series in Mathematics)*. American Mathematical Society, February 1997.

[18] J. Chuzhoy and Y. Rabani. Approximating k-median with non-uniform capacities. In *SODA '05: Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 952–958, Philadelphia, PA, USA, 2005. Society for Industrial and Applied Mathematics.

[19] E. Dahlhaus, D.S. Johnson, C.H. Papadimitriou, P.D. Seymour, and M. Yannakakis. The complexity of multiterminal cuts. *SIAM Journal on Computing*, 23(4):864–894, 1994.

[20] S. Doddi, M. V. Marathe, S. S. Ravi, D. S. Taylor, and P. Widmayer. Approximation algorithms for clustering to minimize the sum of diameters. *Nordic J. of Computing*, 7(3):185–203, 2000.

[21] Guy Even. Fast approximate graph partitioning algorithms. *SIAM J. Comput.*, 28(6):2187–2214, 1999.

[22] T. Feo and M. Resende. A probabilistic heuristic for a computationally difficult set covering problem. *Operations Research Letters*, 8(2):67 – 71, 1989.

[23] P. Foggia, G. Percannella, C. Sansone, and Mario Vento. A graph-based clustering method and its applications. In Francesco Mele, Giuliana Ramella, Silvia Santillo, and Francesco Ventriglia, editors, *BVAI*, volume 4729 of *Lecture Notes in Computer Science*, pages 277–287. Springer, 2007.

[24] L.R. Ford and D.R. Fulkerson. Maximal flow through a network. *Canadian Journal of Mathematics*, 8:399–404, 1956.

[25] M. Gaertler. Clustering with spectral methods. Diplomarbeit, fachbereich informatik und informationswissenschaft, Universität Konstanz, March 2002.

[26] M.R. Garey and D.S. Johnson. *Computers and Intractability: a Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, San Francisco, 1979. Referência básica em teoria da complexidade. Inclui extenso catálogo de problemas NP-completos. As atualizações do catálogo têm sido publicadas na coluna "The NP-completeness column" do *Journal of Algorithms*. [Feofiloff].

[27] C. Godsil and G. Royle. *Algebraic Graph Theory*. Graduate Texts in Mathematics 207. Springer Science+Business Media, 2001.

[28] T. Gonzalez. Clustering to minimize the maximum intercluster distance. *Theor. Comput. Sci.*, 38:293–306, 1985.

[29] A. Gupta and E. Tardos. A constant factor approximation algorithm for a class of classification problems. In *STOC '00: Proceedings of the thirty-second annual ACM symposium on Theory of computing*, pages 652–658, New York, NY, USA, 2000. ACM.

[30] L. Hagen and A. Kahng. New spectral methods for ratio cut partitioning and clustering. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 11(9):1074–1085, 1992.

[31] D. Hochbaum and D. Shmoys. A best possible approximation algorithm for the k-center problem. *Mathematics of Operations Research*, 10(2):180–184, 1985.

[32] A. Howard and C. Rorres. *Álgebra Linear com aplicações*, volume 2. Bookman Companhia, Porto Alegre, RS, 8 edition, 2001.

[33] K. Jain and V. V. Vazirani. Approximation algorithms for metric facility location and k-median problems using the primal-dual schema and lagrangian relaxation. *J. ACM*, 48(2):274–296, 2001.

[34] R. Kannan, S. Vempala, and A. Vetta. On clusterings: Good, bad and spectral. *J. ACM*, 51(3):497–515, 2004.

[35] T. Kanungo, D. Mount, N. Netanyahu, C. Piatko, R. Silverman, and A. Wu. A local search approximation algorithm for k-means clustering. In *SCG '02: Proceedings of the eighteenth annual symposium on Computational geometry*, pages 10–18, New York, NY, USA, 2002. ACM.

[36] O. Kariv and S. L. Hakimi. An algorithmic approach to network location problens. i: The p-centers. *SIAM J. Appl. Math. No. 3*, 37, 1979.

[37] J. Kleinberg and E. Tardos. Approximation algorithms for classification problems with pairwise relationships: Metric labeling and markov random fields. In *Proceedings of the 40th Annuall IEEE Symposium on Foundations of Computer Science*, pages 14–23, 1999.

