

UNIVERSIDADE FEDERAL FLUMINENSE

ANDERSON BELTRÃO DE OLIVEIRA

**Heurísticas Gulosas e Randomizadas para o
Problema da Árvore Geradora com Número
Mínimo de Ramificações**

NITERÓI

2023

ANDERSON BELTRÃO DE OLIVEIRA

Heurísticas Gulosas e Randomizadas para o Problema da Árvore Geradora com Número Mínimo de Ramificações

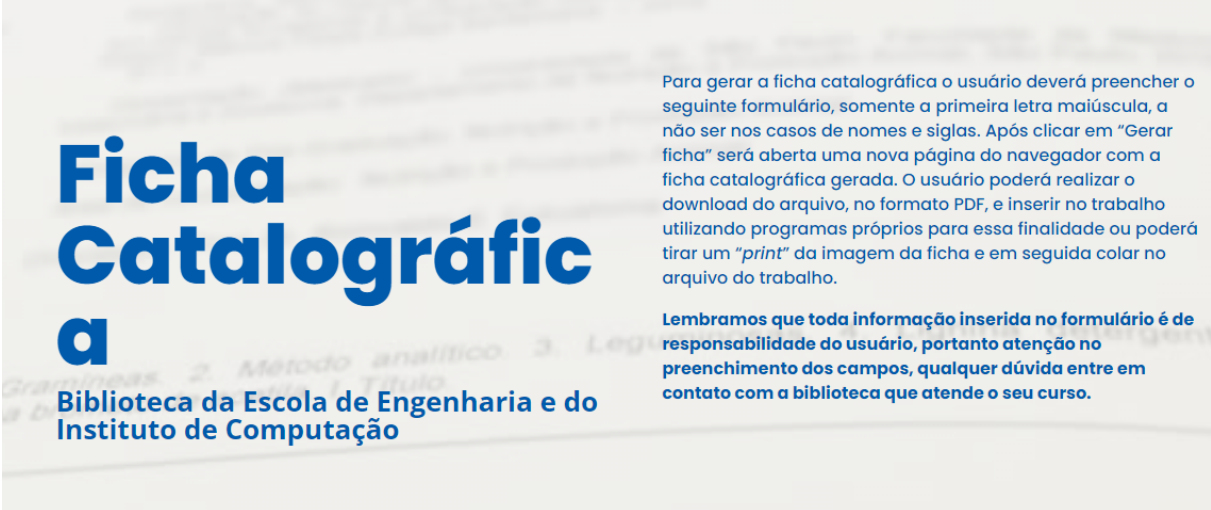
Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Computação da Universidade Federal Fluminense como requisito parcial para a obtenção do Grau de Mestre em Computação. Área de concentração: Algoritmos e Otimização

Orientador:
Fábio Protti

Coorientador:
Rodrigo Lamblet Mafort

NITERÓI

2023



Ficha Catalográfica

Biblioteca da Escola de Engenharia e do Instituto de Computação

Para gerar a ficha catalográfica o usuário deverá preencher o seguinte formulário, somente a primeira letra maiúscula, a não ser nos casos de nomes e siglas. Após clicar em “Gerar ficha” será aberta uma nova página do navegador com a ficha catalográfica gerada. O usuário poderá realizar o download do arquivo, no formato PDF, e inserir no trabalho utilizando programas próprios para essa finalidade ou poderá tirar um “*print*” da imagem da ficha e em seguida colar no arquivo do trabalho.

Lembramos que toda informação inserida no formulário é de responsabilidade do usuário, portanto atenção no preenchimento dos campos, qualquer dúvida entre em contato com a biblioteca que atende o seu curso.

Figura 1: Local da ficha catalográfica

ANDERSON BELTRÃO DE OLIVEIRA

Heurísticas Gulosas e Randomizadas para o Problema da Árvore Geradora com Número
Mínimo de Ramificações

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Computação da Universidade Federal Fluminense como requisito parcial para a obtenção do Grau de Mestre em Computação. Área de concentração: Algoritmos e Otimização

Aprovada em <MES> de 2023.

BANCA EXAMINADORA

Prof. Fábio Protti - Orientador, UFF

Prof. Rodrigo Lamblet Mafort - Coorientador, UERJ

Prof. Simone de Lima Martins, UFF

Prof. Maria Cláudia Silva Boeres, UFES

Niterói

2023

"Nós só podemos ver um pouco do futuro, mas o suficiente para perceber que há muito a fazer."

- Alan Turing

Agradecimentos

À minha família e amigos, pelo apoio e incentivo durante toda a minha formação.

Aos meus orientadores, Fábio Protti e Rodrigo Mafort, pela atenção e pelo esforço incansável. Seus ensinamentos e ponderações foram fundamentais ao longo dessa jornada.

Ao Instituto de Computação da Universidade Federal Fluminense, seus professores e funcionários pela dedicação e presteza.

À Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) pelo apoio financeiro que possibilitou este trabalho.

Finalmente, a Deus, por colocar todas essas pessoas em minha vida, pois nada disso seria possível sem elas.

Resumo

O problema conhecido na literatura como “*Minimum Branch Vertices Problem*” (MBV) consiste em encontrar uma árvore geradora T de um grafo não direcionado G que contenha o menor número de vértices com grau maior ou igual a três em T , chamados de ramificações ou “*branch vertices*”. O MBV é NP-difícil, e, por esta razão, muitas abordagens heurísticas têm sido propostas para contornar sua complexidade computacional. Neste trabalho propomos oito novas heurísticas construtivas para o problema, inspiradas no Algoritmo de Prim: quatro métodos gulosos e quatro variantes randomizadas. Os resultados obtidos para as instâncias de referência se mostraram superiores aos métodos construtivos anteriormente encontrados na literatura.

Palavras-chave: Grafos, Árvore Geradora, Ramificações, Heurísticas

Abstract

The Minimum Branch Vertices Problem (MBV) aims at finding a spanning tree T of an undirected graph G that minimizes the number of vertices with a degree greater than or equal to three in T , known as *branch vertices*. The MBV is NP-hard, and for this reason many heuristic approaches have been proposed to deal with its computational complexity. This study proposes eight novel constructive heuristics for the problem, inspired by Prim's Algorithm: four greedy methods and four randomized variations. The results obtained for the reference instances proved to be superior to the construction methods previously found in the literature.

Keywords: Graph, Spanning Tree, Branch Vertices, Heuristics

Lista de Figuras

1	Local da ficha catalográfica	
2	Exemplo de Lightpath	13
3	Exemplo de Light-tree	13
4	Grafo Ponderado P	18
5	Exemplo de Aplicação do Algoritmo de Kruskal no Grafo P (Fig. 4)	19
6	Exemplo de Aplicação do Algoritmo de Prim no Grafo P (Fig. 4)	20
7	Grafo G	21
8	Duas Possíveis Árvores Geradoras de G (Fig. 7)	22
9	Exemplo de Aplicação do Algoritmo BEP (7)	36
10	Exemplo de Aplicação do Algoritmo CEP (9)	41

Lista de Tabelas

1	Resultados Mínimos para <i>Medium Instances</i>	52
2	Gap dos Resultados Mínimos para <i>Medium Instances</i>	52
3	Resultados Mínimos para <i>Large Instances</i>	53
4	Gap dos Resultados Mínimos para <i>Large Instances</i>	54
5	Resultados Médios para <i>Medium Instances</i>	55
6	Resultados Médios para <i>Large Instances</i>	55

Lista de Algoritmos

1	Algoritmo de Kruskal (1956)	19
2	Algoritmo de Prim (1957)	20
3	Esboço da Heurística de Marín (2015)	24
4	Solução Inicial de Moreno, Frota e Martins (2018)	27
5	Pré - Processamento	31
6	Expansão de Ramificações	32
7	BEP - <i>Branch Expanding Prim</i>	35
8	EEP - <i>Edge Expanding Prim</i>	38
9	CEP - <i>Controlled Expanding Prim</i>	40
10	CEEP - <i>Controlled Edge Expanding Prim</i>	43
11	R-BEP - <i>Randomized Branch Expanding Prim</i>	45
12	R-CEP - <i>Randomized Controlled Expanding Prim</i>	46
13	R-EEP - <i>Randomized Edge Expanding Prim</i>	47
14	R-CEEP - <i>Randomized Controlled Edge Expanding Prim</i>	48

Sumário

1	Introdução	12
2	Preliminares	16
2.1	Conceitos Básicos da Teoria dos Grafos	16
2.2	Algoritmos Fundamentais	18
3	O Problema MBV	21
3.1	Revisão da Literatura	21
3.2	O Trabalho de Marín (2015)	22
3.2.1	Pré-processamento	23
3.2.2	Solução Heurística	23
3.3	O Trabalho de Moreno, Frota e Martins (2018)	25
3.3.1	Decomposição Baseada em Pontes	25
3.3.2	Construindo uma Solução Inicial	26
4	Novas Heurísticas Construtivas	29
4.1	Pré-processamento	29
4.2	Processo de Expansão de Ramificações	31
4.3	Algoritmos Gulosos	32
4.3.1	<i>Branch Expanding Prim</i>	33
4.3.2	<i>Edge Expanding Prim</i>	37
4.3.3	<i>Controlled Expanding Prim</i>	39
4.3.4	<i>Controlled Edge Expanding Prim</i>	42

4.4	Algoritmos Randomizados	42
4.4.1	Randomização dos algoritmos BEP e CEP	44
4.4.2	Randomização dos algoritmos EEP e CEEP	44
5	Experimentos Computacionais	50
5.1	Resultados para as Instâncias de Carrabs et al. (2013)	51
5.1.1	Estudo dos Resultados Mínimos	51
5.1.2	Estudo dos Resultados Médios	54
6	Conclusão	57
	REFERÊNCIAS	59

1 Introdução

Encontrar uma árvore geradora com número mínimo de ramificações (vértices com três ou mais vizinhos na árvore) é um problema complexo, conhecido na literatura como “*Minimum Branch Vertices Problem*” (MBV). Este problema foi apresentado inicialmente em (GARGANO et al., 2002), associado a uma aplicação onde ocorre a necessidade de alocação de um número mínimo de *switches* WDM (*Wavelength Division Multiplexing*) em redes óticas *multicast*. Como tais dispositivos são sofisticados e caros, minimizar sua quantidade em uma rede é um objetivo desejável. O trabalho de (GARGANO et al., 2002) abordou a complexidade computacional do problema e demonstrou que o MBV é um problema NP-difícil.

Nas redes óticas, WDM é a técnica utilizada para realizar a multiplexação por divisão de comprimento de onda, permitindo a transmissão de vários feixes de luz através de uma única fibra óptica, desde que cada feixe utilize comprimentos de onda distintos (STERNE; BALA, 1999). Uma sequência de enlaces de fibra que conecta dois nós de uma rede utilizando o mesmo comprimento de onda são denominadas de “*lightpaths*”. Contudo, quando dois *lightpaths* compartilham o mesmo enlace óptico, é necessário que eles utilizem comprimentos de onda diferentes (MUKHERJEE, 1997).

Switches são equipamentos utilizados em uma rede para interconectar equipamentos que tenham uma interface *Ethernet*. A tecnologia de *switches* totalmente ópticos tem revolucionado as transmissões em fibras ópticas, ao replicarem o sinal óptico por meio da divisão dos feixes de luz. Um avanço que encontra sua compreensão na evolução do conceito de *lightpaths* para o mais complexo “*light-tree*” (SAHASRABUDDHE; MUKHERJEE, 1999). Enquanto *lightpaths* são rotas diretas entre pontos, o conceito de “*light-tree*” permite conexões mais flexíveis, ramificadas como uma árvore. Ao adotar a arquitetura de *light-tree*, torna-se possível realizar a comunicação *multicast* de maneira eficiente em redes ópticas, possibilitando que um nó de origem se comunique com múltiplos nós de destino. Isso viabiliza que *switches* WDM copiem informações diretamente no domínio óptico ao fragmentar um feixe de luz. Esse processo oferece uma notória vantagem em

relação ao *multicast* eletrônico, que, em contraste com a separação do feixe luminoso, replica pacotes de dados de um lado para outro, recorrendo a *buffers* de armazenamento temporário (SAHASRABUDDHE; SINGHAL; MUKHERJEE, 2000), o que resulta na diminuição do desempenho da transmissão. As Figuras 2 e 3 ilustram os conceitos de *lightpath* e *light-tree*, respectivamente, em uma rede óptica.



Figura 2: Exemplo de Lightpath

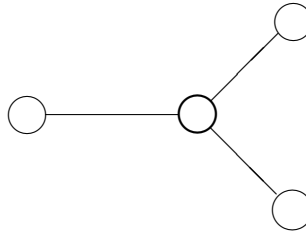


Figura 3: Exemplo de Light-tree

No contexto das redes ópticas, uma ramificação na árvore representa um ponto de divisão do sinal óptico na rede, onde se torna necessário alocar um *switch* WDM para efetuar a divisão. Sendo assim, evitar situações como a exemplificada na Figura 3 é crucial em redes ópticas, dado que adquirir e posicionar esse *switch* de forma a garantir operações *multicast* ágeis e eficientes é dispendioso. Portanto, diminuir a quantidade de *switches* na rede é fundamental para reduzir os custos (ALMEIDA, 2015). Tal problema foi reformulado como uma questão de otimização combinatória, visando reduzir os *switches* WDM em redes ópticas, ou seja, minimizar a quantidade de *switches* na rede com mais de duas conexões. Em termos do MBV, isso envolve encontrar uma árvore geradora T em um grafo G com número mínimo de ramificações ou “*branch vertices*”.

O MBV amplia o clássico problema de encontrar um caminho hamiltoniano em um grafo. Esse problema pode ser definido da seguinte forma: dado um grafo G conexo com n vértices, deseja-se obter um caminho que percorra todos os n vértices do grafo G passando apenas uma vez por cada vértice. Por certo, uma vez que um caminho hamiltoniano é uma árvore geradora sem qualquer ramificação, o número de ramificações na solução ótima do MBV para G será zero se, e somente se, G possui caminho hamiltoniano (ALMEIDA, 2015). Já que a determinação da propriedade de um grafo conter um caminho hamiltoniano é um problema classificado como NP-completo (GAREY; JOHNSON, 1979), a complexidade do MBV como problema NP-difícil é estabelecida, como demonstrado em (GARGANO et al., 2002).

Assim como o MBV, existem diversas variantes naturais de otimização associadas ao problema do caminho hamiltoniano. Por exemplo, pode-se querer minimizar o número de folhas, ou minimizar o grau máximo, em uma árvore geradora de G . Em ambos os casos, o número é igual a dois se e somente se G tiver um caminho Hamiltoniano (GARGANO et al., 2002). O desafio de otimização mais famoso desse gênero é o problema do caminho mais longo, onde G tem um caminho Hamiltoniano se e somente se o caminho mais longo tem n vértices.

Algumas heurísticas construtivas já foram propostas para gerar soluções para o MBV; por exemplo, os trabalhos de (CERULLI; GENTILI; IOSSA, 2009), (CARRABS et al., 2013), (ALMEIDA; NOGUEIRA; SÁ, 2014), (MARÍN, 2015), (MELO; SAMER; URRUTIA, 2016), (MORENO; PLASTINO; MARTINS, 2016) e (MORENO; FROTA; MARTINS, 2018), aliando boa qualidade das soluções com tempos computacionais extremamente baixos. Alguns desses trabalhos consideram as soluções encontradas pelas heurísticas construtivas como ponto de partida para métodos ainda mais elaborados, como, por exemplo, o método ILS proposto por (MORENO; PLASTINO; MARTINS, 2016; MORENO; FROTA; MARTINS, 2018). Entretanto, as heurísticas construtivas geralmente têm algumas características que as tornam interessantes como ferramentas para gerar boas soluções: simplicidade, facilidade de implementação e rapidez de processamento. O objetivo deste trabalho é explorar as possibilidades que tais heurísticas oferecem, propondo novas estratégias para a geração de soluções para o MBV, mantendo ainda a possibilidade de aproveitar os métodos propostos em acoplamento com métodos mais sofisticados, como, por exemplo, os baseados em meta-heurísticas.

Neste trabalho, foram propostas oito novas heurísticas construtivas para o MBV. As quatro primeiras, BEP, EEP, CEP e CEEP, são algoritmos baseados em estratégias gulosas. As outras quatro, R-BEP, R-EEP, R-CEP e R-CEEP, são variantes das primeiras e incorporam critérios randomizados. As heurísticas exploram as pontes e articulações existentes em um grafo para a construção de uma solução. Em seguida, foram comparados os resultados destes algoritmos com a melhor heurística construtiva gulosa da literatura (de acordo com o conhecimento do autor desta dissertação), pertencente ao trabalho de (MORENO; FROTA; MARTINS, 2018).

