UNIVERSIDADE FEDERAL DO CEARÁ

CAMPUS QUIXADÁ

BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

JOÃO VITOR CHAVES DE OLIVEIRA

UM ALGORITMO APROXIMATIVO PARA O PROBLEMA DE ÁRVORE GERADORA MINIMIZANDO EXCESSO DE GRAU

QUIXADÁ – CEARÁ

2017

JOÃO VITOR CHAVES DE OLIVEIRA

UM ALGORITMO APROXIMATIVO PARA O PROBLEMA DE ÁRVORE GERADORA MINIMIZANDO EXCESSO DE GRAU

Monografia apresentada no curso de Ciência da Computação da Universidade Federal do Ceará, como requisito parcial à obtenção do título de bacharel em Ciência da Computação. Área de concentração: Computação.

Orientador: Me. Lucas Ismaily Bezerra Freitas

QUIXADÁ – CEARÁ

2017

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Biblioteca Universitária
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

O47a Oliveira, João Vitor Chaves de.
Um algoritmo aproximativo para o problema de árvore geradora minimizando excesso de grau / João Vitor Chaves de Oliveira. – 2017.
45 f. : il. color.

Trabalho de Conclusão de Curso (graduação) – Universidade Federal do Ceará, Campus de Quixadá, Curso de Ciência da Computação, Quixadá, 2017.
Orientação: Prof. Me. Lucas Ismaily Bezerra Freitas.

1. Teoria dos Grafos. 2. Árvore Geradora Mínima. 3. Algoritmo de Aproximação. I. Título.

CDD 004

JOÃO VITOR CHAVES DE OLIVEIRA

UM ALGORITMO APROXIMATIVO PARA O PROBLEMA DE ÁRVORE GERADORA MINIMIZANDO EXCESSO DE GRAU

Monografia apresentada no curso de Ciência da Computação da Universidade Federal do Ceará, como requisito parcial à obtenção do título de bacharel em Ciência da Computação. Área de concentração: Computação.

Aprovado em: ____/_____/_____.

BANCA EXAMINADORA

Me. Lucas Ismaily Bezerra Freitas (Orientador)
Universidade Federal do Ceará – UFC

Dr. Criston Pereira de Souza
Universidade Federal do Ceará - UFC

Me. Roberto Cabral Rabelo Filho
Universidade Federal do Ceará - UFC

À minha mãe.

AGRADECIMENTOS

Agradeço primeiramente a Deus, não só pela monografia, mas por tudo.

Depois, à minha mãe. Mãe, muito obrigado! Sem você não poderia ter chegado aqui. Você sempre será a responsável por todas as conquistas que eu possa realizar.

Aos meus irmãos Juliana, Samara, Sarina, Sávyo, Soraia e Suiane por sempre estarem ao meu lado e me apoiarem em tudo que preciso.

À UFC e seu corpo docente, direção e administração que não medem esforços para tornar a UFC Campus Quixadá cada vez melhor e proporcionar aos alunos uma ótima estrutura.

Ao Lucas, pela excelente orientação, disponibilidade, paciência e amizade. Lucas você é um exemplo a ser seguido por todos. Sua simplicidade e inteligência serão sempre por mim copiados (a inteligência será difícil, mas tudo bem). Sou muito grato pelas inúmeras conversas que mudavam totalmente o tema, que com certeza nunca serão esquecidas. Ah, e as caronas também.

Ao professor Criston que infelizmente nunca fui seu orientando, mas tive o privilégio de ser aluno e que durante a graduação pude aprender bastante. Criston, atribuo o meu bom rendimento acadêmico às cadeiras tão bem ministradas por você nas quais pude criar uma base bastante sólida para conseguir bons resultados nas cadeiras subsequentes.

Ao professor Roberto Cabral pelo tempo, pelas valiosas colaborações e sugestões.

Ao professor Paulo de Tarso pelo excelente professor, orientador e coordenador que não mediu esforços para nos ajudar, informar e motivar.

Aos professores Carlos Igor e Regis Pires pela excelente orientação no período de bolsas de PIBID e extensão.

Aos meus colegas de graduação Arthur, Daiane, Décio, Deives, Raul e Salathiel pela ajuda em meio as dificuldades durante essa jornada.

Por fim, a todos que direta ou indiretamente fizeram parte da minha formação, o meu muito obrigado.

“A persistência é o caminho do êxito”

(Charles Chaplin)

RESUMO

O problema de se obter uma árvore geradora é bem conhecida e pode ser resolvida de forma eficiente. Entretanto, a adição de pequenas restrições tornam o problema NP-difícil (WU; CHAO, 2004). Muitas restrições para o problema foram apresentadas na literatura e em particular, enfatiza-se o problema da árvore geradora de grau máximo mínimo. Este é o problema de se obter uma árvore geradora T de um grafo G tal que o grau máximo de T seja o menor dentre todas as árvores geradoras obtidas a partir de G. É fácil verificar que este problema é NP-difícil, pois quando o grau máximo de T é dois, T é equivalente a um caminho hamiltoniano. Neste trabalho, é proposta uma nova abordagem para o problema de árvore geradora com restrição de grau descrito da seguinte maneira: Dado um grafo G e um inteiro d, queremos encontrar uma árvore geradora T cuja soma dos graus excedentes em cada vértice (o quanto o grau do vértice excede d) em T seja a menor possível. Dessa maneira, é apresentado um algoritmo aproximativo onde se é comparado ao algoritmo descrito em (FURER; RAGHAVACHARI, 1994), algoritmo aproximativo que minimiza o grau máximo de uma árvore geradora e que possui uma íntima relação com a abordagem proposta neste trabalho. As comparações são realizadas em relação a média das somas dos excessos e o tempo de resposta retornadas por ambos os algoritmos.

Palavras-chave: Teoria dos Grafos. Árvore Geradora Mínima. Algoritmo de Aproximação.

ABSTRACT

The problem of obtaining a spanning tree is well-known and can be solved efficiently. However, even minor constraints makes the problem NP-hard (WU; CHAO, 2004). Many constraints to the problem have been presented in the literature and in particular, the problem of the minimum-degree spanning tree is emphasized. This is the problem of obtaining a spanning tree T of a graph G such that the maximum degree of T is the smallest among all the spanning trees obtained from G. It is easy to verify that this problem is NP-hard because when the maximum degree of T is two, T is equivalent to a hamiltonian path. In this work, a new approach is proposed for the problem of spanning tree with degree restriction, described as follows: Given a graph G and an integer d, we want to find a generating tree T whose sum of the degrees exceeds in each vertex (how much the degree of the vertex exceeds d) in T is the smallest possible. In this way, an approximate algorithm is presented, which is compared to the algorithm described in (FURER; RAGHAVACHARI, 1994), an algorithm that minimizes the maximum degree of a spanning tree, and which has an intimate relationship with the approach proposed in this work. The comparisons are performed in relation to the mean of the sums and the response time returned by both algorithms.

Keywords: Graph Theory. Minimum Spanning Tree. Approximation Algorithm.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

AGGMM	Árvore Geradora de Grau Máximo Mínimo
PAGMS$_d$	Problema de Árvore Geradora que Minimiza Excesso de Grau

LISTA DE SÍMBOLOS

Δ	Grau máximo de um grafo
Δ^*	Grau máximo de uma solução ótima
$\psi_G(e)$	Função de incidência da aresta e
$\delta(G)$	Grau mínimo de um grafo G
$d_G(v)$	Grau do vértice v em G
$e(D)$	Quantidade de arestas que incidem apenas no conjunto D
$\partial(D)$	Corte do conjunto de vértices D
$\mid \partial(D) \mid$	Número de arestas do corte do conjunto de vértices D

SUMÁRIO

1 INTRODUÇÃO

Muitas situações do mundo real podem ser convenientemente descritas por meio de um diagrama que consiste em um conjunto de pontos juntamente com linhas que unem certos pares destes pontos. Por exemplo, os pontos poderiam representar pessoas, com linhas juntando pares de amigos; ou podem ser centros de comunicação, com linhas representando ligações de comunicação.

Observe que em tais diagramas se está principalmente interessado em se dois pontos dados são unidos por uma linha. A maneira em que eles são unidos é imaterial. Uma abstração matemática de situações deste tipo dá origem ao conceito de um grafo (BONDY; MURTY, 2008). Chamamos os pontos de vértices e as ligações entre os pontos de arestas.

Podemos citar vários exemplos em que podemos usar grafos, como por exemplo, topologia de redes onde os vértices são os dispositivos (computadores, *smartphones, tablets, hubs*, roteadores etc) e as arestas são as ligações que cada dispositivo tem com os demais vértices. Em rodovias, ferrovias, projeto de transmissão de rede de energia, uma rede social onde a relação que existe é a de que um usuário A é amigo de um usuário B, entre muitos outros exemplos. Um problema bastante estudado é o de determinar uma Árvore Geradora de Grau Máximo Mínimo AGGMM. O problema da AGGMM consiste em construir uma árvore geradora de um grafo G no qual o grau máximo é o menor dentre todas as árvores geradoras de G. Esse problema é uma generalização do problema do caminho hamiltoniano, que é um problema NP-difícil. Note que obter um caminho hamiltoniano, equivale a encontrar uma árvore geradora de grau máximo igual a dois.

Alguns trabalhos foram desenvolvidos visando computar uma AGGMM. Em Furer e Raghavachari (1994) é apresentado um algoritmo de aproximação com uma garantia de desempenho aditiva de um, isto é, descrevem um algoritmo de tempo polinomial que encontra uma árvore geradora T de G tal que $\Delta \leq \Delta^* + 1$, onde Δ e Δ^* denotam respectivamente o grau máximo de T e o grau máximo de uma solução ótima. A menos que P = NP, este é o melhor resultado possível para este problema. Em Könemann e Ravi (2000), Goemans (2006) e Raidl (2000) são apresentados variações do problema de minimizar o grau máximo de uma árvore geradora com algumas restrições adicionais, e com técnicas variadas e que serão apresentadas em capítulos posteriores.