[38] R. Krauthgamer, J. Naor, and R. Schwartz. Partitioning graphs into balanced components. In *SODA '09: Proceedings of the Nineteenth Annual ACM -SIAM Symposium on Discrete Algorithms*, pages 942–949, Philadelphia, PA, USA, 2009. Society for Industrial and Applied Mathematics.

[39] A. Kumar, Y. Sabharwal, and S. Sen. A simple linear time (1+ ϵ)-approximation algorithm for k-means clustering in any dimensions. In *FOCS '04: Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science*, pages 454–462, Washington, DC, USA, 2004. IEEE Computer Society.

[40] P.J.Rousseeuw L. Kaufman. *Finding Groups in Data: An Introduction to Cluster Analysis*. John Wiley & Sons, 1990.

[41] T. Leighton and S. Rao. Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms. *Journal of the ACM*, 46 , No. 6:787–832, Nov. 1999.

[42] U. Luxburg. A tutorial on spectral clustering. *Statistics and Computing*, 17(4):395–416, 2007.

[43] J. Matousek. On approximate geometric k-clustering. *Discrete and Computational Geometry*, 24:61–83, 2000.

[44] M. Meilă and J. Shi. A random walks view of spectral segmentation. In *IEEE International Conference on Artificial Intelligence and Statistics*, 2001.

[45] Z. Michalewicz and D. B. Fogel. *How to Solve It: Modern Heuristics*. Springer, 2004.

[46] I. Milis. *Approximating a Class of Classification Problems*, volume 3484/2006 of *Lecture Notes in Computer Science*, pages 213–249. Springer Berlin / Heidelberg, January 2006.

[47] G. L. Nemhauser and L. A. Wolsey. *Integer and Combinatorial Optimization*. John Wiley and Sons, New York, 1988.

[48] A. Y. Ng, M. I. Jordan, and Y. Weiss. On spectral clustering: Analysis and an algorithm. In *Advances in Neural Information Processing Systems 14*, pages 849–856. MIT Press, 2001.

[49] L. Ramachandran, M. Kapoor, A. Sarkar, and A. Aggarwal. Clustering algorithms for wireless ad hoc networks. *Proceedings of the 4th international workshop on Discrete Algorithms and Methods for Mobile Computing and Communications*, 2000.

[50] J. Shi and J. Malik. Normalized cuts and image segmentation. In *CVPR '97: Proceedings of the 1997 Conference on Computer Vision and Pattern Recognition (CVPR '97)*, page 731, Washington, DC, USA, 1997. IEEE Computer Society.

[51] M. Steinbach, G. Karypis, and V. Kumar. A comparison of document clustering techniques. *KDD Workshop on Text Mining*, 2000.

[52] M. Stoer and F. Wagner. A simple min cut algorithm. In *ESA '94: Proceedings of the Second Annual European Symposium on Algorithms*, pages 141–147, London, UK, 1994. Springer-Verlag.

[53] M. Thorup. Quick k-median, k-center, and facility location for sparse graphs. *SIAM J. Comput.*, 34(2):405–432, 2005.

[54] S. van Dongen. *Graph Clustering by Flow Simulation*. PhD thesis, University of Utrecht, May 2000.

[55] D. Verma and M. Meila. A comparison of spectral clustering algorithms. *University of Washington, Tech. Rep. UW-CSE-03-05-01*, 2003.

[56] L.A. Wolsey. *Integer Programming*. Wiley, 1998.

[57] T. Xiang and S. Gong. Spectral clustering with eigenvector selection. *Pattern Recogn.*, 41(3):1012–1029, 2008.

[58] G. Xu. *Approximation algorithms for clustering and related problems*. PhD thesis, Buffalo, NY, USA, 2006. Adviser-Xu, Jinhui.

[59] K.U Yeung. *Cluster Analysis of Gene Expression Data*. PhD thesis, University of Washington, 2001.

[60] C. T. Zahn. Graph-theoretical methods for detecting and describing gestalt clusters. *Transactions on Computers*, C-20(1):68–86, 1971.