O restante deste trabalho se encontra estruturado da seguinte maneira: o Capítulo 2 apresentará alguns conceitos básicos da Teoria dos Grafos e de Algoritmos. No Capítulo 3 é definido o problema MBV. Nele também serão vistos os trabalhos mais relevantes da literatura em relação ao MBV. O Capítulo 4 descreve as heurísticas propostas. O

Capítulo 5 apresenta os resultados obtidos por tais heurísticas. Por fim, as conclusões são discutidas no Capítulo 6.

2 Preliminares

Este capítulo apresentará alguns conceitos básicos da Teoria dos Grafos e de Algoritmos, necessários para o estudo do problema MBV.

2.1 Conceitos Básicos da Teoria dos Grafos

Um grafo $G = (V_G, E_G)$ é definido por um conjunto de vértices ou nós (V_G) e outro de arestas (E_G). Cada aresta $e \in E_G$ é representada por um par não ordenado de vértices (v, u) , onde $v, u \in V_G$. Os vértices v e u são os extremos da aresta e , sendo chamados de vértices adjacentes ou vizinhos, uma vez que estes são extremos de uma mesma aresta que os liga. A aresta e é dita incidente ao vértices v e u quando estes são seus extremos. Existe duas medidas em relação ao tamanho de um grafo G , o seu número de vértices $n = |V_G|$ e o seu número de arestas $m = |E_G|$.

Este trabalho abordará grafos conexos, simples, não direcionados e não ponderados. Um grafo é dito conexo se existir pelo menos um caminho entre cada par de vértices do grafo. Um caminho em um grafo é uma sequência de vértices distintos conectados por uma série de arestas distintas. Um grafo é chamado de simples se ele não contém multiarestas ou laços. Uma multiaresta corresponde a duas ou mais arestas com os mesmos extremos. Já um laço representa uma aresta cujos extremos são iguais, ou seja, é uma aresta que conecta um vértice a ele mesmo. Um grafo é denominado não direcionado quando a existência de uma aresta entre dois vértices v e u do grafo implica uma relação bilateral entre eles, ou seja, v é vizinho de u , assim como u é vizinho de v . Em um grafo direcionado, cada uma de suas arestas é associada a um par ordenado de vértices. No grafo não direcionado, os pares de vértices não são ordenados. Um grafo é dito não ponderado quando não há existência de pesos em suas arestas.

Uma árvore é um grafo conexo que não possui ciclos. Um ciclo é formado por um caminho em que o primeiro e o último vértices são adjacentes. Uma floresta é um grafo desconexo composto pela união disjunta de árvores. Uma árvore geradora T de G é uma

árvore contendo todos os n vértices de G e um subconjunto mínimo de arestas de G com $n - 1$ arestas que conectam todos os n vértices sem formar ciclo em T . Uma arborescência é uma árvore direcionada com uma única fonte (vértice raiz) em que todos os demais vértices tem grau de entrada igual a 1.

Um subgrafo de um grafo G é um grafo cujos conjuntos de vértices e arestas são subconjuntos dos conjuntos de vértices e arestas de G . Intitula-se subárvore quando o subgrafo é uma árvore. Uma componente conexa é um subgrafo conexo que não faz parte de nenhum subgrafo conexo maior. As componentes conexas de qualquer grafo dividem seus vértices em conjuntos disjuntos e são os subgrafos induzidos desses conjuntos. Um subgrafo de G é dito induzido se ele possui todas as arestas que aparecem em G sobre o mesmo conjunto de vértices.

Uma folha é um vértice que possui grau igual a 1. O grau de um vértice v é o número de arestas incidentes a ele, ou seja, é o número de vértices adjacentes a v . Uma ponte ou aresta de corte é uma aresta cuja remoção aumenta o número de componentes conexas do grafo. Uma articulação ou vértice de corte é um vértice cuja remoção em um grafo aumenta o seu número de componentes conexas. Uma ramificação ou “*branch vertex*” é um vértice com grau maior ou igual a 3. Um arco é uma aresta direcionada, isto é, o seu par de vértices é ordenado.

A notação $d_G(v)$ é usada para representar o grau do vértice v no grafo G , enquanto a notação $d_T(v)$ representa o grau do vértice v em uma árvore T . As notações $adj_G(v)$ e $w(G)$ retratam respectivamente a lista de vértices adjacentes ao vértice v no grafo G e o número de componentes conexas do grafo G . A densidade de um grafo é a razão entre a quantidade de arestas do grafo e a quantidade de arestas do grafo completo com o mesma quantidade de vértices. Um grafo completo é um grafo simples em que todo vértice é adjacente a todos os outros vértices.

Existem diversas formas de representar um grafo. Neste trabalho é utilizada a representação geométrica do grafo, onde os vértices representam pontos distintos no plano, enquanto as arestas correspondem a linhas interligando os vértices adjacentes.

Como referências na literatura em português de Teoria dos Grafos, destacam-se: (SZWARCFITER, 2018) e (BOAVENTURA NETTO, 2012). Em inglês, destacam-se: (BONDY; MURTY, 2008) e (DIESTEL, 2017).

2.2 Algoritmos Fundamentais

Um algoritmo é uma sequência finita de instruções ou comandos realizados de maneira sistemática com o propósito de resolver um problema ou executar uma tarefa.

Uma heurística em Ciência da Computação é uma estratégia elaborada para agilizar a resolução de um problema diante da lentidão dos métodos convencionais, ou para descobrir uma solução aproximada nos casos em que os métodos exatos possuem dificuldade em alcançar uma solução ótima. Isso é alcançado mediante a priorização da velocidade em vez da exatidão ou precisão. De certa forma, pode ser vista como um atalho.

Um algoritmo guloso ou míope obtém uma solução comum e de fácil implementação para problemas de otimização. Ele segue uma estratégia de escolher sempre a opção mais apetitosa (um ótimo local) em cada etapa do problema, na esperança de atingir um ótimo global. O algoritmo guloso é dito míope, pois ele toma decisões com base nas informações disponíveis na iteração corrente, sem olhar as consequências que essas decisões terão no futuro. As escolhas que o algoritmo faz em cada iteração são definitivas.

Algoritmos randomizados ou aleatorizados são aqueles que utilizam um grau de aleatoriedade como parte de sua lógica ou procedimento, para decidir, em um ou mais momentos durante sua execução, o que fazer ou para onde ir. Portanto, seu comportamento é determinado não apenas por sua entrada, mas também por valores produzidos por um gerador de números pseudo-aleatórios. Diferentemente dos algoritmos determinísticos, um algoritmo randomizado, dada uma mesma entrada, não necessariamente leva a uma mesma saída.

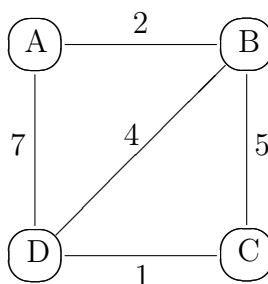


Figura 4: Grafo Ponderado P

O Algoritmo de Kruskal ([KRUSKAL, 1956](#)) é um algoritmo que busca uma árvore geradora mínima para um grafo conexo, não direcionado e ponderado. Uma árvore geradora é dita mínima se, dentre todas as árvores geradoras que existem no grafo, a soma dos pesos nas arestas dela é o menor possível. A Figura 5 exemplifica o processo desse algoritmo. Dado um grafo P (Figura 4), o algoritmo começa criando uma floresta F

contendo todos os n vértices de P (Figura 5(a)). Em seguida, o algoritmo vai buscar uma aresta de menor peso de P (marcada em azul), que ainda não esteja presente em F , para inserir na floresta, conectando duas componentes e sem gerar ciclo (Figura 5(b) - (d)). Esse processo será repetido até que a floresta F se torne uma árvore geradora mínima de P (Figura 5(d)). O Algoritmo de Kruskal é um exemplo de algoritmo guloso, pois a cada iteração ele busca pela aresta mais apetitosa. Sua complexidade é $O(m \log m)$. O Algoritmo de Kruskal está detalhado como Algoritmo 1.

Algoritmo 1: Algoritmo de [Kruskal \(1956\)](#)

Entrada: Grafo ponderado $P = (V_P, E_P)$

Saída: Uma árvore geradora $T = (V_T, E_T)$ de P com custo mínimo

```

1  $V_T \leftarrow V_P$ ;  $E_T \leftarrow \emptyset$  // Inicialização de  $T$ 
2  $S \leftarrow E_P$  // Vetor auxiliar de arestas
3 Ordene as arestas de  $S$  em ordem crescente baseado nos seus respectivos pesos
4  $i \leftarrow 0$  // Considere 0 a posição inicial de um vetor
5 enquanto  $|E_T| < |V_P| - 1$  faça
6    $e \leftarrow S[i]$  //  $e$  recebe uma aresta de  $S$  na posição  $i$ 
7   se  $e$  não forma ciclo com as arestas em  $E_T$  então
8      $E_T \leftarrow E_T \cup \{e\}$ 
9    $i \leftarrow i + 1$ 
10 retorne ( $T$ )
```

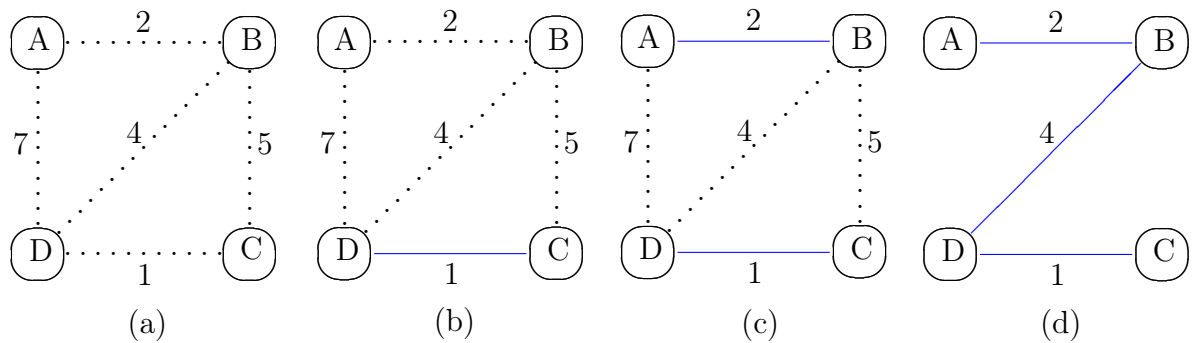


Figura 5: Exemplo de Aplicação do Algoritmo de Kruskal no Grafo P (Fig. 4)

O Algoritmo de Prim ([PRIM, 1957](#)), assim como o de Kruskal ([KRUSKAL, 1956](#)), é um algoritmo guloso que busca uma árvore geradora mínima para um grafo conexo, não direcionado e ponderado. A Figura 6 ilustra o processo desse algoritmo. Dado um grafo P (Figura 4), o algoritmo escolhe um vértice (uma subárvore) aleatório do grafo como ponto de partida para cultivar uma árvore geradora mínima de P (Figura 6(a)). O crescimento ocorre pela adição de arestas de menor peso de P (marcadas em azul), que conectam os vértices da subárvore cultivada aos seus vizinhos em P que ainda não fazem parte dela (Figura 6(b) - (d)). Esses vizinhos serão incluídos na subárvore com a adição

dessas arestas. A subárvore será cultivada até possuir todos os n vértices do grafo P , se tornando uma árvore geradora mínima de P (Figura 6(d)). A complexidade do Algoritmo de Prim é relativa a estrutura de dados empregada, sendo $O(m \log n)$ utilizando lista de adjacência e $O(n^2)$ usando matriz de adjacência. O Algoritmo de Prim está detalhado como Algoritmo 2.

Como referência na literatura em Algoritmos, destaca-se: (CORMEN et al., 2009).

Algoritmo 2: Algoritmo de Prim (1957)

Entrada: Grafo ponderado $P = (V_P, E_P)$

Saída: Uma árvore geradora $T = (V_T, E_T)$ de P com custo mínimo

```

1 Escolher um vértice qualquer  $v$  em  $V_P$ 
2  $V_T \leftarrow v$ ;  $E_T \leftarrow \emptyset$  // Inicialização de  $T$ 
3 enquanto  $|E_T| < |V_P| - 1$  faça
4   Seja  $e = (x, y)$  a aresta de peso mínimo com um extremo  $x$  em  $V_T$  e outro  $y$ 
   em  $V_P \setminus V_T$ 
5    $E_T \leftarrow E_T \cup \{e\}$ 
6    $V_T \leftarrow V_T \cup \{y\}$ 
7 retorne  $(T)$ 

```

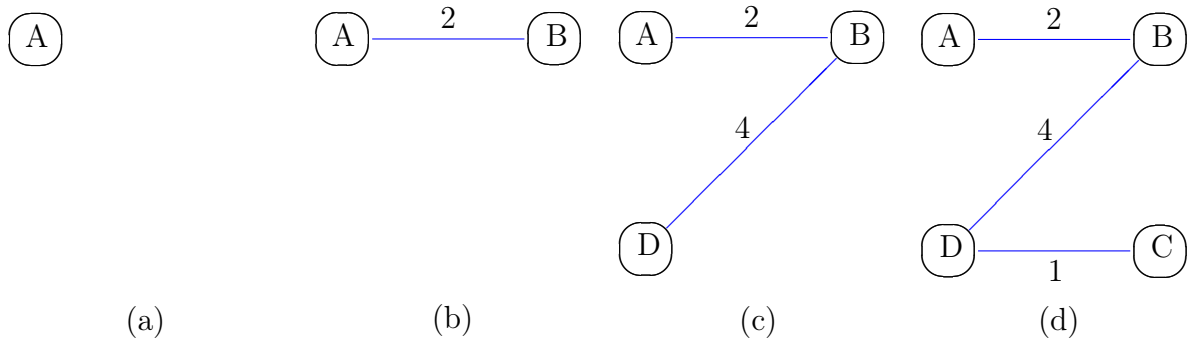


Figura 6: Exemplo de Aplicação do Algoritmo de Prim no Grafo P (Fig. 4)

3 O Problema MBV

Dado um grafo conexo, simples, não direcionado e não ponderado G , composto por um conjunto de vértices (V_G) e outro de arestas (E_G), o problema MBV pede uma árvore geradora $T = (V_T, E_T)$ de G , sendo $V_T = V_G, E_T \subseteq E_G$, que possua a menor quantidade possível de vértices v com grau maior ou igual a 3 ($d_T(v) \geq 3$), chamados de ramificações ou “*branch vertices*”. Formalmente, o MBV pode ser definido como se segue:

Entrada: Grafo $G = (V_G, E_G)$ não direcionado.

Objetivo: Encontrar uma árvore geradora $T = (V_T, E_T)$ de G com número mínimo de ramificações.

Como exemplo, temos as Figuras 7 e 8 que retratam, respectivamente, um grafo conexo e não direcionado G e duas possíveis árvores geradoras de G . Enquanto a árvore da Figura 8(a) possui uma ramificação, a árvore da Figura 8(b) não possui nenhuma. Logo, das árvores geradoras de G ilustradas na Figura 8, a árvore da Figura 8(b) apresenta uma melhor solução para o problema MBV em relação à da Figura 8(a), já que a árvore da Figura 8(b) contém um menor número de “*branch vertices*”.

3.1 Revisão da Literatura

Na literatura, várias abordagens foram propostas para resolver o MBV. Em (CERULLI; GENTILI; IOSSA, 2009), os autores propuseram três estratégias heurísticas para o problema: a de ponderação de arestas, a de coloração de nós e uma combinação de ambas.

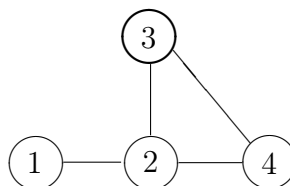


Figura 7: Grafo G

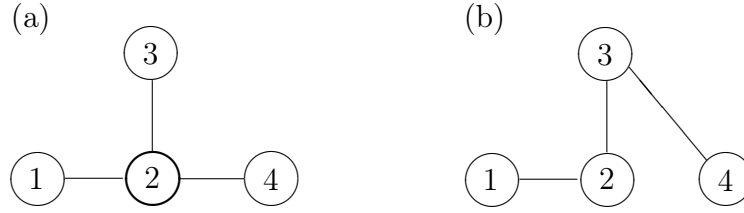


Figura 8: Duas Possíveis Árvore Geradoras de G (Fig. 7)

Em ([SILVA et al., 2014](#)) foi apresentada uma heurística de troca de arestas, que reduzia iterativamente o número de ramificações de uma árvore geradora aleatória. Em ([MARÍN, 2015](#)) foram apresentados um método heurístico de dois estágios que fornece soluções iniciais muito boas e um algoritmo *branch-and-cut* baseado em uma formulação de Programação Inteira. Uma abordagem de decomposição baseada em pontes e vértices de corte do grafo para dividir o problema em vários subproblemas menores foi proposta por ([MELO; SAMER; URRUTIA, 2016](#)). Esse mesmo trabalho apresentou heurísticas construtivas eficazes para o MBV que levam em consideração a estrutura do problema para obter boas soluções viáveis. Em ([SILVESTRI; LAPORTE; CERULLI, 2017](#)), os autores desenvolveram uma formulação híbrida contendo variáveis direcionadas e não direcionadas, que foi resolvida por um algoritmo *branch-and-cut*. Por fim, os trabalhos de ([MORENO; PLASTINO; MARTINS, 2016](#); [MORENO; FROTA; MARTINS, 2018](#)) apresentam uma meta-heurística ILS (*Iterated Local Search*), que recebe como solução inicial uma árvore geradora concebida por uma heurística construtiva.