Neste trabalho, apresentamos um novo problema de árvore geradora com restrição de grau, porém com uma íntima relação com o AGGMM. Seja $G = (V, E)$ um grafo simples e

$d \in \mathbb{Z}$, $d \geq 2$, queremos encontrar uma árvore geradora T de G tal que $\sum_{v \in D} d_T(v) - d(a_T + b_T)$ seja o menor dentre todas as árvores geradoras obtidas a partir de G, onde D é o conjunto de vértices com grau maior ou igual a d, $d_T(v)$ é o grau do vértice v em T, a_T é o número de vértices com grau igual a d em T e b_T é o número de vértices com grau maior que d em T. Denotamos este número como soma dos excessos e o problema por PAGMS$_d$.

Dessa maneira, foi implementado um algoritmo aproximativo para o PAGMS$_d$ e comparado seus resultados com a aplicação do algoritmo descrito em Furer e Raghavachari (1994) no PAGMS$_d$. A intuição de aplicar o algoritmo de Furer e Raghavachari (1994) no PAGMS$_d$ é que ao minimizar o grau máximo de uma árvore geradora, estamos indiretamente minimizando a soma dos graus que excedem d, que é o objeto do PAGMS$_d$. Vale ressaltar que os testes feitos neste trabalho se concentraram em $d = 3$ e que para este valor de d, o problema é NP-difícil.

2 FUNDAMENTAÇÃO TEÓRICA

Neste Capítulo serão apresentados os conceitos utilizados neste trabalho. Na Seção 2.1 são apresentados Conceitos Básicos em Grafos, na Seção 2.2 são abordados conceitos sobre Classes de Complexidade e na Seção 2.3 Algoritmos Aproximativos.

2.1 Conceitos Básicos em Grafos

Um *grafo* G é um par ordenado $(V(G), E(G))$, onde $V(G)$ é um conjunto finito de elementos chamado *vértices*, $E(G)$ é um conjunto finito de elementos, disjuntos de $V(G)$, chamado *arestas* e, além disso G possui uma função de incidência ψ_G que para cada aresta $e \in E(G)$ associa um par não-ordenado de vértices, não necessariamente distintos, $\psi_G(e) = \{u, v\}$ (BONDY; MURTY, 2008).

Dado um grafo $G = (V, E)$, dizemos que $u, v \in V$ são *adjacentes* e $e \in E$ é *incidente* a u e v, se $\psi_G(e) = \{u, v\}$. Dizemos também que u, v são *extremos* de e. Quando G estiver claro no contexto, escreveremos apenas V, E em vez de $V(G)$ e $E(G)$.

Um *laço* é uma aresta da forma $\psi_G(e) = \{v, v\}$. Duas arestas e, f são *múltiplas* se possuem os mesmos extremos, ou seja, $\psi_G(e) = \{u, v\}$ e $\psi_G(f) = \{u, v\}$. Um grafo G é *simples* se não possui laços e arestas múltiplas. Caso contrário, dizemos que G é um *multigrafo*. Neste trabalho, abordamos apenas grafos simples. Assim, geralmente deixamos implícita a função de incidência, uma vez que dois vértices definem unicamente uma aresta em um grafo simples. Portanto, denotamos um grafo por $G = (V, E)$ e usamos $e = uv$ em vez de $\psi_G(e) = \{u, v\}$. Em geral, escrevemos grafo com o sentido de grafo simples, casos especiais são explicitados ao leitor.

O *grau* de um vértice v é o número de arestas que incidem em v. O grau de v em um grafo G será denotado por $d_G(v)$. O grau *grau mínimo* de um grafo G é o número $\delta(G) = \min\{d_G(v) : v \in V\}$. O *grau máximo* do grafo é o número $\Delta(G) = \max\{d_G(v) : v \in V\}$ (FEOFILOFF; KOHAYAKAWA; WAKABAYASHI,).

O Teorema 2.1.1 estabelece uma identidade fundamental relacionada ao grau dos vértices de um grafo G e o número de arestas m.

Teorema 2.1.1. *Para qualquer grafo G,*

$$\sum_{v \in V} d_G(v) = 2m$$

(BONDY; MURTY, 2008). □

De fato, se no grafo há m arestas, cada aresta incide em exatamente dois vértices aumentando assim o grau do grafo em duas unidades a cada aresta. Portanto a soma dos graus de G é $2m$.

Um *caminho* é um grafo cujos vértices podem ser dispostos em uma sequência linear, de tal forma que dois vértices são adjacentes se e somente se forem consecutivos na sequência. Um *ciclo* é um caminho fechado com três ou mais vértices, ou seja, um caminho onde os vértices inicial e final são adjacentes.

Um grafo é *conexo* se, para qualquer par de seus vértices u e v, existe um caminho com extremos u e v (FEOFILOFF; KOHAYAKAWA; WAKABAYASHI,). Caso contrário, ele é *desconexo*, ou seja, existe pelo menos um par de vértices que não estão ligados por nenhum caminho.

Em muitas aplicações em teoria dos grafos estamos interessados em determinar se um grafo possui certa propriedade. O Teorema 2.1.2 fornece uma condição suficiente para um grafo conter um ciclo.

Teorema 2.1.2. *Seja G um grafo em que todos os vértices têm grau no mínimo dois. Então, G contém um ciclo (BONDY; MURTY, 2008).* □

Um grafo é *acíclico* se não contém ciclo. A partir do Teorema 2.1.2 podemos verificar que um grafo acíclico deve ter pelo menos um vértice de grau menor que dois. De fato, todo grafo acíclico com $|V| > 1$, tem no mínimo dois vértices de grau menor que dois (BONDY; MURTY, 2008).

Um grafo H é um *subgrafo* de um grafo G, denotado por $H \subseteq G$, se $V_H \subseteq V_G$ e $E_H \subseteq E_G$ (HARJU, 2014). Um *subgrafo gerador* de um grafo G é um grafo obtido a partir de G por remoção somente de arestas, em outras palavras, um grafo cujo conjunto de vértices é o próprio conjunto $V(G)$. Se S é o conjunto de arestas removidas, este subgrafo de G é denotado por $G \setminus S$ (BONDY; MURTY, 2008).

2.1.1 Árvore Geradora

Um grafo conexo e acíclico é chamado de *árvore* (BONDY; MURTY, 2008). Considere um grafo G em que cada subgrafo conexo é uma árvore. Dessa forma, G é geralmente

chamado de *floresta*. O Teorema 2.1.3 formaliza a relação entre árvores e caminhos a partir de dois vértices.

Teorema 2.1.3. *Em uma árvore, quaisquer dois vértices são conectados por exatamente um caminho (BONDY; MURTY, 2008).* □

Se existissem dois pares de vértices conectados por mais de um caminho, teríamos um ciclo e, por definição não seria uma árvore, pois embora fosse conexo, não seria acíclico.

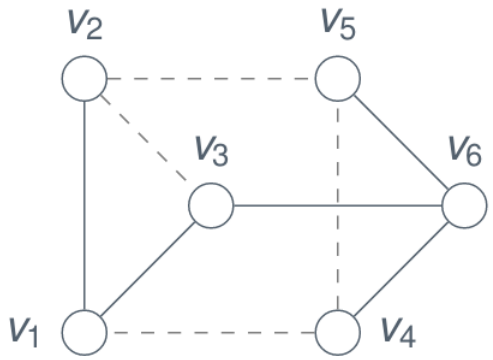
O Teorema 2.1.4 permite de forma exata calcular quantas arestas tem em uma árvore a partir do número de vértices.

Teorema 2.1.4. *Se T é uma árvore, então $m = n - 1$ (BONDY; MURTY, 2008).* □

Uma *subárvore* de um grafo G é um subgrafo que é uma árvore. Se este é um subgrafo gerador, ele é chamado de *árvore geradora*.

Na Figura 1 é mostrado um exemplo de um grafo G com árvore geradora T. As linhas pontilhadas e contínuas são as arestas do grafo original G, e apenas as linhas contínuas são as arestas da árvore geradora T de G.

Figura 1 – Árvore Geradora.



Fonte – Produzida pelo autor.

A remoção de uma aresta f compreende em simplesmente removê-la do grafo e é denotada por $G \backslash f$. A contração de uma aresta f compreende em removê-la e então criar um novo vértice w no qual todas as arestas incidentes nos extremos de f passam a incidir em w. Por fim, os extremos de f deixam de existir. Denotamos a contração de uma aresta f em um grafo G como G/f.