Uma generalização do problema MBV foi proposta por ([MERABET; MOLNAR, 2016](#)) e visa minimizar o número de vértices com grau estritamente maior do que um parâmetro d (as ramificações). A generalização do problema é conhecida na literatura como “*Minimum d -Branch Vertices Problem*” (d -MBV), e consiste em encontrar uma árvore geradora de um grafo G com o número mínimo de vértices com grau maior que um valor inteiro fixo d . O d -MBV também foi abordado em ([MORENO; FROTA; MARTINS, 2018](#)), onde foram apresentados métodos exatos e heurísticos que refinam as heurísticas inicialmente apresentadas em ([MORENO; PLASTINO; MARTINS, 2016](#)).

3.2 O Trabalho de [Marín \(2015\)](#)

O trabalho de ([MARÍN, 2015](#)), propõe um método heurístico de dois estágios que constrói soluções iniciais muito boas, e que fornece limites superiores utilizados em um algoritmo *branch-and-cut* baseado em uma formulação de Programação Inteira. Nesta seção, focaremos em sua heurística construtiva de dois estágios.

3.2.1 Pré-processamento

No trabalho de ([MARÍN, 2015](#)), uma fase de pré-processamento é realizada a fim de encontrar as pontes de um grafo G . A identificação das pontes é feita através do algoritmo proposto por ([SCHMIDT, 2013](#)), onde uma arborescência geradora (árvore geradora orientada) é construída com todos os arcos direcionados para o vértice raiz por meio de uma busca em profundidade.

Tendo identificado as pontes, estas são removidas do grafo. Ao remover todas as pontes, alguns vértices podem ficar isolados. Após identificar o conjunto de vértices isolados (V_I), estes serão removidos juntamente com as arestas correspondentes de G , produzindo o subgrafo induzido por $V_H = V \setminus V_I$, denominado $H = (V_H, E_H)$. O número de vértices excluídos na fase de pré-processamento entre os vizinhos de j é denotado $\delta_j^p = |N_G(j) \cap V_I|$.

3.2.2 Solução Heurística

O método heurístico de ([MARÍN, 2015](#)) começa com uma árvore geradora construída com uma versão modificada dinamicamente do Algoritmo de Prim ([PRIM, 1957](#)). Esta árvore é então melhorada com diversas estratégias de troca. Algumas das estratégias reduzem o número de ramificações da árvore atual, mas outras apenas a perturbam, sem aumentar o número de ramificações. Como a primeira parte da heurística utiliza pesos aleatórios, e dado que o processo geral é muito rápido, é utilizado um método *multi-start*.

A estrutura básica da heurística é dada no Algoritmo 3. Na primeira etapa da heurística, os pesos do passo 1a só são relevantes quando um dos extremos da aresta (i, j) , digamos i , está marcado (dentro da árvore) e o outro extremo não está marcado (fora da árvore). Seja τ a árvore atual, M um valor suficientemente grande e α uma variável aleatória uniforme entre 0 e 1. Então o peso atribuído a (i, j) será:

$$\begin{aligned} \alpha + d_G(j) + d_\tau(i) & \quad \text{se } d_\tau(i) \geq 3 \text{ ou } \delta_j^p \geq 2, \\ \alpha + M + d_G(j) & \quad \text{se } d_\tau(i) = 1, \\ \alpha + M^2 & \quad \text{caso contrário,} \end{aligned}$$

onde d_G é o grau em G , e d_τ é o grau em τ . O efeito desses pesos é que as arestas com extremo marcado, que se tornarão ramificações, são priorizadas, e preferidas se seu grau original (em G) for baixo, pois suas arestas incidentes serão mais prováveis em qualquer árvore geradora. As arestas tais que um dos extremos é uma folha na árvore atual são

então priorizadas e ordenadas aleatoriamente. Os demais vértices seguem uma ordem aleatória.

Na segunda etapa da heurística, existem dois movimentos diferentes onde as soluções são aprimoradas: IMP1 e IMP2. IMP1 é repetido até que não mais seja possível aplicá-lo. As arestas e de fora da árvore são consideradas iterativamente e selecionadas se nenhum dos extremos tiver grau dois na árvore atual. As arestas da árvore que fecham um ciclo com e são identificadas. Se a substituição de uma das arestas do ciclo por e reduz o grau de qualquer vértice da árvore de 3 para 2, a substituição é feita e o número de “*branch vertices*” é reduzido em 1 ou 2. Aplica-se então o IMP2, onde, ao conseguir aprimorar a solução uma vez, retorna o processo ao movimento IMP1. No IMP2, as arestas e de fora da árvore são consideradas iterativamente. Cada uma dessas arestas fecha um ciclo quando adicionada à árvore. A aresta e é selecionada se (i) um de seus extremos tem grau diferente de dois na árvore atual e (ii) a aresta que é incidente ao outro extremo de e que pertence ao ciclo também é incidente a um segundo vértice j_i de grau 3.

Algoritmo 3: Esboço da Heurística de [Marín \(2015\)](#)

- 1 Repita várias vezes:
 - 2 **Etapa 1.** Algoritmo de Prim modificado dinamicamente.
Comece com uma aresta aleatória, adicionando-a à árvore. Marque os dois extremos.
Repita $|V_H| - 2$ vezes os passos 1a e 1b.
Passo 1a. Dê novos pesos às arestas de E_H .
Passo 1b. Escolha a borda com peso mínimo com exatamente um extremo marcado. Adicione-o à árvore e marque seu extremo não marcado.
 - 3 **Etapa 2.** Repita as etapas 2a e 2b até que não seja possível:
Passo 2a. Fase de melhoria. Repita IMP1 e IMP2 até que nenhuma melhoria seja possível.
Passo 2b. Fase de perturbação. Aplicar PER2. Se a árvore permanecer inalterada, aplique PER1. Se a árvore permanecer inalterada, aplique PER3.
-

Quando nenhuma melhoria é possível aplicando IMP1 e IMP2, a fase de perturbação modifica a árvore sem minimizar o número de “*branch vertices*”, mas reduzindo o grau de alguma ramificação da árvore. PER1 é semelhante ao IMP2, mas será aplicada quando o grau de j_i for maior que 3. PER2 é semelhante ao IMP1 e será aplicada sempre que o grau de alguma ramificação do ciclo fechado pela aresta candidata for reduzido em caso de substituição. Por fim, PER3 é semelhante a PER2, mas a troca é feita quando o grau na árvore de uma ramificação é reduzido de 3 para 2 e outro vértice de grau 2 é convertido em uma ramificação. Para evitar ciclagem, cada aresta candidata é considerada apenas uma vez caso nenhuma nova melhoria seja obtida na execução dos movimentos IMP1 e

IMP2.

3.3 O Trabalho de [Moreno, Frota e Martins \(2018\)](#)

No trabalho de ([MORENO; FROTA; MARTINS, 2018](#)) foi desenvolvido uma meta-heurística ILS para o problema d -MBV, além de uma abordagem exata. Sua meta-heurística ILS recebe como entrada uma solução inicial gerada por uma heurística construtiva gulosa muito eficaz. Neste trabalho, focaremos em sua heurística construtiva, descrita na Subseção 3.3.2, (para $d = 2$), para fins de comparação.

3.3.1 Decomposição Baseada em Pontes

Uma abordagem de decomposição de grafos para o problema MBV, desenvolvida por ([MELO; SAMER; URRUTIA, 2016](#)) e também por ([LANDETE; MARÍN; SAINZ-PARDO, 2017](#)), é utilizada no trabalho de ([MORENO; FROTA; MARTINS, 2018](#)) em seu pré-processamento. Essa abordagem consiste em examinar o problema por meio da decomposição do grafo em subgrafos mais simples de resolver e, posteriormente, combinar as soluções desses subgrafos para criar uma solução para o problema. As pontes de um grafo G são identificadas, permitindo a caracterização de alguns de seus vértices como d -ramificações obrigatórias e possibilitando a criação de importantes estruturas de dados, como descrito a seguir.

Sendo $\beta(v)$ o número o número de pontes incidentes no vértice v , foi definido o seguinte conjunto: $O_D = \{u \in V_G \mid d_G(u) > d \wedge \beta(u) \geq d\}$. O conjunto O_D será composto apenas por vértices que serão d -ramificações (*d -branch vertices*) na solução ótima. Se o número de pontes adjacentes for maior que d , então o vértice deve ser d -ramificação na solução ótima. Considerando que o vértice v tenha exatamente d pontes adjacentes, se essas pontes fossem removidas, o vértice v pertenceria a uma componente composta por mais de um vértice ($d_G(v) > d$) e, portanto, para qualquer árvore geradora desta componente, seria necessário pelo menos uma aresta incidente ao vértice v . Como resultado, v teria pelo menos $d + 1$ arestas incidentes e, portanto, tem que ser necessariamente uma d -ramificação.

As pontes identificadas do grafo são eliminadas para realizar a sua decomposição, incorporado o parâmetro $l(v)$ em seu lugar. O parâmetro $l(v)$ indica o número de pontes incidentes ao vértice v .

3.3.2 Construindo uma Solução Inicial

A heurística construtiva de ([MORENO; FROTA; MARTINS, 2018](#)), assim como o Algoritmo de ([KRUSKAL, 1956](#)), é baseada na seleção de arestas de um grafo não direcionado G que ainda não estão na solução parcial T .

Os pesos w_{sOD} e w_{aOD} de uma aresta (u, v) são definidos como:

$$\begin{aligned} w_{sOD} &= d_G(u) + l(u) + d_G(v) + l(v) - f_D(u) \times n - f_D(v) \times n \\ w_{aOD} &= d_G(u) + l(u) + d_G(v) + l(v) + f_D(u) \times n + f_D(v) \times n \end{aligned}$$

A função $f_D(v)$ define se um vértice v pertence ao conjunto O_D , da seguinte forma:

$$f_D(v) = \begin{cases} 1, & \text{se } v \in O_D \\ 0, & \text{se } v \notin O_D \end{cases}$$

O Algoritmo 4 contém o pseudocódigo da estratégia utilizada para a construção da solução inicial. Uma floresta T é inicializada com todos os vértices de G , sem arestas. Neste algoritmo, um conjunto de arcos A é definido, tal que para todo $(i, j) \in E_G$, temos $(i, j), (j, i) \in A$. O conjunto A é explorado para selecionar as arestas que estarão em T . Entre as linhas 3 e 6, o arco (u, v) com aresta associada (u, v) de valor w_{sOD} mínimo é selecionado se atender às seguintes condições: (i) o vértice u não tem arestas incidentes em T ; (ii) o vértice v é incidente a T ; (iii) a soma do grau de v em T com $l(v)$ é diferente de d . Para este arco selecionado, a aresta associada será adicionada em T . Essa aresta adicionada não criará ciclos nem novas d -ramificações em T . Além disso, selecionar arcos com valores w_{sOD} mínimos para as arestas associadas prioriza aqueles arcos cujos vértices são d -ramificações e possuem um grau pequeno no grafo G . Desta forma, os vértices obrigatórios terão mais arestas adicionadas.

Entre as linhas 7 e 9, os arcos são selecionados de acordo com os seguintes critérios ordenados. Primeiro, no critério (a), um arco (u, v) é selecionado se o vértice u tiver um grau menor que d e se o vértice v for uma d -ramificação; nessa situação, inserir a aresta associada não irá gerar novas d -ramificações em T . Esse critério prioriza arcos cujos vértices devem ser d -ramificações na solução final e têm grau alto em G . Se nenhum vértice for encontrado no critério (a), então procura-se um arco com valor máximo w_{aOD} em T para a aresta associada e cujos extremos já são d -ramificações selecionadas (critério (b)), de modo que o número de d -ramificações não seja incrementado. Esse critério também prioriza arcos cujos vértices devem ser d -ramificações e possuir grau alto. Novamente, se

Algoritmo 4: Solução Inicial de [Moreno, Frota e Martins \(2018\)](#)

Entrada: Um Grafo $G = (V_G, E_G)$ sem pontes, o valor $l(v)$ associado a cada vértice, o conjunto O_D e o conjunto de arcos A tal que para todo $(i, j) \in E_G$, $(i, j), (j, i) \in A$.

Saída: Uma árvore geradora T de G

```

1  $T \leftarrow (V_G, \emptyset)$ 
2  $m \leftarrow 0$ 
3 repita
4   Encontre o arco  $(u, v) \in A$  tal que  $d_T(u) = 0$  e  $d_T(v) + l(v) \neq d$  e cuja aresta
     associada  $(u, v)$  tenha valor  $w_{sOD}$  mínimo; então, adicione a aresta  $(u, v)$  a  $T$ .
5    $m \leftarrow m + 1$ 
6 até não há arco que satisfaça esta condição
7 repita
8   Considere os critérios (a) – (f), cujas prioridades estão em ordem decrescente
     ((a) tem a prioridade mais alta). Encontre um arco  $(u, v)$  em  $A$  com aresta
     associada  $(u, v)$  tal que  $T \cup (u, v)$  seja uma árvore e nenhum outro arco
     satisfaça um critério de prioridade mais alto.

     (a) arco  $(u, v)$  tem valor máximo  $d_G(v) + l(v) + n \times f_D(v)$  tal que
          $d_T(u) + l(u) < d, d_T(v) + l(v) > d$ .

     (b) aresta  $(u, v)$  tem  $w_{aOD}$  máximo em  $T$ , e arco  $(u, v)$  com
          $d_T(u) + l(u) > d, d_T(v) + l(v) > d$ .

     (c) aresta  $(u, v)$  tem mínimo  $w_{sOD}$  em  $T$  e arco  $(u, v)$  com
          $d_T(u) + l(u) < d, d_T(v) + l(v) < d$ .

     (d) aresta  $(u, v)$  tem  $w_{aOD}$  máximo em  $T$  e arco  $(u, v)$  com
          $d_T(u) + l(u) > d, d_T(v) + l(v) = d$ .

     (e) aresta  $(u, v)$  tem  $w_{aOD}$  máximo em  $T$  e arco  $(u, v)$  com
          $d_T(u) + l(u) < d, d_T(v) + l(v) = d$ .

     (f) aresta  $(u, v)$  tem  $w_{aOD}$  máximo em  $T$  e arco  $(u, v)$  com
          $d_T(u) + l(u) = d, d_T(v) + l(v) = d$ .

9   Adicionar aresta  $(u, v)$  em  $T$ 
10   $m \leftarrow m + 1$ 
11 até  $m = |V_G| - 1$ 
12 retorne  $(T)$ 

```

nenhum vértice for encontrado no critério (b), um arco é selecionado se ambos os vértices ainda não forem d -ramificações em T de tal forma que quando uma aresta é inserida em T nenhum deles se torna uma d -ramificação (critério (c)). Esse critério prioriza arcos cujos vértices possuem grau baixo, deixando que os vértices com graus maiores sejam analisados posteriormente. Além disso, os critérios (d), (e) e (f) escolhem arcos cujas arestas associadas criarão novas d -ramificações, (uma ao usar (d) ou (e), e duas ao usar

(f)). Esses critérios selecionam arcos cujos vértices possuem grau alto; assim, quando um vértice passa a ser uma d -ramificação, há uma chance de que novas arestas incidentes a este vértice sejam escolhidas nas próximas seleções. A aresta associada ao arco selecionado na linha 8 será inserida na árvore T .

4 Novas Heurísticas Construtivas

Neste capítulo são apresentadas oito novas heurísticas para a solução do MBV. Os algoritmos são semelhantes entre si, mas cada uma apresenta particularidades em relação aos critérios usados na construção das árvores geradoras. Essas heurísticas foram inspiradas no Algoritmo de Prim ([PRIM, 1957](#)), que busca uma árvore geradora mínima de um grafo conexo, não direcionado e valorado. O Algoritmo de Prim cultiva uma subárvore do grafo até que ela se torne geradora. Embora os novos métodos tenham se baseado nesse crescimento, o cultivo realizado pelos algoritmos permite que em uma determinada iteração existam várias subárvores, que convergirão em uma única árvore geradora ao final do processo.

Os métodos heurísticos desenvolvidos contemplam uma etapa inicial de pré-processamento que visa mapear o grafo, identificando suas pontes e articulações. A partir desse resultado, as heurísticas podem identificar de antemão arestas que devem estar contidas obrigatoriamente na árvore geradora e articulações especiais que invariavelmente serão ramificações na árvore. A Subseção [4.1](#) apresenta essa etapa de pré-processamento do grafo.

4.1 Pré-processamento

A identificação das pontes e articulações do grafo de entrada é o primeiro passo do pré-processamento, e pode ser realizada em tempo $O(n + m)$. Tal identificação é alcançada através de um algoritmo de busca em profundidade desenvolvido por ([HOPCROFT; TARTAGIAN, 1973](#)).

Dentre o conjunto das articulações detectadas, um subconjunto apresenta interesse particular: articulações cuja remoção implique na existência de três ou mais componentes conexas (vale frisar que o grafo inicial é sempre conexo). Uma articulação v que atenda a essa regra será obrigatoriamente uma ramificação na árvore, uma vez que todo caminho entre os vértices de componentes conexas diferentes passa obrigatoriamente por v e existem

pelo menos três componentes conexas diferentes. Dessa forma, fica evidente que o vértice v terá pelo menos grau 3 na árvore. Todo vértice v que atenda a essa condição é marcado como uma ramificação da árvore para processamento futuro.

O próximo passo do algoritmo é incorporar cada ponte (u,v) do grafo na árvore em construção. Cabe ressaltar que os vértices que porventura apresentarem três ou mais vizinhos durante a incorporação das pontes já foram marcados anteriormente como ramificações, pois integram o conjunto de articulações cuja remoção implica na existência de três ou mais componentes conexas.

Considerando que os vértices marcados como ramificações já excederam ou irão exceder obrigatoriamente o limite de grau para transformação em ramificação, e que não existe nenhuma condição inerente ao problema que vise minimizar o grau das ramificações, optamos por maximizar o grau desses vértices na árvore. Esse critério visa permitir vários pontos em que a árvore em construção pode ser cultivada (ou aumentada). Essa operação foi chamada de *Expansão* e é detalhada na Subseção 4.2.

Um caso particular do algoritmo de pré-processamento ocorre quando o grafo não contém pontes ou articulações que atendem às condições apresentadas anteriormente. Considerando que esses vértices constituem uma subárvore inicial que será cultivada pelos algoritmos, sua ausência impossibilita a continuidade do algoritmo. Desta forma, para contornar esses casos, o algoritmo identifica uma aresta (v,u) do grafo, tal que v é um vértice de grau mínimo no grafo e u é um vizinho de v com o menor grau possível. A aresta (v,u) escolhida e seus extremos v e u são adicionados à solução em construção. Considerando que o grafo é conexo, um desses vértices certamente possibilitará o cultivo (e expansão) da árvore.

O resultado dessa etapa de pré-processamento consiste na solução em construção T , com os vértices e arestas adicionados anteriormente (ramificações, pontes e vértices processados pelo Algoritmo de Expansão), e de duas estruturas de dados auxiliares B_T e $Pontas$. A primeira contém todas as ramificações incorporadas à árvore ao longo do pré-processamento, e a segunda os vértices de T com apenas um vizinho na árvore e com pelo menos dois vizinhos no grafo. Vale observar que esses vértices constituem pontos em que a árvore pode ser aumentada pela simples inclusão de vizinhos. O pré-processamento corresponde ao Algoritmo 5.

Algoritmo 5: Pré - Processamento**Entrada:** Grafo $G = (V_G, E_G)$ **Saída:** Floresta $T = (V_T, E_T)$, Conjunto de ramificações B_T , Conjunto de vértices $Pontas$ **Função** Pré-Processamento(G)

```

// Inicialização das Estruturas de Dados:
1   $V_T \leftarrow \emptyset$      $E_T \leftarrow \emptyset$      $B_T \leftarrow \emptyset$      $Pontas \leftarrow \emptyset$ 

// Tratar articulações especiais:
2   $A_G \leftarrow \{v \in V_G \mid w(G - v) \geq w(G) + 2\}$ 
3   $V_T \leftarrow A_G$                                      // Adicionar vértices na árvore
4   $B_T \leftarrow A_G$                                      // Marcar que todos serão ramificações da árvore

// Tratar pontes:
5   $P_G \leftarrow \{(v,u) \in E_G \mid w(G - (v,u)) > w(G)\}$ 
6  para  $(v,u) \in P_G$  faça
7      se  $v \notin V_T$  então  $V_T \leftarrow V_T \cup \{v\}$ 
8      se  $u \notin V_T$  então  $V_T \leftarrow V_T \cup \{u\}$ 
9       $E_T \leftarrow E_T \cup (v,u)$ 

// Expansão - Tratar vértices marcados como ramificações em  $T$ :
10 para cada  $v \in B_T$  faça
11     ExpandirRamificação( $G, T, v$ )

// Verificar quais caminhos podem ser explorados:
12 para cada  $v \in V_T$  faça
13     se  $d_T(v) = 1 \wedge d_G(v) > 1$  então  $Pontas \leftarrow Pontas \cup \{v\}$ 

// Se nenhuma ponta foi identificada, cria uma nova:
14 se  $Pontas = \emptyset$  então
15     Seja  $v$  o vértice de menor grau em  $V_G$ 
16     Seja  $u$  o vértice de menor grau em  $adj_G(v)$ 
17      $V_T \leftarrow V_T \cup \{v, u\}$      $E_T \leftarrow E_T \cup (v,u)$ 
18      $Pontas \leftarrow Pontas \cup \{v, u\}$ 
19 retorne  $(T, B_T, Pontas)$ 

```

4.2 Processo de Expansão de Ramificações

Como mencionado anteriormente, o MBV não possui nenhum critério que busque minimizar o grau das ramificações, sendo este um problema à parte conhecido como “*Problem of Minimizing the Degree Sum of Branch Vertices*” ou DMS, que foi proposto e estudado por (CERULLI; GENTILI; IOSSA, 2009). Desta forma, sempre que um vértice se torna uma ramificação, uma das abordagens possíveis é explorar ao máximo seu potencial na árvore.

Visando esse objetivo, estabelecemos uma rotina chamada **ExpandirRamificação** que, dados o grafo $G = (V_G, E_G)$, a floresta (solução em construção) $T = (V_T, E_T)$ e uma nova ramificação v , adiciona na árvore T todas as arestas (v, u) incidentes a v em $E_G \setminus E_T$ que satisfaçam um dos seguintes critérios: (i) $u \notin V_T$ ou (ii) $CConexa(v) \neq CConexa(u) \wedge (d_T(u) = 1 \vee u \in B_T)$. O primeiro critério visa permitir a adição de novos vértices à árvore, enquanto o segundo visa incorporar novas arestas sem adicionar ciclos ou criar novas ramificações. O processo de Expansão de Ramificações foi documentado como o Algoritmo 6.

Para verificar se dois vértices estão em uma mesma componente conexa da árvore T , definimos uma estrutura de dados atrelada à árvore T chamada *CConexa*. Essa estrutura mantém controle sobre todas as componentes conexas da árvore e os vértices nela contidos. Essa estrutura será utilizada ao longo dos próximos algoritmos para identificar se uma aresta é capaz de fechar um ciclo na árvore (arestas com ambos os extremos em uma mesma componente conexa). A atualização dessa estrutura é feita de forma implícita sempre que a árvore é modificada.

Algoritmo 6: Expansão de Ramificações

Entrada: Grafo $G = (V_G, E_G)$, Floresta $T = (V_T, E_T)$, Vértice v

Saída: Floresta $T = (V_T, E_T)$ atualizada

Procedimento ExpandirRamificação(G, T, v)

```

1  para cada  $u \in adj_G(v)$  faça
2      se  $u \notin V_T$  então
3          //  $u$  não está na árvore:
4           $V_T \leftarrow V_T \cup \{u\}$ 
5           $E_T \leftarrow E_T \cup (v, u)$ 
6      senão
7          //  $u$  já está na árvore: necessário verificar se  $(v, u)$  não
          // cria ciclos ou transforma  $u$  em ramificação:
          se  $(d_T(u) = 1 \vee u \in B_T) \wedge CConexa(v, T) \neq CConexa(u, T)$  então
               $E_T \leftarrow E_T \cup (v, u)$ 

```

4.3 Algoritmos Gulosos

As quatro primeiras heurísticas propostas nesse trabalho são algoritmos gulosos, cujas descrições estão nas subseções a seguir. Cada algoritmo recebe um grafo conexo G e constrói uma árvore geradora T para G , buscando minimizar o número de ramificações em T .

4.3.1 *Branch Expanding Prim*

A primeira heurística a ser apresentada é o Algoritmo BEP (*Branch Expanding Prim*). Este algoritmo serviu como base para o desenvolvimento das demais heurísticas gulosas propostas neste trabalho. A primeira etapa do Algoritmo BEP é executar o pré-processamento, discutido anteriormente. Desta forma, dado o grafo G , o pré-processamento retorna uma árvore (ou floresta) em construção T , o conjunto de ramificações B_T e o conjunto $Pontas$, que inclui os vértices onde a árvore pode ser cultivada.

Em seguida, dois procedimentos serão adotados até que T seja uma árvore geradora para G . O primeiro considera que existe uma ponta que pode ser cultivada ($Pontas \neq \emptyset$). Nesse caso, um vértice v de menor grau em $Pontas$ será selecionado e removido deste conjunto. Em seguida, será escolhido um vizinho u de v em G de menor grau, tal que a aresta (v,u) não gere uma nova ramificação ou um ciclo em T . Em caso de empate entre vértices vizinhos de menor grau, terá preferência aquele que apresentar o menor grau na árvore (vale ressaltar que assumimos como zero o grau de um vértice que ainda não está na árvore, implicando que esses vértices terão vantagem sobre os demais no critério de desempate). A cada adição de uma nova aresta na árvore, o conjunto $Pontas$ é atualizado, em busca de novas opção para o crescimento que possam ter surgido na árvore.

Já o segundo procedimento é adotado quando o conjunto $Pontas$ se torna vazio e a árvore T ainda não está completa. Quando isso ocorre, não há nenhuma folha ou ramificação na árvore que possa estabelecer novas conexões entre vértices que ainda não estão na árvore ou entre componentes conexas distintas. Desta forma, a única conclusão possível é que para continuar cultivando a árvore é necessário transformar pelo menos um vértice em ramificação.

Para verificar se existe um vértice que possibilite a continuidade do algoritmo, se transformando em ramificação, o algoritmo busca dentre os vértices da árvore um vértice v que maximize a expressão $C(v) = |\{u \in adj_G(v) \mid u \notin V_T \vee (CConexa(u,T) \neq CConexa(v,T) \wedge (u \in B_T \vee d_T(u) = 1))\}|$. A expressão $C(v)$ define uma estimativa de quantos vizinhos de v podem ser conectados à árvore T caso v se torne uma ramificação. As condições expressas em $C(v)$ consideram apenas vizinhos de v que ainda não estão na árvore ou que estão em componentes conexas diferentes de v e são folhas ou ramificações em T . No entanto, vale observar que a estimativa definida por $C(v)$ é um limite máximo, pois apenas um vértice de cada componente conexa pode ser conectado a v sem criar ciclos. Seja v o vértice que maximiza $C(v)$.

Se $C(v) > 0$ então é possível continuar o algoritmo transformando v em ramificação. Considerando que a Expansão de Ramificações (definida na Seção 4.2) já define o tratamento dado às novas ramificações, esse método é executado para v . Considerando que esse método pode ter liberado novos pontos em que a árvore pode ser cultivada, o algoritmo varre todos os vizinhos de v em busca desses vértices e os adiciona em $Pontas$.

Entretanto, existem casos em que a transformação de um único vértice em ramificação ainda é insuficiente para permitir a continuação do algoritmo. Nesses casos, as únicas arestas que ainda restam no grafo conectam pares de vértices de grau 2 na árvore, implicando que $C(v) = 0$ para todo vértice v da árvore.

Desta forma, para continuar o algoritmo é necessário converter pelo menos dois vértices em ramificações. Novamente, visando identificar o vértice v que possibilitará o maior número de novas conexões (vizinhos em componentes conexas distintas), definimos uma nova expressão $CC(v) = |\{u \in adj_G(v) \mid CConexa(u, T) \neq CConexa(v, T)\}|$. Vale observar que $CC(v)$ é uma versão menos restrita de $C(v)$, pois não são consideradas as limitações de grau. Seja v o vértice que maximiza $CC(v)$.

Considerando que o caso anterior, onde $C(v) > 0$, é capaz de processar as novas conexões liberadas ao transformar v em ramificação e que o procedimento de Expansão seria inócua nesse caso (se houvesse novas conexões possíveis ao transformar somente v em ramificação, o valor de $C(v)$ seria diferente de 0), optamos apenas por marcar v como ramificação (incluir em B_T), sem adicionar novas arestas ou elevar seu grau. Essa condição será corrigida na próxima iteração, pois um vizinho u de v atenderá a condição $C(u) > 0$.

Em relação à corretude do método, vale destacar que o algoritmo impede a adição de ciclos e assegura que T tenha exatamente $n - 1$ arestas ($n = |V_G|$). Desta forma, fica evidente que todo vértice de G será eventualmente adicionado em T , pois do contrário não seria possível adicionar todas as $n - 1$ arestas sem criar ciclos. Logo, o algoritmo retorna uma árvore geradora T para G . O Algoritmo 7 apresenta o método descrito.

A Figura 9 ilustra a execução do Algoritmo BEP em um grafo conexo com $n = 11$ e $m = 13$ (Figura 9(a)). Inicialmente, o algoritmo identifica todas as pontes e ramificações obrigatórias do grafo. As pontes são então incorporadas à solução em construção, enquanto as ramificações são marcadas para o processo de expansão. O resultado deste processo pode ser visto na Figura 9(b): as pontes foram adicionadas ao grafo e a ramificação obrigatória, marcada em azul. Em seguida, as ramificações marcadas anteriormente passam pelo processo da Expansão (detalhado na Subseção 4.2), ilustrado na Figura 9(c). Após essa etapa inicial, o algoritmo identifica e marca todo vértice w que satisfaz

Algoritmo 7: BEP - *Branch Expanding Prim*

Entrada: Grafo $G = (V_G, E_G)$
Saída: Árvore Geradora $T = (V_T, E_T)$

```

1   $(T, B_T, Pontas) \leftarrow \text{Pré-Processamento}(G)$ 
2  enquanto  $|E_T| < |V_G| - 1$  faça
3      se  $Pontas \neq \emptyset$  então
4          Seja  $v$  o vértice de menor grau  $d_G(v)$  em  $Pontas$ 
5           $Pontas \leftarrow Pontas \setminus \{v\}$ 
6          Seja  $N_v = \{u \in adj_G(v) \mid u \notin V_T \vee$ 
               $(CConexa(u, T) \neq CConexa(v, T) \wedge (u \in B_T \vee d_T(u) = 1))\}$ 
7          se  $N_v \neq \emptyset$  então
8              Seja  $u$  o vértice de menor grau  $d_G(u) \in N_v$ 
9              (em caso de empate, o algoritmo escolhe algum  $u \notin V_T$ )
10             se  $u \notin V_T$  então  $V_T \leftarrow V_T \cup \{u\}$ 
11              $E_T \leftarrow E_T \cup (v, u)$ 
12             se  $d_T(u) = 1$  então  $Pontas \leftarrow Pontas \cup \{u\}$ 
13             senão
14                 se  $u \in Pontas$  então  $Pontas \leftarrow Pontas \setminus \{u\}$ 
15         senão
16             // Não existe uma nova ponta para ser explorada: necessário
17             converter um vértice da árvore em ramificação.
18             Seja  $v \in V_T$  um vértice que maximiza  $C(v) = |\{u \in adj_G(v) \mid u \notin V_T \vee$ 
19              $(CConexa(u, T) \neq CConexa(v, T) \wedge (u \in B_T \vee d_T(u) = 1))\}|$ 
20             se  $C(v) > 0$  então
21                 // Transformar  $v$  em ramificação permite estabelecer
22                 até  $C(v)$  novas conexões:
23                  $B_T \leftarrow B_T \cup \{v\}$ 
24                 ExpandirRamificação( $G, T, v$ )
25                 para  $u \in adj_G(v)$  faça
26                     se  $d_T(u) = 1 \wedge d_G(u) > 1$  então
27                          $Pontas \leftarrow Pontas \cup \{u\}$ 
28             senão
29                 // Transformar um único vértice em ramificação não conecta
30                 duas subárvores, mas garante que vizinhos de  $v$ 
31                 atenderão à condição da linha 16 nas próximas
32                 iterações.
33             Seja  $v \in V_T$  um vértice que maximiza
34              $CC(v) = |\{u \in adj_G(v) \mid CConexa(u, T) \neq CConexa(v, T)\}|$ 
35              $B_T \leftarrow B_T \cup \{v\}$ 
36 retorne  $(T)$ 