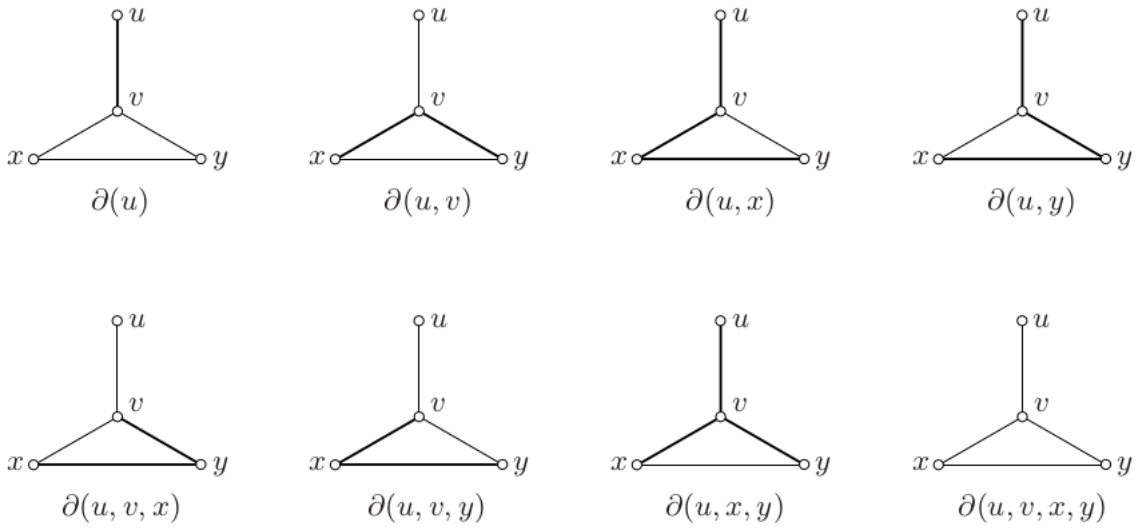
Denotamos o número de árvores geradoras de um grafo G por $t(G)$. Existe um teorema relacionando o número de árvores geradoras de um grafo G com o número de árvores geradoras de dois grafos $G\backslash f$ e G/f obtidos de G através de remoção e contração de uma aresta f (BONDY; MURTY, 2008). O Teorema 2.1.5 mostra como calcular o número de árvores geradoras em um grafo G.

Teorema 2.1.5. *Sejam G um grafo e f uma aresta de G. Então $t(G) = t(G\backslash f) + t(G/f)$ (BONDY; MURTY, 2008).* □

Uma solução possível para o problema da AGGMM seria obter todas as $t(G)$ árvores e verificar qual árvore possui o menor Δ. No PAGMS$_d$ poderíamos verificar qual minimiza a soma dos vértices que excedem d. No entanto, esta estratégia não é eficiente, pois o número de árvores possíveis é exponencial em relação ao número de vértices do grafo.

Sejam X e Y conjuntos de vértices (não necessariamente disjuntos) de um grafo $G = (V,E)$. Denotamos por $E[X,Y]$ o conjunto de arestas de G com um extremo em X e o outro extremo em Y, e por $e(X,Y)$ o número de arestas de G com um extremo em X e o outro extremo em Y. Se $Y = X$, nós simplesmente escrevemos $E(X)$ e $e(X)$ para $E[X,X]$ e $e(X,X)$, respectivamente. Quando $Y = V\backslash X$, o conjunto $E[X,Y]$ é chamado de *corte de arestas* de G associado ao X, ou a co-fronteira de X, e é denotado por $\partial(X)$; Exemplos de corte de arestas de um grafo são ilustrados na Figura 2.

Figura 2 – As arestas de corte de um grafo.



Fonte – (BONDY; MURTY, 2008).

As arestas em negrito para cada exemplo denotam as arestas que estão no corte. Por exemplo, na figura relacionada ao $\partial(u,v)$ as aresta do corte são as arestas (v,y) e (v,x).

O Teorema 2.1.6 apresenta uma generalização natural do Teorema 2.1.1, pois é o caso em que X = V.

Teorema 2.1.6. *Para qualquer grafo G e qualquer subconjunto X de V,*

$$|\partial(X)| = \sum_{v \in X} d_G(v) - 2e(X)$$

(BONDY; MURTY, 2008). □

2.2 Classes de complexidade

Desenvolver e criar algoritmos são de extrema importância na computação. Entretanto, existe uma preocupação adicional: codificar um algoritmo de forma que seu desempenho seja eficiente. Existem muitos problemas que conseguimos resolver de maneira ótima, por outro lado, existem muitos outros que ainda não foi possível obter solução ótima dentro de um tempo limitado por um polinômio. Desse modo, estudar a complexidade de algoritmos é de grande importância para resolução de problemas.

Uma classe de complexidade é um conjunto de problemas que estão relacionados aos recursos computacionais baseados em tempo e/ou espaço. A classe P consiste nos problemas que podem ser resolvidos em tempo polinomial. Mais especificamente, são problemas que podem ser resolvidos no tempo $O(n^k)$ para alguma constante k, onde n é o tamanho da entrada para o problema (CORMEN et al., 2002). A classe NP consiste nos problemas que são "verificáveis" em tempo polinomial. Em outras palavras, se tivéssemos de algum modo um "certificado" para uma entrada do problema, poderíamos verificar se o certificado é correto em tempo polinomial no tamanho da entrada do problema. Por exemplo, no problema do ciclo hamiltoniano, dado um grafo $G = (V,E)$ um certificado seria uma sequência $(v_1, v_2, v_3, ..., v_{|V|})$ de $|V|$ vértices. É fácil verificar em tempo polinomial que $(v_i, v_{i+1}) \in E$ para $i = 1,2,3,...,|V|-1$ e também que $(v_{|V|}, v_1) \in E$ (CORMEN et al., 2002).

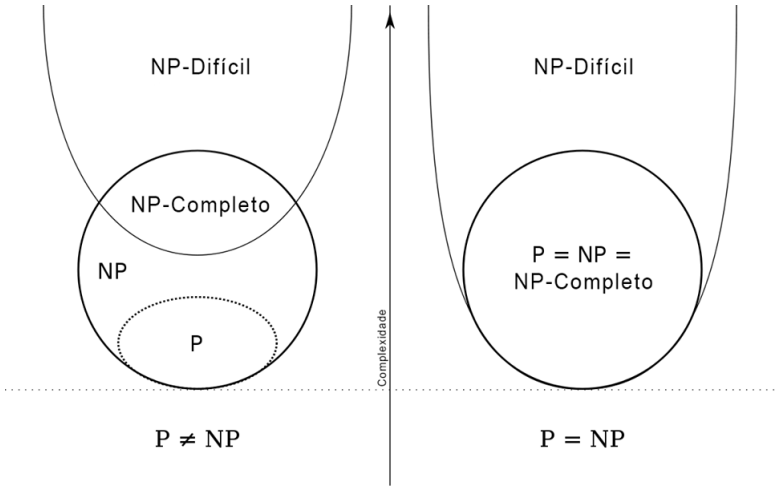
Qualquer problema em P também está em NP, tendo em vista que, se um problema está em P então podemos resolvê-lo em tempo polinomial sem sequer receber um certificado. Alguns problemas em NP possuem complexidade individual relacionada àquela da classe inteira. Dessa forma, se existir um algoritmo de tempo polinomial para quaisquer desses problemas,

todos os problemas em NP seriam solúveis em tempo polinomial. Esses problemas são chamados de NP-completos (SIPSER, 2007).

Ainda não foi descoberto nenhum algoritmo de tempo polinomial para um problema NP-completo, tampouco provou-se que não pode existir nenhum algoritmo de tempo polinomial para qualquer um deles (CORMEN et al., 2002).

Existem problemas que não estão em NP tal que todos os problemas da classe NP se reduzem a eles, tais problemas constituem a classe NP-difícil. Alguns problemas NP-difícil não são verificáveis em tempo polinomial, ou seja, dado um certificado de um solução, não é possível verificar se o certificado é correto em tempo polinomial. Caso um problema na classe NP-difícil também esteja em NP, então ele também pertence a classe de problemas NP-completos. Assim, todos os problemas NP-completos são igualmente difíceis de resolver, uma vez que são inter-redutíveis. Se houver um algoritmo de tempo polinomial para qualquer problema NP-completo, então P = NP e cada problema NP pode ser resolvido em tempo polinomial. Apesar dos enormes esforços, a questão de saber se P = NP ainda não foi respondida. A conjectura em vigor é a de que P $\neq$ NP (KANN, 1992).

Figura 3 – P = NP ou P $\neq$ NP ?



Fonte – Produzida pelo autor.

A Figura 3 retrata os casos em que P$\neq$NP e P = NP. No primeiro caso, nem todos os problemas da classe NP podem ser resolvidos em tempo polinomial. No segundo caso, conseguimos obter um algoritmo que retorna a solução ótima em tempo polinomial para qualquer problema daquela classe.

2.3 Algoritmos aproximativos

Muitos problemas de significado prático são NP-completos, mas são importantes demais para serem abandonados apenas porque é desconhecida uma forma de se obter uma solução ótima em tempo hábil (CORMEN et al., 2002). Existem várias abordagens para contornarmos o NP-completo, uma delas é através de soluções aproximadas em tempo polinomial.

Os algoritmos *aproximativos* são algoritmos polinomiais que buscam sacrificar o mínimo possível da qualidade, obtida nos métodos exatos, ganhando, simultaneamente, o máximo possível em eficiência (tempo polinomial). A busca do equilíbrio entre estas situações conflitantes é o grande paradigma dos algoritmos aproximativos.

Williamson e Shmoys (2011) afirmam que um algoritmo de α-aproximação para um problema de otimização é um algoritmo de tempo polinomial que para todas as instâncias do problema produz uma solução cujo valor está dentro de um fator α do valor de uma solução ótima.

3 TRABALHOS RELACIONADOS

Neste Capítulo é apresentado alguns trabalhos relacionados e um algoritmo aproximativo para AGGMM. Algumas definições serão mostradas bem como um fator de aproximação juntamente com o pseudocódigo do algoritmo.

Em Könemann e Ravi (2000) é apresentado um algoritmo de aproximação bicrítica para o Problema da Árvore Geradora Mínima de Grau com Limite Mínimo. Neste problema, é dado um grafo que é uma árvore geradora de custo mínimo T com o grau máximo no máximo B^*. Em um grafo com n vértices, este algoritmo encontra uma árvore geradora com gau máximo $O(B^* + \log n)$ e custo $O(opt_{B^*})$ onde opt_{B^*} é o custo mínimo de qualquer árvore geradora cujo grau máximo é no máximo B^*. Este algoritmo usa ideias da dualidade Lagrangeana, e mostra como um conjunto de multiplicadores lagrangeanos ótimos produz limites tanto no grau quanto no custo da solução calculada.

Goemans (2006) considera o Problema da Árvore Geradora Mínima sob a restrição de que todos os graus devem ser no máximo um dado valor k. É mostrado que de forma eficientemente podemos encontrar uma árvore geradora de grau no máximo $k+2$ cujo custo é no máximo o custo de uma árvore geradora ótima de grau máximo no máximo k. Esta abordagem usa uma sequência de argumentos algébricos, poliédricos e combinatórios.

Raidl (2000) em seu trabalho propõe uma nova técnica de representação e operadores adequados de variação para o problema da árvore geradora mínima com restrição de grau. Para um grafo ponderado e não-direcionado $G = (V,E)$, este problema procura identificar a menor árvore geradora cujo grau dos vértices não exceda um limite superior de $d \geq 2$. Busca-se computar uma árvore geradora de grau máximo mínimo. A técnica utilizada neste trabalho usa algoritmos evolucionários bem como conceitos de mutação e *cross-over*.

3.1 Approximating the Minimum-Degree Steiner Tree to within One of Optimal Furer e Raghavachari (1994)

Nesta Seção é proposto um algoritmo de aproximação para AGGMM com uma boa garantia de aproximação. O Teorema 3.1.1 formaliza o comportamento do algoritmo.

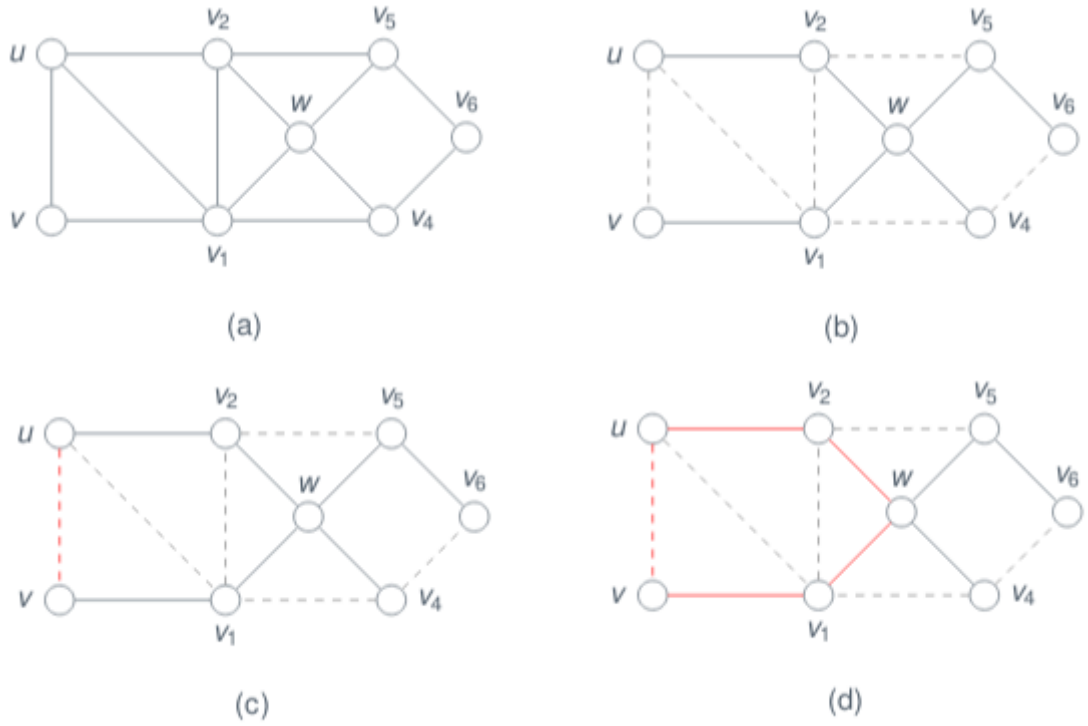
Teorema 3.1.1. *Seja $G = (V,E)$ um grafo. Seja Δ^* o valor ótimo de uma solução de G. Existe um algoritmo de aproximação de tempo polinomial para o problema de árvore geradora com*

grau máximo mínimo que encontra uma árvore geradora com grau no máximo $\Delta^* + 1$ *(FURER; RAGHAVACHARI, 1994).* □

No que diz respeito ao algoritmo, ele começa com uma árvore arbitrária T de G e tenta reduzir seu grau com uma série de melhoramentos que são definidos da seguinte maneira: seja (u,v) uma aresta de G que não está em T. Seja C o único ciclo gerado quando (u,v) é adicionado a T. Suponha que há um vértice w de grau k em C, onde k é o grau máximo da árvore, enquanto os graus dos vértices u e v são no máximo $k-2$. Um *melhoramento* em T é uma modificação de T adicionando a aresta (u,v) em T e removendo uma das arestas em C incidente em w. Em tal melhoria, dizemos que w se *beneficia* de (u,v) (FURER; RAGHAVACHARI, 1994). Dizemos também que w é um vértice *redutível* via (u,v). Perceba que após o melhoramento, o vértice w passou a ter grau $k-1$ e os vértices u e v grau no máximo $k-1$ reduzindo assim o grau de um vértice ruim. Um vértice *ruim*, é um vértice de grau k e portanto deve ser reduzido o máximo possível pelo algoritmo. Dada um aresta que não está no grafo, o critério para a não ocorrência de um melhoramento usando aquela aresta acontece da seguinte forma: considere T uma árvore geradora de grau k. Seja $d_T(u)$ o grau do vértice $u \in T$. Seja $(u,v) \notin T$ uma aresta em G. Suponha que w seja um vértice de grau k no ciclo gerado pela adição de (u,v) em T. Se $d_T(u) \geq k-1$, dizemos que *u bloqueia w* de (u,v) (FURER; RAGHAVACHARI, 1994). Dizemos também que w é um vértice *não redutível* via (u,v) e portanto, não é possível fazer o melhoramento usando aquela aresta.

As figuras 4 e 5 mostram como ocorre o melhoramento para efeitos de exemplo. A Figura 4(a) mostra o grafo original. A Figura 4(b) a árvore geradora T obtida do grafo em 4(a) onde as linhas pontilhadas são arestas que não estão no grafo e as linhas contínuas as arestas da árvore. A Figura 4(c) mostra que é escolhida uma aresta (u,v) que não está no grafo indicada como uma linha pontilhada em vermelho. A Figura 4(d) mostra o ciclo gerado ao adicionar (u,v) no grafo.

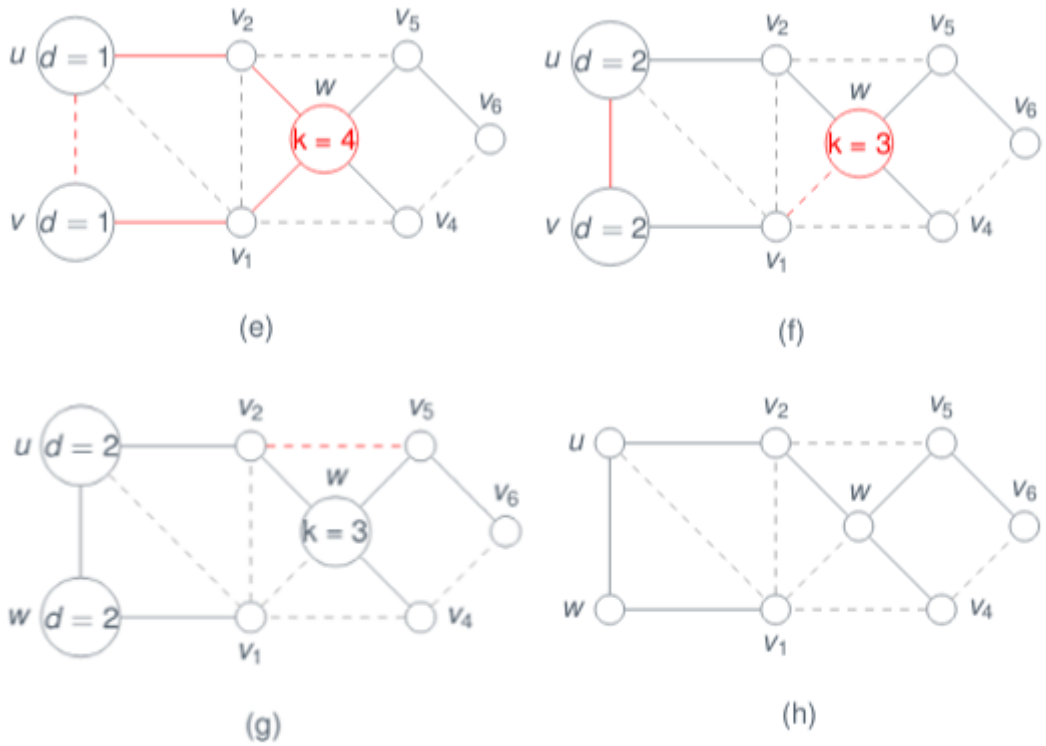
Figura 4 – Exemplo de um melhoramento.



Fonte – Produzida pelo autor.