```

$d_T(w) = 1$ e $d_G(w) > 1$ (Figura 9(d)). Os vértices que atendem à condição apresentada foram marcados em verde e correspondem ao conjunto $Pontas$. Nas iterações seguintes

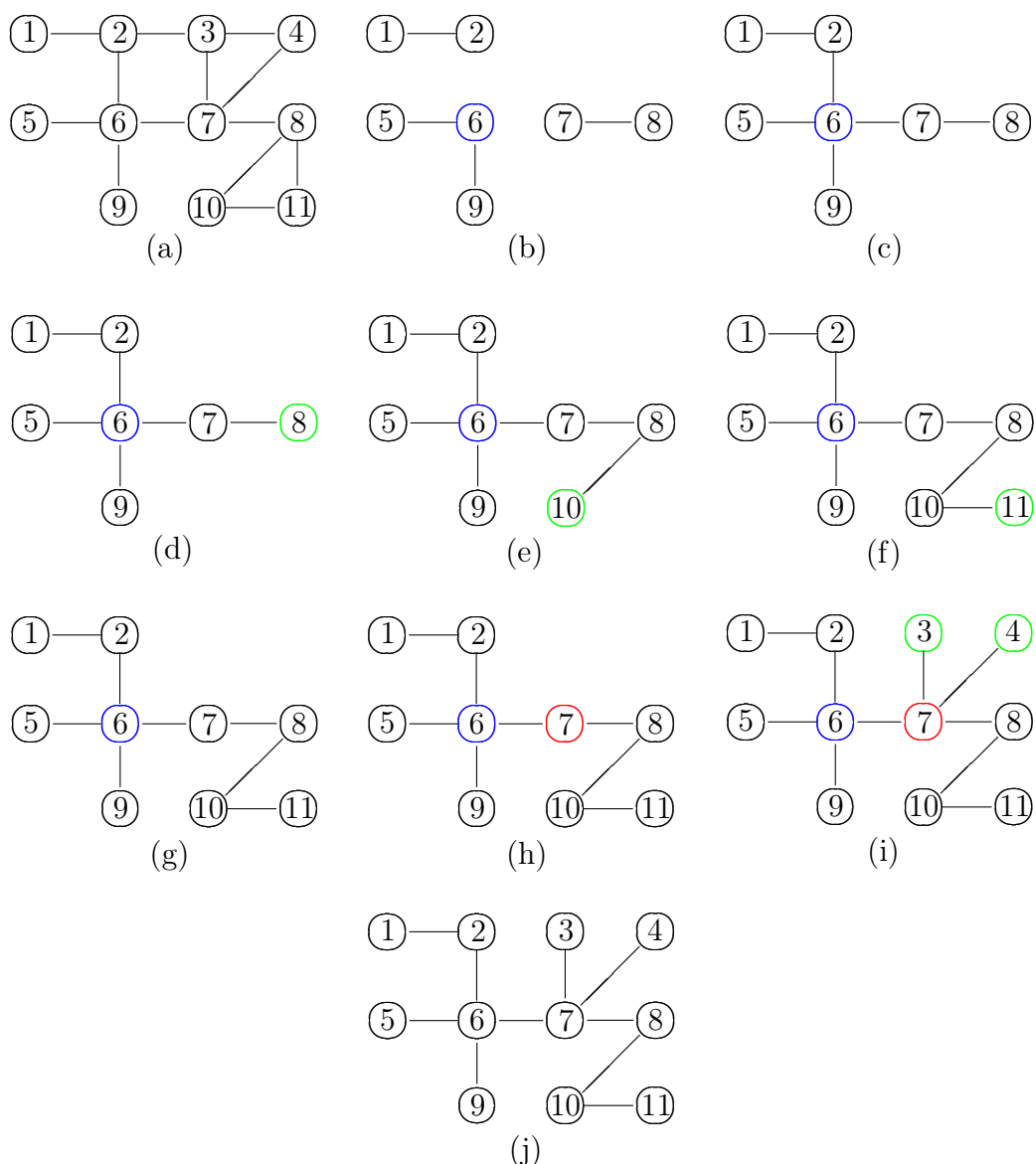


Figura 9: Exemplo de Aplicação do Algoritmo BEP (7)

(Figura 9(e), (f)), o algoritmo seleciona um vértice v de menor grau em *Pontas* e busca por um vizinho u de v cuja adição não crie ciclos nem ramificações na árvore, dando prioridade aos vizinhos de grau mínimo. O vértice selecionado é então adicionado à solução em construção e o conjunto *Pontas* é atualizado. Durante esse processo, um caso merece atenção: na Figura 9(f), a ponta (vértice 11) recém adicionada não possui um vizinho u que satisfaça as condições necessárias e, portanto, foi simplesmente descartado do conjunto *Pontas* como ilustra a figura seguinte (9(g)). Tendo o conjunto *Pontas* ficado vazio, para continuar a construção da solução é necessário converter um vértice da árvore em ramificação para formar novas conexões. Para isso, será selecionado um vértice i que maximize o número de vizinhos j de i fora da árvore ou em componentes conexas diferentes com $d_T(j) = 1$ ou $d_T(j) \geq 3$. Entre as opções disponíveis (vértices 2 e 7), o vértice 7 será

selecionado e marcado como ramificação (Figura 9(h)). Em seguida, o vértice 7 (marcado em vermelho) passará pelo processo de Expansão, inserindo os vértices 3 e 4 na árvore, como ilustra a Figura 9(i). Os vértices 3 e 4 serão incluídos no conjunto *Pontas*, uma vez que atendem à sua condição. A árvore geradora resultante é apresentada na Figura 9(j) e corresponde a uma solução ótima para a instância em estudo.

4.3.2 *Edge Expanding Prim*

O Algoritmo EEP (*Edge Expanding Prim*) é uma variação do Algoritmo BEP, cujo objetivo é explorar uma diferença na ordem de adição de arestas na árvore durante o tratamento do conjunto *Pontas*. O EEP adota exatamente os procedimentos e conjuntos vistos anteriormente no Algoritmo BEP, variando contudo os critérios de seleção de arestas (linhas 6 e 8 do Algoritmo 7).

Durante o tratamento dos vértices em *Pontas*, enquanto o Algoritmo BEP escolhe um vértice v de menor grau em G em *Pontas* e posteriormente um dos vizinhos u de v para, em seguida, adicionar na árvore a aresta (v,u) , o Algoritmo EEP utiliza diretamente nas arestas que considera um peso calculado pela soma dos graus de seus extremos.

Enquanto existir algum vértice em *Pontas*, o Algoritmo EEP constrói um conjunto restrito de arestas do grafo $(a,b) \in E_G$ que satisfazem as seguintes condições (i) $a \in Pontas$ e (ii) $b \in adj_G(a) \mid b \notin V_T \vee (CConexa(b,T) \neq CConexa(a,T) \wedge (b \in B_T \vee d_T(b) = 1))$. Esse conjunto, chamado de *ACandidatas*, contém todas as arestas do grafo com um extremo em *Pontas* cuja adição em T não implique em ciclos ou novas ramificações. Se esse conjunto *ACandidatas* não for vazio, é possível cultivar a árvore adicionado uma dessas arestas. Caso contrário, os vértices de pontas não permitem crescer a árvore sem adicionar novas ramificações. Essa conclusão permite descartar todo o conjunto *Pontas*, acelerando um processo que levaria várias iterações no BEP.

Retornando ao caso em que existem arestas que atendam às condições anteriormente definidas, o algoritmo escolhe a aresta (v,u) que minimize a soma dos graus dos vértices v e u em G ($d_G(v) + d_G(u)$). Caso ocorra empate entre os pesos, o desempate será realizado da mesma maneira vista no Algoritmo BEP, dando prioridade aos vértices não presentes em T . Esse critério visa priorizar os vértices de menor grau, deixando os de maior grau para depois, pois um número maior de arestas aparentemente torna esses vértices mais propensos a estabelecer novas conexões com a árvore mesmo quando os demais (de menor grau) já não admitem essas conexões.

Algoritmo 8: EEP - *Edge Expanding Prim***Entrada:** Grafo $G = (V_G, E_G)$ **Saída:** Árvore Geradora $T = (V_T, E_T)$

```

1   $(T, B_T, Pontas) \leftarrow \text{Pré-Processamento}(G)$ 
2  enquanto  $|E_T| < |V_G| - 1$  faça
3      se  $Pontas \neq \emptyset$  então
4          Seja  $ACandidatas$  o conjunto de arestas  $(a,b) \in E_G$  que satisfazem
               $a \in Pontas$  e  $b \in adj_G(a) \mid b \notin V_T \vee (CConexa(b,T) \neq CConexa(a,T) \wedge$ 
               $(b \in B_T \vee d_T(b) = 1))$ 
5          se  $ACandidatas \neq \emptyset$  então
6              Seja  $(v,u)$  uma aresta de  $ACandidatas$  que minimiza  $d_G(v) + d_G(u)$ 
7              (em caso de empate, o algoritmo escolhe uma aresta tal que  $u \notin V_T$ )
8               $Pontas \leftarrow Pontas \setminus \{v\}$ 
9              se  $u \notin V_T$  então  $V_T \leftarrow V_T \cup \{u\}$ 
10              $E_T \leftarrow E_T \cup (v,u)$ 
11             se  $d_T(u) = 1$  então  $Pontas \leftarrow Pontas \cup \{u\}$ 
12             senão
13                 se  $u \in Pontas$  então  $Pontas \leftarrow Pontas \setminus \{u\}$ 
14             senão  $Pontas \leftarrow \emptyset$ 
15         senão
16             // Não existe uma nova ponta para ser explorada: necessário
                converter um vértice da árvore em ramificação.
17             Seja  $v \in V_T$  um vértice que maximiza  $C(v) = |\{u \in adj_G(v) \mid u \notin V_T \vee$ 
                 $(CConexa(u,T) \neq CConexa(v,T) \wedge (u \in B_T \vee d_T(u) = 1))\}|$ 
18             se  $C(v) > 0$  então
19                 // Transformar  $v$  em ramificação permite estabelecer
                    até  $C(v)$  novas conexões:
20                  $B_T \leftarrow B_T \cup \{v\}$ 
21                 ExpandirRamificação( $G, T, v$ )
22                 para  $u \in adj_G(v)$  faça
23                     se  $d_T(u) = 1 \wedge d_G(u) > 1$  então
24                          $Pontas \leftarrow Pontas \cup \{u\}$ 
25                 senão
26                     // Transformar um único vértice em ramificação não conecta
                        duas subárvores, mas garante que vizinhos de  $v$ 
                        atenderão à condição da linha 16 nas próximas
                        iterações.
27                     Seja  $v \in V_T$  um vértice que maximiza
                         $CC(v) = |\{u \in adj_G(v) \mid CConexa(u,T) \neq CConexa(v,T)\}|$ 
28                      $B_T \leftarrow B_T \cup \{v\}$ 
29         retorne  $(T)$ 

```

Após selecionar e adicionar a aresta (v,u) na árvore (floresta) T , o algoritmo verifica se $d_T(u) = 1$ e $d_G(u) > 1$. Em caso afirmativo, u é adicionado ao conjunto $Pontas$,

pois é um novo ponto de cultivo na árvore. Esse processo é repetido até que *Pontas* se torne vazio ou até que T se torne uma árvore geradora para G . Os casos em que *Pontas* se torna vazio seguem idênticos aos descritos no Algoritmo BEP. Considerando que a modificação no critério utilizado para selecionar arestas ainda assegura que T não conterá ciclos, pode-se afirmar que a demonstração da corretude do BEP pode ser estendida para o Algoritmo EEP, uma vez que as demais premissas seguem válidas. O Algoritmo EEP é detalhado como o Algoritmo 8.

4.3.3 *Controlled Expanding Prim*

O Algoritmo CEP (*Controlled Expanding Prim*) também é uma variação do Algoritmo BEP, cujo objetivo é possuir um controle maior no processo de expansão de ramificações. O CEP adota exatamente os mesmos procedimentos e conjuntos vistos anteriormente no Algoritmo BEP, possuindo pequenas diferenças no seu método, sendo a mais importante delas a exclusão do processo de Expansão (linha 19 do Algoritmo 7).

Assim como no BEP, o CEP realizará o processo de Expansão no conjunto B_T , que a princípio possuirá apenas as ramificações obrigatórias de T , durante o pré-processamento. Entretanto, durante a etapa de converter um vértice em ramificação, não haverá novas chamadas ao processo de Expansão. Após encontrar o vértice v (linha 16 do Algoritmo 9) para transformá-lo em ramificação, ele será inserido no conjunto B_T , e, em vez de expandi-lo, o algoritmo insere v no conjunto *Pontas* (linha 19 do Algoritmo 9), podendo assim alcançar novos vértices.

A partir do momento que o conjunto *Pontas* deixa de ser vazio, o algoritmo parte para tratá-lo. O tratamento ocorrerá da mesma maneira vista no BEP, mas no CEP, quando uma ramificação entra em *Pontas*, ela só será removida do conjunto quando extinguir as opções de vértices aos quais ela pode se conectar durante o tratamento de *Pontas*. Vale recordar que durante o tratamento desse conjunto, a cada aresta inserida na árvore partindo de um vértice em *Pontas*, o conjunto será atualizado em busca de novas pontas que possam ter surgido na árvore.

Dessa forma, a árvore poderá crescer de maneira distinta em relação ao algoritmo BEP, viabilizando um arranjo de arestas diferente. Isso possibilita um número diferente de ramificações na árvore geradora T . O Algoritmo CEP é detalhado como o Algoritmo 9.

A Figura 10 ilustra a execução do Algoritmo CEP em um grafo conexo com $n = 11$ e $m = 13$ (Figura 10(a)). Como o CEP é baseado no Algoritmo BEP, e o grafo da

Algoritmo 9: CEP - *Controlled Expanding Prim*

Entrada: Grafo $G = (V_G, E_G)$
Saída: Árvore Geradora $T = (V_T, E_T)$

```

1   $(T, B_T, Pontas) \leftarrow \text{Pré-Processamento}(G)$ 
2  enquanto  $|E_T| < |V_G| - 1$  faça
3      se  $Pontas \neq \emptyset$  então
4          Seja  $v$  o vértice de menor grau  $d_G(v)$  em  $Pontas$ 
5          Seja  $N_v = \{u \in \text{adj}_G(v) \mid u \notin V_T \vee$ 
               $(CConexa(u, T) \neq CConexa(v, T) \wedge (u \in B_T \vee d_T(u) = 1))\}$ 
6          se  $N_v \neq \emptyset$  então
7              Seja  $u$  o vértice de menor grau  $d_G(u) \in N_v$ 
8              (em caso de empate, o algoritmo escolhe algum  $u \notin V_T$ )
9              se  $u \notin V_T$  então  $V_T \leftarrow V_T \cup \{u\}$ 
10              $E_T \leftarrow E_T \cup (v, u)$ 
11             se  $d_T(u) = 1$  então  $Pontas \leftarrow Pontas \cup \{u\}$ 
12             senão
13                 se  $u \in Pontas$  então  $Pontas \leftarrow Pontas \setminus \{u\}$ 
14         se  $N_v = \emptyset \vee v \notin B_T$  então  $Pontas \leftarrow Pontas \setminus \{v\}$ 
15     senão
16         // Não existe uma nova ponta para ser explorada: necessário
            converter um vértice da árvore em ramificação.
17         Seja  $v \in V_T$  um vértice que maximiza  $C(v) = |\{u \in \text{adj}_G(v) \mid u \notin V_T \vee$ 
             $(CConexa(u, T) \neq CConexa(v, T) \wedge (u \in B_T \vee d_T(u) = 1))\}|$ 
18         se  $C(v) > 0$  então
19             // Transformar  $v$  em ramificação e adicioná-lo em  $Pontas$ 
                permite que  $v$  estabeleça novas conexões
20              $B_T \leftarrow B_T \cup \{v\}$ 
21              $Pontas \leftarrow Pontas \cup \{v\}$ 
22         senão
23             // Transformar um único vértice em ramificação não conecta
                duas subárvores, mas garante que vizinhos de  $v$ 
                atenderão à condição da linha 17 nas próximas
                iterações.
24             Seja  $v \in V_T$  um vértice que maximiza
                 $CC(v) = |\{u \in \text{adj}_G(v) \mid CConexa(u, T) \neq CConexa(v, T)\}|$ 
25              $B_T \leftarrow B_T \cup \{v\}$ 
26  retorne  $(T)$ 

```

Figura 10(a) é idêntico ao da Figura 9(a), não será necessário repetir a explicação das etapas ilustradas pela Figura 10(b) - (g) do Algoritmo CEP, uma vez que são as mesmas do Algoritmo BEP, já explicadas na Subseção 4.3.1. Partindo da Figura 10(h), onde o conjunto *Pontas* está vazio e, para continuar a construção da solução, é necessário converter um vértice da árvore em ramificação, será selecionado um vértice i que maximize

o número de vizinhos j de i fora da árvore ou em componentes conexas diferentes com $d_T(j) = 1$ ou $d_T(j) \geq 3$. O vértice 7 é selecionado para se converter em ramificação, sendo inserido no conjunto *Pontas* para alcançar novos vértices. Para identificação, os vértices contidos no conjunto *Pontas* são marcados com verde na Figura 10. Após isso, o algoritmo processa o conjunto *Pontas*, selecionando um vértice v de menor grau em *Pontas* e buscando por um vizinho u de v cuja adição não crie ciclos nem ramificações na árvore, dando prioridade aos vizinhos u de grau mínimo (Figura 10(i), (j)). O vértice u é então adicionado à solução em construção e o conjunto *Pontas* é atualizado. Uma ramificação presente no conjunto *Pontas* só será removida do mesmo quando extinguir suas opções de conexão na árvore em construção, diferentemente dos demais vértices do conjunto, que são removidos após serem selecionados em uma iteração do processamento do conjunto *Pontas*. A árvore geradora resultante é apresentada na Figura 10(k) e corresponde a uma solução ótima para a instância em estudo.

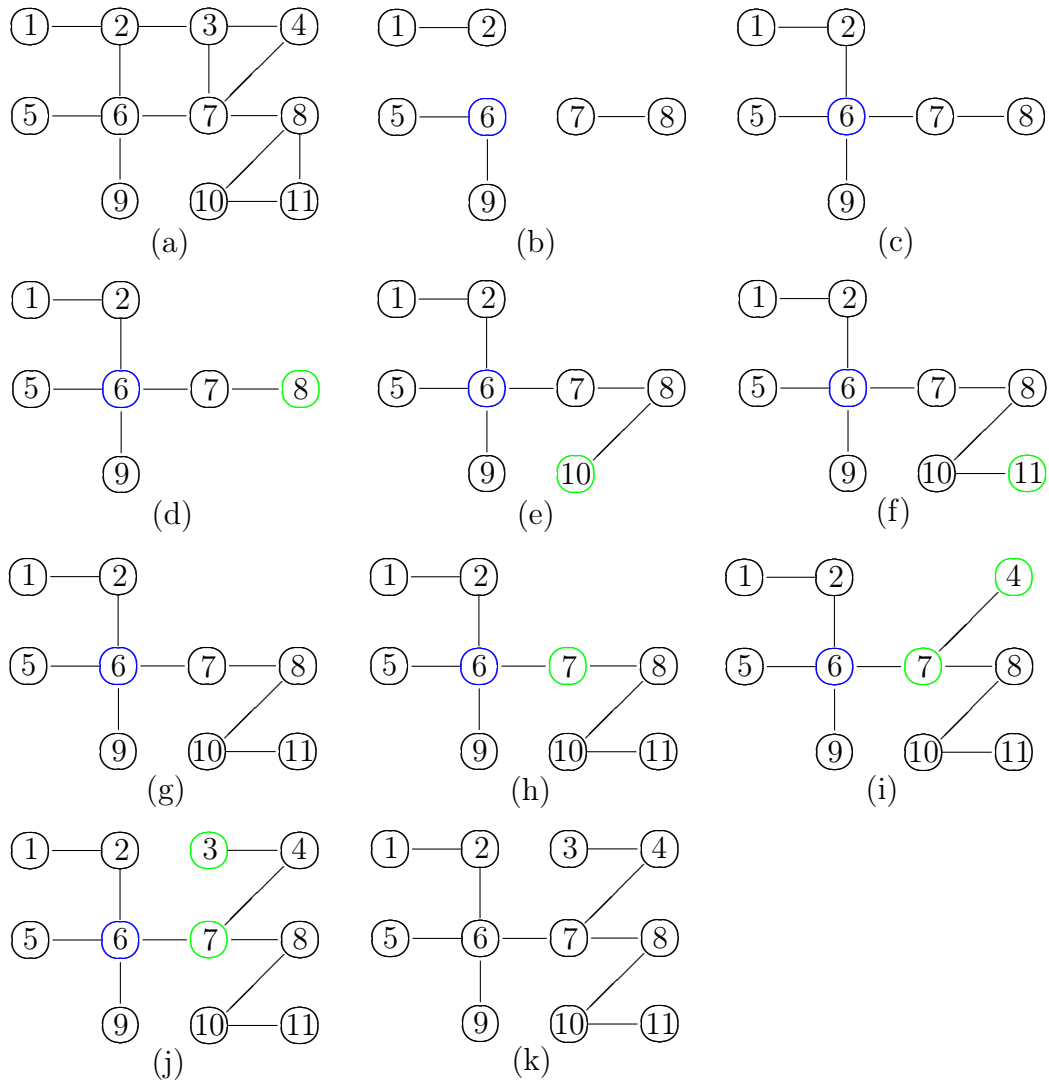


Figura 10: Exemplo de Aplicação do Algoritmo CEP (9)

4.3.4 *Controlled Edge Expanding Prim*

O algoritmo CEEP (*Controlled Edge Expanding Prim*) é uma variação do algoritmo EEP, na qual há um controle maior no processo de Expansão, assim como ocorre no algoritmo CEP.

O CEEP, assim como os demais algoritmos, possui sua base no algoritmo BEP, adotando os mesmos procedimentos e conjuntos vistos anteriormente, mas contém ambas as modificações feitas pelos algoritmos CEP e EEP. Ele possui tanto um controle maior sobre o processo de Expansão apresentado no CEP, quanto a utilização de pesos nas arestas para critério de seleção durante o tratamento do conjunto *Pontas* visto em EEP.

Tendo ele ambas as modificações em relação ao BEP, é possível analisar as diferentes árvores geradoras que o CEEP pode proporcionar em relação aos algoritmos anteriores. O Algoritmo CEEP é detalhado como o Algoritmo 10. O CEEP encerra a lista de algoritmos gulosos a serem vistos.

4.4 Algoritmos Randomizados

As quatro heurísticas descritas a seguir são versões randomizadas dos algoritmos gulosos vistos anteriormente. Visando possibilitar a exploração de novas árvores, os critérios de seleção de arestas adotados nos algoritmos BEP, EEP, CEP e CEEP foram modificados para considerar uma componente randômica nas decisões. Essa componente randômica permite que os algoritmos retornem árvores diferentes das encontradas pelas versões determinísticas dos algoritmos (BEP, EEP, CEP e CEEP).

Entretanto, visando preservar a tendência natural dos métodos apresentados anteriormente, optamos por uma componente capaz de combinar os critérios já definidos e uma fonte de randomização: uma roleta viciada.

Nessa roleta, a cada elemento é associada uma probabilidade proporcional aos critérios utilizados nos algoritmos. Elementos com maior probabilidade tendem a ser selecionados com maior frequência. Contudo, diferentemente da versão determinística, elementos de probabilidade reduzida ainda podem ser selecionados, permitindo estabelecer novas árvores geradoras que não seriam obtidas pelos algoritmos puramente gulosos.

Considerando que os algoritmos apresentados anteriormente usam critérios diferentes para selecionar as arestas que são adicionadas à árvore em construção, as subseções abaixo apresentam detalhadamente como a roleta viciada foi integrada aos Algoritmos BEP, EEP,

Algoritmo 10: CEEP - *Controlled Edge Expanding Prim*

Entrada: Grafo $G = (V_G, E_G)$
Saída: Árvore Geradora $T = (V_T, E_T)$

```

1   $(T, B_T, Pontas) \leftarrow \text{Pré-Processamento}(G)$ 
2  enquanto  $|E_T| < |V_G| - 1$  faça
3      se  $Pontas \neq \emptyset$  então
4          Seja  $ACandidatas$  o conjunto de arestas  $(a,b) \in E_G$  que satisfazem
               $a \in Pontas$  e  $b \in adj_G(a) \mid b \notin V_T \vee (CConexa(b,T) \neq CConexa(a,T) \wedge$ 
               $(b \in B_T \vee d_T(b) = 1))$ 
5          se  $ACandidatas \neq \emptyset$  então
6              Seja  $(v,u)$  uma aresta de  $ACandidatas$  que minimiza  $d_G(v) + d_G(u)$ 
7              (em caso de empate, o algoritmo escolhe uma aresta tal que  $u \notin V_T$ )
8              se  $v \notin B_T$  então  $Pontas \leftarrow Pontas \setminus \{v\}$ 
9              se  $u \notin V_T$  então  $V_T \leftarrow V_T \cup \{u\}$ 
10              $E_T \leftarrow E_T \cup (v,u)$ 
11             se  $d_T(u) = 1$  então  $Pontas \leftarrow Pontas \cup \{u\}$ 
12             senão
13                 se  $u \in Pontas$  então  $Pontas \leftarrow Pontas \setminus \{u\}$ 
14             senão  $Pontas \leftarrow \emptyset$ 
15         senão
16             // Não existe uma nova ponta para ser explorada: necessário
                converter um vértice da árvore em ramificação.
17             Seja  $v \in V_T$  um vértice que maximiza  $C(v) = |\{u \in adj_G(v) \mid u \notin V_T \vee$ 
                 $(CConexa(u,T) \neq CConexa(v,T) \wedge (u \in B_T \vee d_T(u) = 1))\}|$ 
18             se  $C(v) > 0$  então
19                 // Transformar  $v$  em ramificação e adicioná-lo em  $Pontas$ 
                    permite que  $v$  estabeleça novas conexões
20                  $B_T \leftarrow B_T \cup \{v\}$ 
21                  $Pontas \leftarrow Pontas \cup \{v\}$ 
22             senão
23                 // Transformar um único vértice em ramificação não conecta
                    duas subárvores, mas garante que vizinhos de  $v$ 
                    atenderão à condição da linha 16 nas próximas
                    iterações.
24                 Seja  $v \in V_T$  um vértice que maximiza
                     $CC(v) = |\{u \in adj_G(v) \mid CConexa(u,T) \neq CConexa(v,T)\}|$ 
25                  $B_T \leftarrow B_T \cup \{v\}$ 
26     retorne  $(T)$ 

```

CEP e CEEP, respectivamente, resultando nos Algoritmos R-BEP, R-EEP, R-CEP e R-CEEP.

4.4.1 Randomização dos algoritmos BEP e CEP

As versões randomizadas do BEP e CEP são os Algoritmos R-BEP (*Randomized Branch Expanding Prim*) e R-CEP (*Randomized Controlled Expanding Prim*) respectivamente. Durante o tratamento do conjunto *Pontas*, ao invés dos algoritmos escolherem o vértice de menor grau em G de *Pontas*, os vértices desse conjunto serão inseridos em uma roleta com suas respectivas probabilidades.

A probabilidade de um vértice v na roleta será calculada da seguinte maneira: $prob(v) = \max\{d_G(u), \forall u \in Pontas\} - d_G(v) + 1$. Essa condição estabelece que a probabilidade de um vértice ser selecionado é inversamente proporcional ao seu grau. Vale observar também que nenhum vértice terá probabilidade zero de ser selecionado (o vértice de maior grau terá probabilidade 1). Após definir as probabilidades, um vértice v é sorteado pela roleta.

Em seguida, os algoritmos verificam quais os vizinhos de v no grafo que podem ser conectados ao próprio v na árvore sem criar ciclos ou ramificações. Se um vértice $u \in adj_G(v)$ atende a essas condições, é adicionado a um conjunto de candidatos N_v . Em seguida, cada vértice de $u \in N_v$ é adicionado a uma segunda roleta com probabilidade $prob(u) = \max\{d_G(w), \forall w \in N_v\} - d_G(u) + fator$, onde $fator = 2$, se $u \notin V_T$, e $fator = 1$ caso contrário. Vale observar que a variável *fator* aumenta a probabilidade dos vértices que ainda não estão na árvore serem selecionados, sem impossibilitar que os demais também sejam. Após construir a segunda roleta, o algoritmo seleciona o vértice u e adiciona a aresta (v, u) na árvore. Vale observar que as roletas serão descartadas e reconstruídas a cada iteração dos algoritmos. Os casos em que $Pontas = \emptyset$ ou $N_v = \emptyset$ (v não apresenta nenhum vizinho que possa ser adicionado à árvore sem criar ramificações ou ciclos) seguem idênticos aos apresentados nos Algoritmos BEP e CEP. Além disso, as mudanças realizadas nos critérios de seleção não interferem com a correteza do método, uma vez que ainda preservam as condições originais. O R-BEP e R-CEP foram detalhados como os Algoritmos 11 e 12 respectivamente.

4.4.2 Randomização dos algoritmos EEP e CEEP

As versões randomizadas do EEP e CEEP são os Algoritmos R-EEP (*Randomized Edge Expanding Prim*) e R-CEEP (*Randomized Controlled Edge Expanding Prim*). Considerando que o EEP e o CEEP selecionam diretamente a aresta que será inserida na árvore, o R-EEP e o R-CEEP difere do R-BEP e do R-CEP ao usarem apenas uma roleta.

A inclusão da roleta no R-EEP e no R-CEEP é realizada logo após estabelecer o

Algoritmo 11: R-BEP - *Randomized Branch Expanding Prim*

Entrada: Grafo $G = (V_G, E_G)$
Saída: Arvore Geradora $T = (V_T, E_T)$

```

1   $(T, B_T, Pontas) \leftarrow \text{Pré-Processamento}(G)$ 
2  enquanto  $|E_T| < |V_G| - 1$  faça
3      se  $Pontas \neq \emptyset$  então
4          Seja  $d_1$  o maior grau  $d_G$  dentre os vértices contidos em  $Pontas$ 
5           $R_1 \leftarrow \text{InicializarRoleta}()$ 
6          para  $a \in Pontas$  faça
7              AdicionarRoleta( $R_1, a, d_1 - d_G(a) + 1$ )
8           $v \leftarrow \text{SortearRoleta}(R_1)$ 
9           $Pontas \leftarrow Pontas \setminus \{v\}$ 
10         Seja  $N_v = \{u \in \text{adj}_G(v) \mid u \notin V_T \vee$ 
             $(CConexa(u, T) \neq CConexa(v, T) \wedge (u \in B_T \vee d_T(u) = 1))\}$ 
11         se  $N_v \neq \emptyset$  então
12             Seja  $d_2$  o maior grau  $d_G$  dentre os vértice contidos em  $N_v$ 
13              $R_2 \leftarrow \text{InicializarRoleta}()$ 
14             para  $b \in N_v$  faça
15                 se  $b \notin V_T$  então  $fator \leftarrow 2$ 
16                 senão  $fator \leftarrow 1$ 
17                 AdicionarRoleta( $R_2, b, d_2 - d_G(b) + fator$ )
18              $u \leftarrow \text{SortearRoleta}(R_2)$ 
19             se  $u \notin V_T$  então  $V_T \leftarrow V_T \cup \{u\}$ 
20              $E_T \leftarrow E_T \cup (v, u)$ 
21             se  $d_T(u) = 1$  então  $Pontas \leftarrow Pontas \cup \{u\}$ 
22             senão
23                 se  $u \in Pontas$  então  $Pontas \leftarrow Pontas \setminus \{u\}$ 
24         senão
25             Seja  $v \in V_T$  um vértice que maximiza  $C(v) = |\{u \in \text{adj}_G(v) \mid u \notin V_T \vee$ 
             $(CConexa(u, T) \neq CConexa(v, T) \wedge (u \in B_T \vee d_T(u) = 1))\}|$ 
26             se  $C(v) > 0$  então
27                  $B_T \leftarrow B_T \cup \{v\}$ 
28                 ExpandirRamificação( $G, T, v$ )
29                 para  $u \in \text{adj}_G(v)$  faça
30                     se  $d_T(u) = 1 \wedge d_G(u) > 1$  então
31                         Pontas  $\leftarrow Pontas \cup \{u\}$ 
32             senão
33                 Seja  $v \in V_T$  um vértice que maximiza
                     $CC(v) = |\{u \in \text{adj}_G(v) \mid CConexa(u, T) \neq CConexa(v, T)\}|$ 
34                  $B_T \leftarrow B_T \cup \{v\}$ 
35  retorne  $(T)$ 