A Figura 5 mostra a continuação do processo de melhoramento. Na Figura 5(e) verificamos que existe um vértice w onde $d_T(w) = k = 4$, e uma aresta (u,v), onde $d(u)$ e $d(v)$ são no máximo $k-2 = 4-2 = 2$. Na Figura 5(f) adicionamos a aresta (u,v) na árvore e removemos alguma aresta do ciclo incidente em w diminuindo assim o grau máximo da árvore. As figuras 5(g) e 5(h) ilustram respectivamente a nova árvore após o melhoramento mostrando os graus dos vértices usados no melhoramento, e a apenas a imagem da árvore.

Figura 5 – Exemplo de um melhoramento (continuação).



Fonte – Produzida pelo autor.

Perceba que na Figura 5(g) há uma linha/aresta pontilhada e com a cor vermelha. Esta aresta é um exemplo de bloqueio, onde o vértice v_2 bloqueia w de (v_2, v_5), pois $d_T(w) = 3$ e $d_T(v_2) = 2$ e portanto não é possível ocorrer um melhoramento.

Durante o algoritmo, não só o melhoramento tem importância mas a forma como o melhoramento é usado. Assim, a principal fase do algoritmo é aquela aonde não só aplicamos um melhoramento mas também quando propagamos um melhoramento. A fase de propagação funciona da seguinte maneira: seja w um vértice redutível via (u, v). Seja C o único ciclo gerado adicionando (u, v) em T. A intuição seria aplicar apenas um melhoramento, ou seja, ao adicionar (u, v) removendo alguma aresta do ciclo que incidisse em w para assim o grau daquele vértice ser melhorado. Porém, antes deste processo, é feito uma propagação de melhoramentos, ou seja, é verificado se o vértice u e/ou v é redutível via alguma aresta. Suponha sem perda de generalidade que o vértice u seja redutível via uma aresta (x, y). É aplicado um melhoramento nesta fase e verificado de forma análoga se x e/ou y são redutíveis. No final, quando todas as verificações e melhoramentos forem executados, a arestas (u, v) é adicionada, e alguma aresta em C incidente em w é removida.

O pseudocódigo mostrado na Figura 6 descreve com mais detalhes o funcionamento

do algoritmo para AGGMM.

Figura 6 – Pseudocódigo do algoritmo para AGGMM descrito em Furer e Raghavachari (1994).

Algoritmo 1: *Algoritmo $\triangle^* + 1$ para árvores geradoras.*

Entrada: Um grafo G

Saída: Uma árvore T que se aproxima de uma AGGMM

1 Encontre uma árvore geradora T de G. Seja k o seu grau máximo.

2 Marque os vértices de grau k e $k-1$ como ruins. Remova esses vértices de T gerando uma floresta. Marque todos os outros vértices como bons. Seja F o conjunto de componentes conexos na floresta.

3 **enquanto** *existir uma aresta (u,v) conectando dois componentes diferentes de F e todos os vértices de grau k estiverem marcados como ruins* **faça**

4 Encontre os vértices ruins no ciclo C gerado por T juntamente com (u,v) e marque-os como bons.

5 Atualize F ao combinar os componentes ao longo do ciclo C e esses vértices recém-marcados em um único componente. Observe que mais de dois componentes de F podem ser combinados em um nesta etapa.

6 **fim**

7 **se** *existir um vértice w de grau k marcado como bom* **então**

8 Encontre uma sequência de melhorias que se propagam para w e atualize T (e, se necessário, k) e volte para a Etapa 2.

9 **fim**

10 Retorne a árvore final T.

Fonte – Produzida pelo autor.

No Capítulo seguinte, é apresentado o algoritmo para o problema proposto neste trabalho, o PAGMS$_d$.

4 ALGORITMO PROPOSTO

Neste Capítulo são apresentados os passos para o desenvolvimento do algoritmo. Na Seção 4.1 é dada uma descrição do algoritmo e apresentado seu pseudocódigo. Na Seção 4.2 é apresentado um limite inferior para o PAGMS$_d$ bem como a demonstração.

4.1 Algoritmo

O algoritmo começa com uma árvore arbitrária T de G e tenta reduzir seu grau com uma série de melhoramentos que é definido da seguinte maneira: seja (u,v) uma aresta de G que não está em T. Seja C o único ciclo gerado quando (u,v) é adicionado a T. Suponha que há um vértice w de grau maior que d em C enquanto os graus dos vértices u e v são no máximo $d-1$. Um melhoramento em T é uma modificação de T adicionando a aresta (u,v) em T e removendo uma das arestas em C incidente em w. Em tal melhoria, dizemos que w se beneficia de (u,v). Dizemos também, que w é um vértice redutível via (u,v). Perceba que após o melhoramento, o vértice w passou a ter grau reduzido em uma unidade e os vértices u e v grau no máximo d reduzindo assim o grau de um vértice ruim e consequentemente a soma dos excessos. Um vértice ruim neste caso, é um vértice de grau maior que d e portanto deve ser reduzido o máximo possível pelo algoritmo para que a soma seja minimizada.

Seja $d_T(u)$ o grau do vértice $u \in T$. Seja $(u,v) \notin T$ uma aresta em G. Suponha que w seja um vértice de grau maior que d no ciclo gerado pela adição de (u,v) em T. Se $d_T(u) \geq d$, dizemos que u bloqueia w de (u,v). Dizemos também que w é um vértice não redutível via (u,v).

Durante o algoritmo, não só o melhoramento tem importância mas a forma como o melhoramento é usado. Assim, a principal fase do algoritmo é aquela aonde não só aplicamos um melhoramento mas também quando propagamos um melhoramento.

A fase de propagação funciona da seguinte maneira: seja w um vértice redutível via (u,v). Seja C o único ciclo gerado adicionando (u,v) em T. A intuição seria aplicar um melhoramento, ou seja, ao adicionar (u,v) removendo alguma aresta do ciclo que incidisse em w para assim o grau daquele vértice ser melhorado. Porém, antes deste processo, é feito uma propagação de melhoramentos, ou seja, é verificado se o vértice u e/ou v é redutível via alguma aresta. Suponha sem perda de generalidade que o vértice u seja redutível via uma aresta (x,y). É aplicado um melhoramento nesta fase e verificado de forma análoga se x e/ou y são redutíveis. No final, quando todas as verificações e melhoramentos forem executados, a arestas (u,v) é

adicionada, e alguma aresta em C incidente em w é removida.

Por fim, para cada aresta e pertencente ao conjunto de arestas que não estão na árvore e que são incidentes no corte do conjunto D. Considere o ciclo C criado pela adição de e em T. Caso exista uma aresta f em C onde ambos os extremos tenham grau maior que d, então removemos f e adicionamos e na árvore. Esta fase consiste em um melhoramento final da árvore para retirada de arestas que tornam a solução pior.

O pseudocódigo da Figura 7 descreve com mais detalhes o funcionamento do algoritmo para o PAGMS$_d$. Como o algoritmo proposto se baseia no algoritmo descrito em (FURER; RAGHAVACHARI, 1994), usamos o mesmo pseudocódigo e mostramos o que muda no pseudocódigo. As partes do código que estão sublinhadas são partes onde não são úteis para nosso problema, e portanto, são excluidas. As partes de código em negrito são as partes que foram adicionadas ao código ou substituídas por alguma linha de código sublinhada. Por fim, da linha 10 a linha 15 foi adicionado uma fase a mais no algoritmo.

Figura 7 – Pseudocódigo para o PAGMS$_d$.

Algoritmo 2: *Algoritmo para o PAGMS$_d$.*

Entrada: Um grafo G **e um inteiro** d

Saída: Uma árvore T de G que se aproxima de uma AGGMM. **Uma árvore geradora** T **de** G **que minimiza a soma dos excessos.**

1 Encontre uma árvore geradora T de G. Seja k o seu grau máximo.

2 Marque os vértices de grau k e $k-1$ **maior ou igual a** d como ruins. Remova esses vértices de T gerando uma floresta. Marque todos os outros vértices como bons. Seja F o conjunto de componentes conexos na floresta.

3 **enquanto** *existir uma aresta* (u,v) *conectando dois componentes diferentes de* F *e todos os vértices de grau* k ***maior ou igual a*** d *estiverem marcados como ruins* **faça**

4 Encontre os vértices ruins no ciclo C gerado por T juntamente com (u,v) e marque-os como bons.

5 Atualize F ao combinar os componentes ao longo do ciclo C e esses vértices recém-marcados em um único componente. Observe que mais de dois componentes de F podem ser combinados em um nesta etapa.

6 **fim**

7 **se** *existir um vértice* w *de grau* k ***maior ou igual a*** d *marcado como bom* **então**

8 Encontre uma sequência de melhorias que se propagam para w e atualize T e volte para a Etapa 2.

9 **fim**

10 **para cada** *aresta* $e \in E(G) \backslash E(T)$ *incidente em* $\partial(D_d^+ \cup D_d)$ **faça**

11 Seja C o ciclo criado ao adicionar e em T.

12 **se** *existe uma aresta* $f = (u,v) \in C$ *com* $d_T(v) > d$ *e* $d_T(u) > d$ **então**

13 Faça com que T seja $T + e \backslash f$. Atualize o conjunto de vértices de grau maior ou igual a d.

14 **fim**

15 **fim**

16 Retorne a árvore final T.

Fonte – Produzida pelo autor.

4.2 Limite inferior para PAGMS$_d$

Nesta Seção é apresentada a demonstração que o algoritmo proposto possui um limite inferior. O Teorema 4.2.1 apresenta uma propriedade útil para a demonstração do Teorema 4.2.2.

Considere ALG e ALG_T a soma dos excessos e a árvore respectivamente obtidas a partir do PAGMS$_d$ e OPT, OPT_T a soma dos excessos e a árvore de uma solução ótima respectivamente.

Teorema 4.2.1. *Seja T uma árvore obtida através do ALG_T. Seja a e b o número de vértices com grau igual a d não redutíveis e o número de vértices com grau maior que d em T, respectivamente. Então, $OPT \geq ALG - (a+b-1)$.*

Demonstração. Considere $D = \{v : d_T(v) > d \text{ ou } d_T(v) = d\}$. Seja F as arestas de T incidentes nos vértices de D. Seja C o conjunto de componentes conexas formadas removendo F de T. Uma vez que toda solução viável(incluindo a ótima) é uma árvore geradora e não há arestas no grafo original G conectando duas componentes de C, é necessário pelo menos $|F|$ arestas para conectar todas as componentes de C e os vértices em D. Dessa maneira,

$$\sum_{v \in D} d_{OPT}(v) \geq |F|.$$

O somatório da equação acima calcula a soma dos graus dos vértices que estão no conjunto D. Logo, se um vértice tem grau d é contabilizado d no somatório. Da mesma forma se o vértice tem grau $d+k$ com $k > 0$ onde k é um inteiro, no somatório é contabilizado $d+k$. Porém, queremos apenas a soma dos excessos, ou seja, se o vértice tem grau d contabilizamos 0 no somatório, pois não há excesso, e se for $d+k$ deve ser contabilizado apenas k. Dessa forma, como a é o número de vértices com grau igual a d, e b o número de vértices com grau maior que d, basta subtrair do somatório $d(a+b)$ para se obter apenas o excesso. Para a desigualdade acima continuar, subtraímos $d(a+b)$ em ambos os lados e dessa maneira temos que:

$$\sum_{v \in D} d_{OPT}(v) - d(a+b) \geq |F| - d(a+b).$$

Note que $OPT \geq \sum_{v \in D} d_{OPT}(v) - d(a+b) \geq |F| - d(a+b)$, portanto por transitividade,

$$OPT \geq |F| - d(a+b). \tag{4.1}$$

Por outro lado,

$$ALG = \sum_{v \in D} d_{ALG}(v) - d(a+b). \tag{4.2}$$

Ou seja, a soma dos excessos retornada por ALG é exatamente o somatório do grau de todos os vértices do conjunto D menos $d(a+b)$ para apresentar apenas o excesso.

Do Teorema 2.1.6 temos que:

$$|\partial(D)| = \sum_{v \in D} d_{ALG}(v) - 2e(D).$$

Sabemos também que:

$$|F| = e(D) + |\partial(D)|. \tag{4.3}$$

Perceba que o conjunto F são as arestas de T que são incidentes no conjunto D. Logo, $|F|$ equivale ao número de arestas que incidem ambos os extremos nos vértices de D denotado por $e(D)$ mais as arestas onde apenas um extremo incide em D que é $|\partial(D)|$.

Substituímos $|\partial(D)|$ pelo seu valor equivalente mostrado no Teorema 2.1.6 em (4.3) temos que:

$$\begin{aligned} |F| &= e(D) + \sum_{v \in D} d_{ALG}(v) - 2e(D) \\ |F| &= \sum_{v \in D} d_{ALG}(v) + e(D) - 2e(D) \\ |F| &= \sum_{v \in D} d_{ALG}(v) - e(D). \end{aligned} \tag{4.4}$$

Como $e(D) \leq (a+b-1)$, se substituirmos $e(D)$ por $(a+b-1)$ em (4.4) obtemos a seguinte desigualdade:

$$|F| \geq \sum_{v \in D} d_{ALG}(v) - (a+b-1). \tag{4.5}$$

De (4.1) temos que:

$$OPT \geq |F| - d(a+b).$$

De (4.2),

$$ALG = \sum_{v \in D} d_{ALG}(v) - d(a+b),$$

isolando $d(a+b)$ temos que,

$$d(a+b) = \sum_{v \in D} d_{ALG}(v) - ALG. \tag{4.6}$$

Substituindo (4.6) em (4.1),

$$\begin{aligned} OPT &\geq |F| - \left(\sum_{v \in D} d_{ALG}(v) - ALG \right) \\ OPT &\geq |F| - \sum_{v \in D} d_{ALG}(v) + ALG. \end{aligned} \tag{4.7}$$

Usando (4.5) em (4.7),

$$OPT \geq \sum_{v \in D} d_{ALG}(v) - (a+b-1) - \sum_{v \in D} d_{ALG}(v) + ALG.$$

Finalmente, concluimos que $OPT \geq ALG - (a+b-1)$.

□

O Teorema 4.2.1 nos auxília a apresentar um fator aditivo de aproximação para o $PAGMS_d$ e o Teorema 4.2.2, mostrado a seguir, apresenta esse fator de aproximação.

Teorema 4.2.2. *Seja T árvore obtida de ALG. Seja a' o número de vértices com grau igual a d redutível, e s a soma dos graus dos vértices com grau menor ou igual a d redutível em T. Então,*

$$ALG - OPT \leq \left\lfloor \frac{n-2-(d-1)(a'+1)}{d} \right\rfloor.$$

Demonstração. Uma vez que $OPT \geq 0$, temos que $ALG - OPT \leq ALG$. Portanto,

$$ALG - OPT \leq \min\{(a+b-1), ALG\}. \tag{4.8}$$

Seja a' o número de vértices com grau igual a d redutíveis, e s a soma dos graus dos vértices com grau menor ou igual a d redutíveis em T. A soma dos graus dos vértices com grau maior que d podem ser obtidas por $ALG + db$. Aqui há uma sutileza pois ALG é a soma apenas dos excessos e não do grau inteiro dos vértices. Logo ao adicionarmos db obtemos a soma completa de cada vértice.

Um vez que cada vértice tem no mínimo grau um, $s \geq n - a - a' - b$. De fato, se diminuirmos do número total de vértices n, o número de vértices com grau igual a d redutíveis a', e os com grau igual a d não redutíveis a, juntamente com os vértices com grau maior que d, restarão apenas os vértices com grau menor que d. Como cada vértice tem no mínimo grau

um, a soma dos vértice com grau menor que d será no mínimo $n-a-a'-b$ que é o número de vértices com grau menor que d.

Do Teorema 2.1.1 a soma dos graus de um grafo é $2m$ onde m é o número de arestas. Como lidamos aqui com árvores geradoras, temos a partir do Teorema 2.1.4 que $m = n-1$, onde n é o número de vértices, portanto,

$$\begin{aligned} 2m &= da + da' + (ALG + db) + s \\ 2(n-1) &= da + da' + (ALG + db) + s \end{aligned} \tag{4.9}$$

De fato a soma dos graus é igual ao número de vértices com grau igual a d multiplicado por d, $da + da'$ mais a soma dos graus dos vértices maiores que d, $ALG + db$, e a soma do grau dos vértices com grau menor que d que é s.

Substituindo $s \geq n-a-a'-b$ em (4.9),

$$\begin{aligned} da + da' + (ALG + db) + s &\geq da + da' + (ALG + db) + (n-a-a'-b) = \\ da + da' + (ALG + db) + (n-a-a'-b) &= da - a + da' - a' + db - b + ALG + n \\ &= a(d-1) + a'(d-1) + b(d-1) + ALG + n \\ &= (d-1)(a + a' + b) + ALG + n. \end{aligned}$$

Então,

$$\begin{aligned} 2(n-1) &\geq (d-1)(a + a' + b) + ALG + n \\ 2(n-1) - ALG - n &\geq (d-1)(a + a' + b) \\ 2n - 2 - ALG - n &\geq (d-1)(a + a' + b) \\ n - 2 - ALG &\geq (d-1)(a + a' + b) \\ \frac{n-2-ALG}{d-1} &\geq (a + a' + b) \\ \frac{n-2-ALG}{d-1} + a' &\geq a + b. \end{aligned}$$

Invertendo os termos,

$$a + b \leq \frac{n-2-ALG}{d-1} + a'. \tag{4.10}$$

De (4.8),

$$ALG - OPT \leq \min\{(a+b-1), ALG\}.$$

Substituindo $(a+b)$ de (4.10) em (4.8),

$$\begin{aligned} ALG - OPT &\leq \min\left\{\frac{n-2-ALG}{d-1} + a' - 1, ALG\right\} \\ &\leq \min\left\{\frac{n-2-ALG}{d-1} - (a'+1), ALG\right\}. \end{aligned} \tag{4.11}$$

Como o termo à esquerda está diminuindo em função de ALG, e o lado direito está crescendo, obtemos o pior caso igualando os dois termos:

$$\frac{n-2-ALG}{d-1} - (a'+1) = ALG.$$

Colocando os termos do lado esquerdo em um denominador comum,

$$\begin{aligned} \frac{n-2-ALG-(d-1)(a'+1)}{d-1} &= ALG \\ n-2-ALG-(d-1)(a'+1) &= (d-1)ALG \\ n-2-(d-1)(a'+1) &= (d-1)ALG + ALG \\ n-2-(d-1)(a'+1) &= dALG - ALG + ALG \\ n-2-(d-1)(a'+1) &= dALG \\ \frac{n-2-(d-1)(a'+1)}{d} &= ALG \\ ALG &= \frac{n-2-(d-1)(a'+1)}{d} \\ ALG - OPT &\leq \left\lfloor \frac{n-2-(d-1)(a'+1)}{d} \right\rfloor. \end{aligned}$$

□

Como nossos testes se concentram em $d = 3$, temos que:

$$\begin{aligned} ALG - OPT &\leq \left\lfloor \frac{n-2-(3-1)(a'+1)}{3} \right\rfloor \\ &\leq \left\lfloor \frac{n-2a'-4)}{3} \right\rfloor \\ &\leq \left\lfloor \frac{n}{3} - \frac{2a'+4}{3} \right\rfloor. \end{aligned} \tag{4.12}$$

Em (4.12) temos um limite inferior para o PAGMS$_d$ quando $d = 3$. Portanto, nosso algoritmo retorna a soma dos excessos no pior caso aproximadamente a terça parte do número de vértices em relação a solução ótima. Suponha que temos um grafo com n = 300 e um a' = 40. A soma do nosso algoritmo é maior em 72 unidades da solução ótima no pior caso.