```

Algoritmo 12: R-CEP - *Randomized Controlled Expanding Prim*

Entrada: Grafo $G = (V_G, E_G)$
Saída: Arvore Geradora $T = (V_T, E_T)$

```

1   $(T, B_T, Pontas) \leftarrow \text{Pré-Processamento}(G)$ 
2  enquanto  $|E_T| < |V_G| - 1$  faça
3      se  $Pontas \neq \emptyset$  então
4          Seja  $d_1$  o maior grau  $d_G$  dentre os vértices contidos em  $Pontas$ 
5           $R_1 \leftarrow \text{InicializarRoleta}()$ 
6          para  $a \in Pontas$  faça
7               $\text{AdicionarRoleta}(R_1, a, d_1 - d_G(a) + 1)$ 
8           $v \leftarrow \text{SortearRoleta}(R_1)$ 
9          Seja  $N_v = \{u \in \text{adj}_G(v) \mid u \notin V_T \vee$ 
            $(CConexa(u, T) \neq CConexa(v, T) \wedge (u \in B_T \vee d_T(u) = 1))\}$ 
10         se  $N_v \neq \emptyset$  então
11             Seja  $d_2$  o maior grau  $d_G$  dentre os vértice contidos em  $N_v$ 
12              $R_2 \leftarrow \text{InicializarRoleta}()$ 
13             para  $b \in N_v$  faça
14                 se  $b \notin V_T$  então  $fator \leftarrow 2$ 
15                 senão  $fator \leftarrow 1$ 
16                  $\text{AdicionarRoleta}(R_2, b, d_2 - d_G(b) + fator)$ 
17              $u \leftarrow \text{SortearRoleta}(R_2)$ 
18             se  $u \notin V_T$  então  $V_T \leftarrow V_T \cup \{u\}$ 
19              $E_T \leftarrow E_T \cup (v, u)$ 
20             se  $d_T(u) = 1$  então  $Pontas \leftarrow Pontas \cup \{u\}$ 
21             senão
22                 se  $u \in Pontas$  então  $Pontas \leftarrow Pontas \setminus \{u\}$ 
23         se  $N_v = \emptyset \vee v \notin B_T$  então  $Pontas \leftarrow Pontas \setminus \{v\}$ 
24     senão
25         Seja  $v \in V_T$  um vértice que maximiza  $C(v) = |\{u \in \text{adj}_G(v) \mid u \notin V_T \vee$ 
            $(CConexa(u, T) \neq CConexa(v, T) \wedge (u \in B_T \vee d_T(u) = 1))\}|$ 
26         se  $C(v) > 0$  então
27              $B_T \leftarrow B_T \cup \{v\}$ 
28              $Pontas \leftarrow Pontas \cup \{v\}$ 
29         senão
30             Seja  $v \in V_T$  um vértice que maximiza
            $CC(v) = |\{u \in \text{adj}_G(v) \mid CConexa(u, T) \neq CConexa(v, T)\}|$ 
31              $B_T \leftarrow B_T \cup \{v\}$ 
32
33 retorne  $(T)$ 

```

conjunto de arestas candidatas $ACandidatas$. A probabilidade de cada aresta $(a, b) \in ACandidatas$ é definida como $\text{prob}((a, b)) = \max\{d_G(c) + d_G(d), \forall (c, d) \in ACandidatas\} - (d_G(a) + d_G(b) + fator)$, onde $fator = 2$, se $b \notin V_T$, e $fator = 1$ em caso contrário. Após

Algoritmo 13: R-EEP - *Randomized Edge Expanding Prim***Entrada:** Grafo $G = (V_G, E_G)$ **Saída:** Arvore Geradora $T = (V_T, E_T)$

```

1   $(T, B_T, Pontas) \leftarrow \text{Pré-Processamento}(G)$ 
2  enquanto  $|E_T| < |V_G| - 1$  faça
3      se  $Pontas \neq \emptyset$  então
4          Seja  $ACandidatas$  o conjunto de arestas  $(a,b) \in E_G$  que satisfazem
               $a \in Pontas$  e  $b \in adj_G(a) \mid b \notin V_T \vee (CConexa(b,T) \neq CConexa(a,T) \wedge$ 
               $(b \in B_T \vee d_T(b) = 1))$ 
5          se  $ACandidatas \neq \emptyset$  então
6              Seja  $(c,d)$  uma aresta de  $ACandidatas$  que maximiza  $d_G(c) + d_G(d)$ 
7              Seja  $d$  igual a  $d_G(c) + d_G(d)$ 
8               $R \leftarrow \text{InicializarRoleta}()$ 
9              para  $(a,b) \in ACandidatas$  faça
10                 se  $b \notin V_T$  então  $fator \leftarrow 2$ 
11                 senão  $fator \leftarrow 1$ 
12                  $\text{AdicionarRoleta}(R, (a,b), d - (d_G(a) + d_G(b)) + fator)$ 
13              $(v,u) \leftarrow \text{SortearRoleta}(R)$ 
14              $Pontas \leftarrow Pontas \setminus \{v\}$ 
15             se  $u \notin V_T$  então  $V_T \leftarrow V_T \cup \{u\}$ 
16              $E_T \leftarrow E_T \cup (v,u)$ 
17             se  $d_T(u) = 1$  então  $Pontas \leftarrow Pontas \cup \{u\}$ 
18             senão
19                 se  $u \in Pontas$  então  $Pontas \leftarrow Pontas \setminus \{u\}$ 
20             senão  $Pontas \leftarrow \emptyset$ 
21         senão
22             Seja  $v \in V_T$  um vértice que maximiza  $C(v) = |\{u \in adj_G(v) \mid u \notin V_T \vee$ 
               $(CConexa(u,T) \neq CConexa(v,T) \wedge (u \in B_T \vee d_T(u) = 1))\}|$ 
23             se  $C(v) > 0$  então
24                  $B_T \leftarrow B_T \cup \{v\}$ 
25                  $\text{ExpandirRamificação}(G, T, v)$ 
26                 para  $u \in adj_G(v)$  faça
27                     se  $d_T(u) = 1 \wedge d_G(u) > 1$  então
28                          $Pontas \leftarrow Pontas \cup \{u\}$ 
29                 senão
30                     Seja  $v \in V_T$  um vértice que maximiza
                         $CC(v) = |\{u \in adj_G(v) \mid CConexa(u,T) \neq CConexa(v,T)\}|$ 
31                      $B_T \leftarrow B_T \cup \{v\}$ 
32 retorne  $(T)$ 

```

construir a roleta, os algoritmos selecionam uma aresta (v,u) e continuam exatamente os mesmos procedimentos adotados no EEP e no CEEP.

Algoritmo 14: R-CEEP - *Randomized Controlled Edge Expanding Prim***Entrada:** Grafo $G = (V_G, E_G)$ **Saída:** Arvore Geradora $T = (V_T, E_T)$

```

1   $(T, B_T, Pontas) \leftarrow \text{Pré-Processamento}(G)$ 
2  enquanto  $|E_T| < |V_G| - 1$  faça
3      se  $Pontas \neq \emptyset$  então
4          Seja  $ACandidatas$  o conjunto de arestas  $(a,b) \in E_G$  que satisfazem
               $a \in Pontas$  e  $b \in adj_G(a) \mid b \notin V_T \vee (CConexa(b,T) \neq CConexa(a,T) \wedge$ 
               $(b \in B_T \vee d_T(b) = 1))$ 
5          se  $ACandidatas \neq \emptyset$  então
6              Seja  $(c,d)$  uma aresta de  $ACandidatas$  que maximiza  $d_G(c) + d_G(d)$ 
7              Seja  $d$  igual a  $d_G(c) + d_G(d)$ 
8               $R \leftarrow \text{InicializarRoleta}()$ 
9              para  $(a,b) \in ACandidatas$  faça
10                 se  $b \notin V_T$  então  $fator \leftarrow 2$ 
11                 senão  $fator \leftarrow 1$ 
12                  $\text{AdicionarRoleta}(R, (a,b), d - (d_G(a) + d_G(b)) + fator)$ 
13              $(v,u) \leftarrow \text{SortearRoleta}(R)$ 
14             se  $v \notin B_T$  então  $Pontas \leftarrow Pontas \setminus \{v\}$ 
15             se  $u \notin V_T$  então  $V_T \leftarrow V_T \cup \{u\}$ 
16              $E_T \leftarrow E_T \cup (v,u)$ 
17             se  $d_T(u) = 1$  então  $Pontas \leftarrow Pontas \cup \{u\}$ 
18             senão
19                 se  $u \in Pontas$  então  $Pontas \leftarrow Pontas \setminus \{u\}$ 
20             senão  $Pontas \leftarrow \emptyset$ 
21         senão
22             Seja  $v \in V_T$  um vértice que maximiza  $C(v) = |\{u \in adj_G(v) \mid u \notin V_T \vee$ 
               $(CConexa(u,T) \neq CConexa(v,T) \wedge (u \in B_T \vee d_T(u) = 1))\}|$ 
23             se  $C(v) > 0$  então
24                  $B_T \leftarrow B_T \cup \{v\}$ 
25                  $Pontas \leftarrow Pontas \cup \{v\}$ 
26             senão
27                 Seja  $v \in V_T$  um vértice que maximiza
                     $CC(v) = |\{u \in adj_G(v) \mid CConexa(u,T) \neq CConexa(v,T)\}|$ 
28                  $B_T \leftarrow B_T \cup \{v\}$ 
29 retorne  $(T)$ 

```

Os casos em que $Pontas = \emptyset$ ou $ACandidatas = \emptyset$ também seguem idênticos aos apresentados anteriormente para o Algoritmo EEP. Assim como no R-BEP, as modificações implementadas não alteram a corretude do método, pois a única mudança é a seleção de qual aresta será adicionada dentre o conjunto restrito das que atendem às condições inerentes ao problema. Os Algoritmos R-EEP e R-CEEP foram descritos como os

Algoritmos 13 e 14 respectivamente.

Uma vez que os oito métodos foram apresentados, podemos prosseguir para os experimentos computacionais.

5 Experimentos Computacionais

Os experimentos realizados consistem na execução dos oito algoritmos nas 525 instâncias propostas por (CARRABS et al., 2013). Essas instâncias compreendem grafos conexos, esparsos, variando entre 20 e 1000 vértices.

Os experimentos foram executados em um Intel(R) Core(TM) i5-3230M @ 2,60 Ghz, com 3 Mb de cache e 8 Gb de RAM usando o sistema operacional Windows 7. Todas as heurísticas foram programadas na linguagem C++ e compiladas usando o compilador gcc. Para cada uma das instâncias, as heurísticas gulosas foram executadas uma vez, pois apresentam natureza determinística. Para avaliar as heurísticas randômicas, executamos cada instância 100 vezes e identificamos a melhor solução encontrada, assim como a média das soluções geradas. Optamos por não relacionar os tempos de execução neste trabalho, pois os valores encontrados são desprezíveis: o maior tempo de execução foi sempre inferior a 0,01 segundos, mesmo considerando instâncias com 1000 vértices.

Os valores obtidos pelas heurísticas propostas neste trabalho foram comparados com a heurística construtiva gulosa apresentada por (MORENO; FROTA; MARTINS, 2018). A heurística construtiva de (MARÍN, 2015) também apresenta bons resultados, mas como utiliza busca local, preferimos não compará-la com os métodos aqui propostos.

Cabe ressaltar que o trabalho de (MORENO; FROTA; MARTINS, 2018) não avalia os resultados obtidos apenas pela heurística construtiva gulosa, pois esse algoritmo é utilizado como uma fonte de soluções iniciais, que são então aprimoradas através de métodos de busca local em uma meta-heurística ILS. Desta forma, para possibilitar uma comparação entre as heurísticas construtivas, reproduzimos os testes com o algoritmo ILS como proposto por (MORENO; FROTA; MARTINS, 2018), mas interrompendo o algoritmo imediatamente após o algoritmo construtivo guloso. Esse experimento permitiu desacoplar os dois componentes do método de (MORENO; FROTA; MARTINS, 2018) e identificar os resultados de seu método construtivo isoladamente.

5.1 Resultados para as Instâncias de Carrabs et al. (2013)

Dividimos as instâncias propostas por (CARRABS et al., 2013) em dois grupos: instâncias médias e instâncias grandes. O primeiro grupo contém grafos com 20 a 500 vértices, enquanto o segundo grupo, 600 a 1000 vértices. Para cada valor de n , existem cinco conjuntos de grafos com um mesmo número de arestas m . Cada grupo formado por um par (n, m) contém cinco grafos distintos entre si com n vértices e m arestas. Os resultados para as instâncias médias foram agrupados em função do número de vértices. Desta forma, cada resultado apontado agrupa um conjunto de 25 grafos. Já para instâncias grandes, visando um maior detalhamento, optamos por agrupar apenas as instâncias com o mesmo número de vértices e arestas. Logo, cada resultado apontado nas tabelas considera um conjunto com apenas 5 grafos.

Na Subseção 5.1.1, é efetuado uma comparação entre os resultados obtidos pelas heurísticas gulosas e randomizadas propostas neste trabalho e os resultados alcançados pela heurística construtiva apresentada por (MORENO; FROTA; MARTINS, 2018). Essa análise leva em consideração a melhor solução obtida a partir das 100 execuções dos algoritmos randomizados em cada uma das instâncias de (CARRABS et al., 2013).

Já na Subseção 5.1.2, é realizado uma comparação dos resultados das heurísticas randomizadas propostas neste trabalho com a melhor heurística gulosa desenvolvida nesta pesquisa e também com os resultados obtidos pela heurística construtiva de (MORENO; FROTA; MARTINS, 2018). Nesta análise, é levado em consideração a média dos resultados obtidos nas 100 execuções dos algoritmos randomizados em cada uma das instâncias de (CARRABS et al., 2013).

5.1.1 Estudo dos Resultados Mínimos

Nas Tabelas 1 e 3, as duas primeiras colunas correspondem às quantidades médias de vértices (coluna **n'**) e de arestas (coluna **m'**) de cada conjunto. A coluna **Ótimo** representa a média dos valores ótimos para as instâncias analisadas. A coluna **Moreno et al.** representa a média dos resultados obtidos pela heurística construtiva de (MORENO; FROTA; MARTINS, 2018). Da quinta à oitava coluna, respectivamente **BEP**, **EEP**, **CEP** e **CEEP**, são apresentados os resultados médios obtidos pelos algoritmos gulosos propostos neste trabalho. Da nona à décima segunda coluna são representadas as médias considerando a melhor solução obtidas pelas 100 execuções dos algoritmos R-BEP, R-EEP, R-CEP e R-CEEP para cada uma das instâncias.