O Capítulo 5 apresenta todos os resultados obtidos dos algoritmos mostrados nos capítulos 3 e 4.

5 RESULTADOS

Neste Capítulo serão apresentados os resultados obtidos através de testes realizados com os algoritmos descritos nos capítulos 4 e 3. Na Seção 5.1 é mostrado como foram geradas as instâncias. Na Seção 5.2 são mostrados os resultados dos algoritmos sobre as instâncias geradas.

5.1 Gerador de instâncias

O gerador de instâncias para a realização dos testes foi criado usando a linguagem de programação Python versão 3.6.0, usando a biblioteca NetworkX (NetworkX, 2005). As instâncias para o PAGMS$_d$ são compostas por grafos aleatórios e grafos que tenham árvores geradoras com limite inferior de grau máximo mínimo previamente conhecido para efeitos de teste.

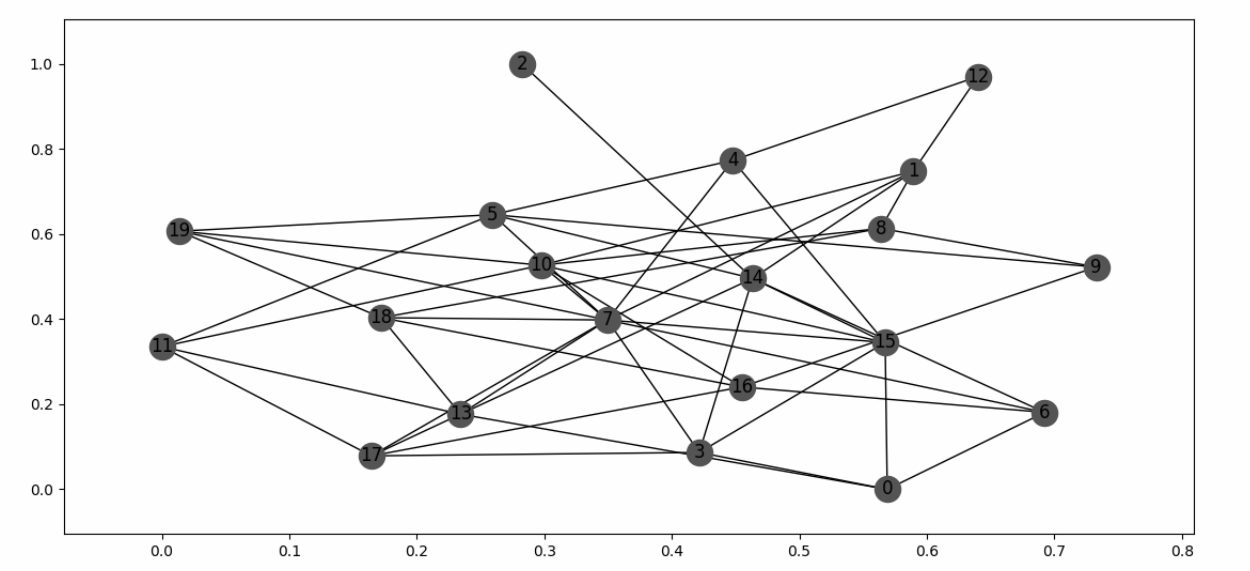
Dentro do pacote NetworkX, podemos utilizar funções que geram grafos aleatórios. Sua interface possui várias funções que ajudam no desenvolvimento de instâncias de grafos com diversas propriedades. A principal função usada é a função **fast_gnp_random_graph***(n, p, seed=None, directed=False)* onde retorna um grafo aleatório $G_{n,p}$ conhecido como Erdős-Rényi ou grafo *binomial*. Usamos apenas os parâmetros n e p, onde n é o número de vértices do grafo e p é a probabilidade de uma aresta estar no grafo. Os outros dois argumentos são opcionais, que são: *seed* (semente para gerador de números aleatórios) e $directed$ (para informar se o grafo é direcionado ou não). O algoritmo define quais das $\frac{n\cdot(n-1)}{2}$ possíveis arestas com probabilidade p irão ser inseridas no grafo.

Além de grafos aleatórios, a outra classe de instância deve ter a propriedade de que toda árvore geradora tenha grau máximo maior ou igual a um certo valor k. Dessa forma, gerar grafos aleatórios não nos dá essa garantia. Assim, com o auxílio da função citada anteriormente, usamos a seguinte estratégia: criamos uma função que gera uma árvore geradora com o formato de estrela com k vértices, logo haverá um vértice central v' onde todos os outros $k-1$ vértices serão adjacentes a ele. Depois, criamos um grafo binomial com os $n-k$ vértices restantes. Por fim, adicionamos uma aresta da componente estrela a outra componente gerada aleatoriamente. A componente estrela é uma árvore com grau máximo $k-1$ e nossa tarefa é garantir que o grafo resultante da união das duas componentes tenha grau máximo igual a k, e portanto a adição de uma aresta não pode ser aleatória. A aresta que conecta as componentes deve possuir como um dos extremos o vértice central v' da estrela com grau $k-1$, garantindo assim que o grafo possua

grau máximo maior ou igual a k com a adição da aresta.

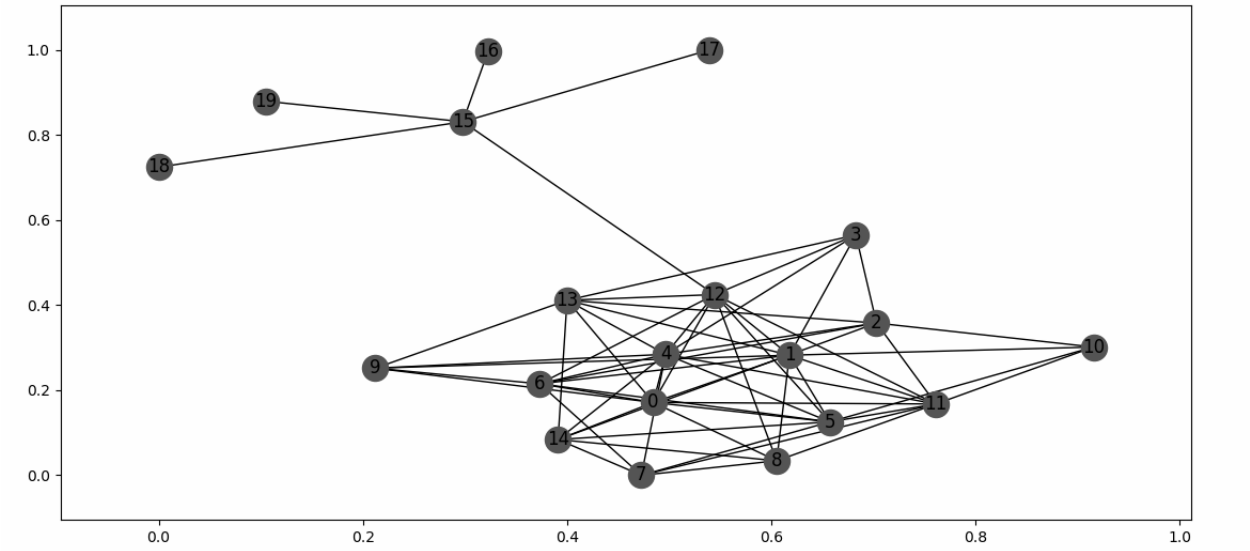
A Figura 8 mostra um exemplo de grafo aleatório com 20 vértices e com a probabilidade de 30%. A Figura 9 apresenta um exemplo de grafo onde a soma dos excessos considerando $d = 3$ será no mínimo 2, pois é certo de que toda árvore geradora obtida terá grau no mínimo 5, pois $k = 5$. Além disso, o exemplo possui 20 vértices e probabilidade de 50%.

Figura 8 – Grafo Aleatório com $n = 20$ e $p = 30\%$.



Fonte – Produzida pelo autor.

Figura 9 – Grafo aleatório com estrela com $n = 20$, $p = 50\%$ e $k = 5$.



Fonte – Produzida pelo autor.

5.2 Testes

Os testes para os algoritmos foram realizados em um computador com 8GB de memória RAM, processador Intel(R) Xeon(R) CPU E31240 @ 3.30GHz e Sistema Operacional Ubuntu 16.04.3 LTS.

Os resultados são em relação ao valor da soma dos excessos retornados por ambos os algoritmos e o tempo de resposta em segundos onde é calculado a média apresentada por AM e o desvio padrão apresentado por STD. Cada tipo de instância é executada 10 vezes por cada algoritmo para calcular a média e o desvio padrão.

Foi adicionado uma fase a mais nos algoritmos uma vez que ambos apenas retornam árvores geradoras e se faz necessário saber qual dos algoritmos retorna melhor soma dos excessos. Dessa maneira dado uma árvore, existe uma função que recebe um valor $d = 3$ e para cada vértice v de T, se o seu grau for maior que d, uma variável acumuladora recebe $d - v$. No final, a soma dos excessos é retornada.

5.2.1 Grafos aleatórios

Da Tabela 1 à Tabela 4 são usados grafos aleatórios com variação de número de vértices (n) e variação de densidade, ou seja, número de arestas.

Tabela 1 – Resultados do Grupo 1.

Grupo G_1		PAGMS$_d$				Furer			
		Soma dos excessos		Tempo		Soma dos excessos		Tempo	
n	Densidade	AM	STD	AM	STD	AM	STD	AM	STD
100	10%	0	0	0.9	0.316	0	0	1.7	0.483
	30%	0	0	2.8	0.421	0	0	4.7	0.483
	50%	0	0	4.9	0.737	0	0	8.4	0.516
	70%	0	0	8.0	0.471	0	0	13.6	0.516

Fonte – Elaborado pelo autor.

Tabela 2 – Resultados do Grupo 2.

Grupo G2		PAGMS$_d$				Furer			
		Soma dos excessos		Tempo		Soma dos excessos		Tempo	
n	Densidade	AM	STD	AM	STD	AM	STD	AM	STD
150	10%	0	0	4.8	0.788	0	0	8.7	1.159
	30%	0	0	13.9	0.994	0	0	25.0	1.333
	50%	0	0	26.6	1.349	0	0	49.1	1.911
	70%	0	0	41.7	1.337	0	0	79.9	2.183

Fonte – Elaborado pelo autor.

Tabela 3 – Resultados do Grupo 3.

Grupo G_3		*PAGMS$_d$*				Furer			
		Soma dos excessos		Time		Soma dos excessos		Time	
n	Densidade	AM	STD	AM	STD	AM	STD	AM	STD
200	10%	0	0	16.0	2.357	0	0	29.5	2.635
	30%	0	0	47.4	2.270	0	0	88.0	3.972
	50%	0	0	87.4	3.098	0	0	171.4	5.146
	70%	0	0	138.4	7.676	0	0	281.0	5.734

Fonte – Elaborado pelo autor.

Tabela 4 – Resultados do Grupo 4.

Grupo G_4		*PAGMS$_d$*				Furer			
		Soma dos excessos		Tempo		Soma dos excessos		Tempo	
n	Densidade	AM	STD	AM	STD	AM	STD	AM	STD
300	10%	0	0	81.1	10.461	0	0	161.4	12.816
	30%	0	0	249.8	10.982	0	0	504.7	21.802
	50%	0	0	470.1	23.927	0	0	1024.5	31.658
	70%	0	0	757.0	39.721	0	0	1700.8	39.131

Fonte – Elaborado pelo autor

Das tabelas acima podemos verificar que ambos os algoritmos retornaram solução ótima nos testes realizados com grafos aleatórios. Porém, o tempo médio para obtenção da solução no PAGMS$_d$ foi menor, assim como sua variância. Portanto nos nossos testes para grafos aleatórios o PAGMS$_d$ foi melhor do que o algoritmo para AGGMM aplicado ao PAGMS$_d$.

5.2.2 Grafos com estrela

A Tabela 5 e 6 são resultados de testes com grafos aleatórios, mas que possuem uma componente estrela conectada ao restante do grafo onde k representa o grau desta componente.

Tabela 5 – Resultados do Grupo G1 com estrela.

Grupo G_1-estrela			$PAGMS_d$				Furer			
			Soma dos excessos		Tempo		Soma dos excessos		Tempo	
n	Densidade	k	AM	STD	AM	STD	AM	STD	AM	STD
100	30%	3	0	0	2.6	0.516	0	0	4.0	0.666
		5	2	0	3.6	1.173	30.3	1.418	5.1	0.737
		7	4	0	3.8	0.918	54.9	0.316	4.6	0.516
		10	7	0	3.6	1.264	67.9	1.286	4.1	0.316
	50%	3	0	0	4.7	0.674	0	0	7.7	0.674
		5	2	0	6.8	2.043	31.0	0.471	9.5	1.080
		7	4	0	5.4	0.527	55.9	1.523	8.8	0.421
		10	7	0	6.6	1.897	70.3	1.251	7.9	0.567
	80%	3	0	0	8.8	0.421	0	0	15.7	0.823
		5	2	0	9.8	0.421	31.3	0.674	18.7	1.337
		7	4	0	9.4	0.516	55.9	1.197	17.2	1.475
		10	7	0	8.8	0.421	71.3	0.823	15.0	0.666

Fonte – Elaborado pelo autor.

Tabela 6 – Resultados do Grupo G2 com estrela.

Grupo G_2-estrela			$PAGMS_d$				Furer			
			Soma dos excessos		Tempo		Soma dos excessos		Tempo	
n	Densidade	k	AM	STD	AM	STD	AM	STD	AM	STD
300	30%	3	0	0	247.2	10.2	0	0	491.1	14.5
		5	2	0	310.5	78.5	94.8	1.0	569.0	11.9
		7	4	0	423.4	128.9	169.9	1.3	578.9	44.0
		10	7	0	391.0	116.4	212.6	1.4	514.7	15.7
	50%	3	0	0	471.0	19.6	0	0	1009.2	22.7
		5	2	0	615.5	180.7	94.6	0.8	1173.4	97.7
		7	4	0	518.4	25.4	171.1	1.7	1089.0	20.5
		10	7	0	507.9	23.0	215.7	1.5	1034.1	19.8
	80%	3	0	0	905.1	44.9	0	0	1999.5	39.5
		5	2	0	1026.6	33.0	93.8	1.0	2318.2	171.3
		7	4	0	1010.9	36.7	170.0	1.7	2201.4	41.6
		10	7	0	1054.9	208.5	215.0	1.9	2073.5	37.3

Fonte – Elaborado pelo autor.

Nos testes realizados com grafos com estrela o PAGMS$_d$ retornou melhores resultados quando comparamos ao algoritmo de AGGMM. Vale ressaltar que a diferença é muito grande de soma. Por exemplo, no grupo G_2 com estrela com densidade 80% e $k = 10$ o PAGMS$_d$ retorna soma 7, enquanto o algoritmo de AGGMM retorna soma 215. No PAGMS$_d$ todas as soluções retornadas foram as soluções ótimas ao contrário do algoritmo para AGGMM. Exceto para $k = 3$, onde nos dois grupos para toda densidade o algoritmo de AGGMM se iguala

ao PAGMS$_d$ em relação a soma dos excessos. Porém nosso algoritmo retorna a solução ótima em menor tempo.

6 CONCLUSÃO E TRABALHOS FUTUROS

Este trabalho apresentou um algoritmo para um novo problema de árvore geradora com restrição de grau e que apresentou melhores resultados em relação ao algoritmo descrito em (FURER; RAGHAVACHARI, 1994). No decorrer do trabalho, se conseguiu atingir os seguintes objetivos:

a) Desenvolver um algoritmo de geração de instâncias para o PAGMS$_d$.
b) Implementar o algoritmo descrito em (FURER; RAGHAVACHARI, 1994).
c) Propor um algoritmo aproximativo para o PAGMS$_d$.
d) Encontrar um fator de aproximação e demonstrá-lo.
e) Implementar o algoritmo proposto.
f) Comparar os tempos de respostas e a qualidade das soluções geradas por ambos os algoritmos.

Utilizando nossa abordagem e comparando com (FURER; RAGHAVACHARI, 1994) aplicado ao PAGMS$_d$, verificou-se que o algoritmo para o PAGMS$_d$ alcançou melhores resultados apresentando para todas as instâncias uma solução ótima. Uma pesquisa buscando encontrar limites inferiores melhores bem como encontrar grafos e propriedades de grafos onde o PAGMS$_d$ não encontra os melhores resultados poderiam ser trabalhos futuros. Por questões de tempo, estas pesquisas não puderam ser realizadas durante este trabalho.

REFERÊNCIAS

BONDY, J.; MURTY, U. **Graph theory (graduate texts in mathematics)**. [S.l.]: Springer New York, 2008.

CORMEN, T. H.; LEISERSON, C. E.; RIVEST, R. L.; STEIN, C. **Algoritmos**: teoria e prática. Editora Campus, v. 2, [S.I: S.n], 2002.

FEOFILOFF, P.; KOHAYAKAWA, Y.; WAKABAYASHI, Y. **Uma introdução sucinta à teoria dos grafos**. [S.l: S.n], 2011.

FURER, M.; RAGHAVACHARI, B. Approximating the minimum-degree steiner tree to within one of optimal. **Journal of Algorithms**, Elsevier, v. 17, n. 3, p. 409–423, 1994.

GOEMANS, M. X. Minimum bounded degree spanning trees. In: IEEE. **Foundations of Computer Science, 2006. FOCS'06. 47th Annual IEEE Symposium on**. [S.l.], 2006. p. 273–282.

HARJU, T. **Lecture notes on graph theory**. [S.l.]: Wiley, 2014.

KANN, V. **On the approximability of NP-complete optimization problems**. Tese (Doutorado) — Royal Institute of Technology Stockholm, 1992.

KÖNEMANN, J.; RAVI, R. A matter of degree: Improved approximation algorithms for degree-bounded minimum spanning trees. In: ACM. **Proceedings of the thirty-second annual ACM symposium on Theory of computing**. [S.l.], 2000. p. 537–546.

NetworkX. **NetworkX**. 2005. Disponível em: <http://networkx.readthedocs.io/en/stable/>. Acesso em: 28 julho 2017.

RAIDL, G. R. An efficient evolutionary algorithm for the degree-constrained minimum spanning tree problem. In: IEEE. **Evolutionary Computation, 2000. Proceedings of the 2000 Congress on**. [S.l.], 2000. v. 1, p. 104–111.

SIPSER, M. **Introdução à Teoria da Computação**, 2ª edição. Cengage Learning, p. 978–85. [S.l: S.n], 2007.

WILLIAMSON, D. P.; SHMOYS, D. B. **The design of approximation algorithms**. [S.l.]: Cambridge university press, 2011.

WU, B. Y.; CHAO, K.-M. **Spanning trees and optimization problems**. [S.l.]: CRC Press, 2004.