A Tabela 1 apresenta os resultados para as instâncias médias, enquanto a Tabela 3 demonstra os resultados para as instâncias grandes. Para melhor identificação, foram marcados em fonte *bold* os melhores resultados obtidos em cada grupo. Nas colunas **BEP**, **EEP**, **CEP** e **CEEP**, foram coloridos em azul os valores que são iguais ou superiores aos obtidos pelo método construtivo de (MORENO; FROTA; MARTINS, 2018), para melhor comparação dos algoritmos determinísticos.

n'	m'	Ótimo	Moreno et al	BEP	EEP	CEP	CEEP	R-BEP	R-EEP	R-CEP	R-CEEP
20	41,8	0,8	1,04	1,08	1,04	1,08	1,04	0,84	0,84	0,84	0,84
40	70,8	2,8	3,76	3,84	3,68	3,80	3,72	3,28	3,48	3,36	3,60
60	95,0	6,3	8,16	7,72	7,80	7,68	7,72	7,12	7,32	7,08	7,28
80	119,8	9,2	11,68	10,76	10,56	10,80	10,56	10,24	10,40	10,20	10,40
100	144,0	13,3	16,24	15,24	14,84	15,24	14,84	14,76	14,84	14,76	15,04
120	168,8	17,5	20,88	19,92	19,04	20,00	19,04	19,00	19,28	18,92	19,44
140	193,0	20,9	24,52	23,68	22,96	23,64	22,96	22,68	23,08	22,84	23,04
160	217,8	25,0	29,84	28,28	27,60	28,32	27,60	27,24	27,40	27,20	27,44
180	242,0	29,1	33,44	32,36	31,28	32,36	31,32	31,28	31,76	31,32	31,68
200	266,8	32,6	37,56	36,00	35,20	35,96	35,20	35,20	35,40	35,24	35,64
250	321,0	44,6	50,72	48,48	47,76	48,48	47,76	47,76	48,00	47,48	48,04
300	380,0	57,4	63,16	61,60	60,84	61,60	60,84	60,40	60,92	60,48	60,80
350	434,8	68,6	76,12	73,44	72,68	73,48	72,72	72,36	72,40	72,12	72,44
400	489,0	81,8	90,84	87,20	86,52	87,20	86,52	85,88	86,36	86,12	86,48
450	548,0	93,4	102,04	98,80	97,84	98,84	97,88	97,68	98,12	97,56	98,00
500	602,8	106,7	116,64	112,68	111,28	112,68	111,28	111,00	111,80	111,36	111,56

Tabela 1: Resultados Mínimos para *Medium Instances*.

n'	m'	Moreno et al	BEP	EEP	CEP	CEEP	R-BEP	R-EEP	R-CEP	R-CEEP
20	41,8	30,0	35,0	30,0	35,0	30,0	5,0	5,0	5,0	5,0
40	70,8	34,3	37,1	31,4	35,7	32,9	17,1	24,3	20,0	28,6
60	95,0	29,5	22,5	23,8	21,9	22,5	13,0	16,2	12,4	15,6
80	119,8	27,0	17,0	14,8	17,4	14,8	11,3	13,0	10,9	13,0
100	144,0	22,1	14,6	11,6	14,6	11,6	11,0	11,6	11,0	13,1
120	168,8	19,3	13,8	8,8	14,3	8,8	8,6	10,2	8,1	11,1
140	193,0	17,3	13,3	9,9	13,1	9,9	8,5	10,4	9,3	10,2
160	217,8	19,4	13,1	10,4	13,3	10,4	9,0	9,6	8,8	9,8
180	242,0	14,9	11,2	7,5	11,2	7,6	7,5	9,1	7,6	8,9
200	266,8	15,2	10,4	8,0	10,3	8,0	8,0	8,6	8,1	9,3
250	321,0	13,7	8,7	7,1	8,7	7,1	7,1	7,6	6,5	7,7
300	380,0	10,0	7,3	6,0	7,3	6,0	5,2	6,1	5,4	5,9
350	434,8	11,0	7,1	5,9	7,1	6,0	5,5	5,5	5,1	5,6
400	489,0	11,1	6,6	5,8	6,6	5,8	5,0	5,6	5,3	5,7
450	548,0	9,3	5,8	4,8	5,8	4,8	4,6	5,1	4,5	4,9
500	602,8	9,3	5,6	4,3	5,6	4,3	4,0	4,8	4,4	4,6

Tabela 2: Gap dos Resultados Mínimos para *Medium Instances*.

As Tabelas 2 e 4 apresentam os *gaps* dos algoritmos, sendo eles o construtivo de (MORENO; FROTA; MARTINS, 2018) (terceira coluna) e os construtivos propostos neste trabalho (da quarta à décima primeira coluna), considerando a diferença percentual entre cada algoritmo e a solução ótima para cada instância (com base nos resultados apresentados nas Tabelas 1 e 3), sendo calculado da seguinte maneira: $gap = \frac{Algoritmo - \acute{O}timo}{\acute{O}timo} \times 100$.

A primeira (**n'**) e a segunda (**m'**) coluna correspondem, respectivamente, às duas primeiras colunas das Tabelas 1 e 3.

n'	m'	Ótimo	Moreno et al	BEP	EEP	CEP	CEEP	R-BEP	R-EEP	R-CEP	R-CEEP
600	637	183,8	187,6	185,4	185,6	185,4	185,6	184,6	184,6	184,6	184,6
600	674	167,2	173,4	171,4	171,0	171,4	171,0	170,2	170,0	170,0	170,0
600	712	150,6	159,4	156,2	156,6	156,2	156,6	154,0	154,2	154,4	154,4
600	749	138,8	145,4	145,6	144,4	145,6	144,4	142,6	143,2	143,0	143,6
600	787	125,8	135,4	133,8	133,8	133,8	133,8	131,6	132,0	131,4	131,4
700	740	214,4	218,8	217,0	217,2	217,0	217,2	216,2	216,2	216,2	216,2
700	780	198,0	204,4	202,6	202,0	202,6	202,0	201,0	201,0	201,0	201,2
700	821	180,0	189,0	185,6	185,8	185,6	185,8	184,0	184,0	184,0	183,8
700	861	164,0	174,2	171,4	170,8	171,4	170,8	168,8	168,6	168,8	168,6
700	902	154,2	165,2	161,8	162,2	161,8	162,2	159,0	159,6	159,0	159,4
800	843	245,6	250,8	248,4	248,8	248,4	248,8	247,8	247,8	247,8	247,8
800	886	227,6	235,0	232,0	231,8	232,0	231,8	230,6	230,6	230,8	230,6
800	930	208,4	216,2	215,2	215,2	215,2	215,2	212,6	213,2	213,2	212,8
800	973	194,2	205,0	201,6	200,6	201,6	200,6	198,8	199,0	199,0	199,2
800	1017	176,2	189,0	184,8	184,4	184,8	184,4	182,6	182,8	182,8	182,8
900	944	279,6	285,2	282,8	282,6	282,8	282,6	282,0	282,0	282,0	282,0
900	989	259,2	268,8	264,8	264,4	264,8	264,4	263,4	263,4	263,4	263,2
900	1034	240,6	250,6	247,0	247,4	247,0	247,4	245,0	244,6	245,0	244,6
900	1079	223,2	235,8	232,4	231,0	232,4	231,0	229,6	229,4	229,8	229,8
900	1124	206,0	219,0	214,8	214,2	214,8	214,2	213,0	213,2	212,6	213,0
1000	1047	312,0	317,4	315,4	314,6	315,4	314,6	314,6	314,6	314,6	314,6
1000	1095	290,0	299,6	295,6	296,0	295,6	296,0	294,4	294,2	294,0	293,8
1000	1143	271,2	283,0	277,8	278,4	277,8	278,4	276,2	276,0	276,2	276,2
1000	1191	251,0	264,0	259,4	259,4	259,4	259,4	257,2	257,2	257,6	257,2
1000	1239	235,2	249,6	244,0	244,4	244,0	244,4	242,2	242,2	241,6	242,0

Tabela 3: Resultados Mínimos para *Large Instances*.

Em uma comparação restrita aos algoritmos determinísticos (o método construtivo de (MORENO; FROTA; MARTINS, 2018)), o BEP, o EEP, o CEP e o CEEP – colunas 4, 5, 6, 7 e 8 das Tabelas 1 e 3), nota-se que o BEP e o CEP foram capazes de superar o construtivo de (MORENO; FROTA; MARTINS, 2018) em quase todos os grupos de instâncias, exceto em três. Já o EEP e o CEEP foram capazes de apontar resultados melhores ou equivalentes em todos os grupos de instâncias estudadas (empatando em apenas um caso cada).

Analisando agora os resultados das Tabelas 1 e 3 de forma global, nota-se que o Algoritmo R-BEP, em comparação aos demais métodos, apresentou o melhor resultado em 23 dos 41 grupos de instâncias. Esse resultado demonstra que o R-BEP foi superior aos demais métodos, incluindo o algoritmo construtivo da melhor meta-heurística conhecida na literatura (MORENO; FROTA; MARTINS, 2018). Em segundo lugar fica o R-CEP, atingindo o melhor resultado em 20 grupos de instâncias. Em terceiro e quarto lugares ficam o R-CEEP e o R-EEP, atingindo o melhor resultado em 15 e 14 grupos de instâncias respectivamente.

n'	m'	Moreno et al	BEP	EEP	CEP	CEEP	R-BEP	R-EEP	R-CEP	R-CEEP
600	637	2,1	0,9	1,0	0,9	1,0	0,4	0,4	0,4	0,4
600	674	3,7	2,5	2,3	2,5	2,3	1,8	1,7	1,7	1,7
600	712	5,8	3,7	4,0	3,7	4,0	2,3	2,4	2,5	2,5
600	749	4,8	4,9	4,0	4,9	4,0	2,7	3,2	3,0	3,5
600	787	7,6	6,4	6,4	6,4	6,4	4,6	4,9	4,5	4,5
700	740	2,1	1,2	1,3	1,2	1,3	0,8	0,8	0,8	0,8
700	780	3,2	2,3	2,0	2,3	2,0	1,5	1,5	1,5	1,6
700	821	5,0	3,1	3,2	3,1	3,2	2,2	2,2	2,2	2,1
700	861	6,2	4,5	4,1	4,5	4,1	2,9	2,8	2,9	2,8
700	902	7,1	4,9	5,2	4,9	5,2	3,1	3,5	3,1	3,4
800	843	2,1	1,1	1,3	1,1	1,3	0,9	0,9	0,9	0,9
800	886	3,3	1,9	1,8	1,9	1,8	1,3	1,3	1,4	1,3
800	930	3,7	3,3	3,3	3,3	3,3	2,0	2,3	2,3	2,1
800	973	5,6	3,8	3,3	3,8	3,3	2,4	2,5	2,5	2,6
800	1017	7,3	4,9	4,7	4,9	4,7	3,6	3,7	3,7	3,7
900	944	2,0	1,1	1,1	1,1	1,1	0,9	0,9	0,9	0,9
900	989	3,7	2,2	2,0	2,2	2,0	1,6	1,6	1,6	1,5
900	1034	4,2	2,7	2,8	2,7	2,8	1,8	1,7	1,8	1,7
900	1079	5,6	4,1	3,5	4,1	3,5	2,9	2,8	3,0	3,0
900	1124	6,3	4,3	4,0	4,3	4,0	3,4	3,5	3,2	3,4
1000	1047	1,7	1,1	0,8	1,1	0,8	0,8	0,8	0,8	0,8
1000	1095	3,3	1,9	2,1	1,9	2,1	1,5	1,4	1,4	1,3
1000	1143	4,4	2,4	2,7	2,4	2,7	1,8	1,8	1,8	1,8
1000	1191	5,2	3,3	3,3	3,3	3,3	2,5	2,5	2,6	2,5
1000	1239	6,1	3,7	3,9	3,7	3,9	3,0	3,0	2,7	2,9

Tabela 4: Gap dos Resultados Mínimos para *Large Instances*.

5.1.2 Estudo dos Resultados Médios

Nas Tabelas 5 e 6, as duas primeiras colunas correspondem às quantidades médias de vértices (coluna **n'**) e de arestas (coluna **m'**) de cada conjunto. A coluna **Ótimo** representa a média dos valores ótimos para as instâncias analisadas. A coluna **Moreno et al.** representa a média dos resultados obtidos pela heurística construtiva de (MORENO; FROTA; MARTINS, 2018). A coluna **EEP** representa a média dos valores obtidos pelo melhor algoritmo guloso proposto neste trabalho, que apresentou o melhor resultado em 30 dos 41 grupos de instâncias em comparação com as demais heurísticas gulosas. Da sexta à nona coluna são representadas as médias considerando o resultado médio obtido pelas 100 execuções dos algoritmos R-BEP, R-EEP, R-CEP e R-CEEP para cada uma das instâncias.

A Tabela 5 apresenta os resultados para as instâncias médias, enquanto a Tabela 6 demonstra os resultados para as instâncias grandes. Para melhor identificação, foram marcados em fonte *bold* os melhores resultados obtidos em cada grupo. Nas colunas **R-BEP**, **R-EEP**, **R-CEP** e **R-CEEP**, foram coloridos em azul os valores que são iguais ou superiores aos obtidos pelo método de (MORENO; FROTA; MARTINS, 2018).

n	m	Ótimo	Moreno et al	EEP	RBEP	REEP	RCEP	RCEEP
20	41,8	0,8	1,04	1,04	1,50	1,58	1,55	1,63
40	70,8	2,8	3,76	3,68	4,71	4,82	4,77	4,87
60	95,0	6,3	8,16	7,80	8,70	8,90	8,75	8,90
80	119,8	9,2	11,68	10,56	12,01	12,28	12,06	12,28
100	144,0	13,3	16,24	14,84	16,59	16,94	16,64	16,92
120	168,8	17,5	20,88	19,04	20,98	21,29	20,98	21,35
140	193,0	20,9	24,52	22,96	24,92	25,28	24,95	25,35
160	217,8	25,0	29,84	27,60	29,36	29,83	29,43	29,79
180	242,0	29,1	33,44	31,28	33,79	34,24	33,79	34,24
200	266,8	32,6	37,56	35,20	37,64	38,18	37,68	38,22
250	321,0	44,6	50,72	47,76	50,17	50,76	50,23	50,71
300	380,0	57,4	63,16	60,84	63,02	63,49	63,01	63,47
350	434,8	68,6	76,12	72,68	75,02	75,56	75,01	75,59
400	489,0	81,8	90,84	86,52	88,76	89,22	88,74	89,28
450	548,0	93,4	102,04	97,84	100,52	101,09	100,51	101,15
500	602,8	106,7	116,64	111,28	114,05	114,59	114,04	114,53

Tabela 5: Resultados Médios para *Medium Instances*

n	m	Ótimo	Moreno et al	EEP	RBEP	REEP	RCEP	RCEEP
600	637	183,8	187,6	185,6	186,1	185,9	186,0	185,8
600	674	167,2	173,4	171,0	171,2	171,2	171,2	171,2
600	712	150,6	159,4	156,6	156,3	156,6	156,4	156,5
600	749	138,8	145,4	144,4	145,2	145,3	145,2	145,4
600	787	125,8	135,4	133,8	134,0	134,5	134,0	134,4
700	740	214,4	218,8	217,2	217,3	217,3	217,3	217,2
700	780	198,0	204,4	202,0	202,5	202,6	202,5	202,6
700	821	180,0	189,0	185,8	185,8	185,8	185,8	185,8
700	861	164,0	174,2	170,8	171,2	171,2	171,2	171,3
700	902	154,2	165,2	162,2	162,3	162,7	162,2	162,5
800	843	245,6	250,8	248,8	248,5	248,6	248,4	248,5
800	886	227,6	235,0	231,8	231,9	231,9	232,0	231,9
800	930	208,4	216,2	215,2	214,9	214,9	214,9	214,9
800	973	194,2	205,0	200,6	201,4	201,5	201,5	201,6
800	1017	176,2	189,0	184,4	185,4	185,7	185,6	185,7
900	944	279,6	285,2	282,6	282,7	282,7	282,8	282,7
900	989	259,2	268,8	264,4	265,0	264,9	265,0	265,0
900	1034	240,6	250,6	247,4	247,0	247,0	247,0	247,0
900	1079	223,2	235,8	231,0	232,0	232,0	232,0	232,0
900	1124	206,0	219,0	214,2	215,3	215,9	215,3	215,8
1000	1047	312,0	317,4	314,6	315,9	315,8	315,9	315,7
1000	1095	290,0	299,6	296,0	295,9	295,7	295,9	295,8
1000	1143	271,2	283,0	278,4	278,0	277,9	277,9	278,0
1000	1191	251,0	264,0	259,4	260,0	259,9	260,0	259,8
1000	1239	235,2	249,6	244,4	244,7	245,0	244,8	245,0

Tabela 6: Resultados Médios para *Large Instances*

Ao comparar os valores adquiridos pelo método construtivo de (MORENO; FROTA; MARTINS, 2018) (coluna 4 das Tabelas 5 e 6) com as médias obtidas pelos algoritmos R-BEP, R-EEP, R-CEP e R-CEEP - colunas 6, 7, 8 e 9 das Tabelas 5 e 6, percebe-se que o R-BEP e o R-CEP foram capazes de superar o construtivo de (MORENO; FROTA; MARTINS, 2018) em 32 dos 41 grupos de instâncias. Já o R-EEP e o R-CEEP superaram 30 dos 41 grupos de instâncias. Isso mostra a superioridade dos algoritmos randomizados deste trabalho em comparação ao método de (MORENO; FROTA; MARTINS, 2018), mesmo nos casos médios.

Examinando agora os resultados dessas tabelas (5 e 6) de forma global, percebe-se que o Algoritmo guloso EEP, em comparação aos demais métodos, apresentou o melhor resultado em 34 dos 41 grupos de instâncias. Isso revela a eficiência dos algoritmos determinísticos apresentados nesse trabalho, mesmo frente aos randomizados.

6 Conclusão

Problemas NP-difíceis continuam sendo um desafio significativo na pesquisa, e soluções simples para problemas como o MBV provavelmente serão obtidas usando métodos não exatos, como heurísticas e algoritmos aproximativos. Este estudo abordou o MBV, um problema que envolve encontrar uma árvore geradora com o menor número possível de vértices de grau maior ou igual a 3. Foi visto neste trabalho que o MBV amplia o clássico problema de encontrar um caminho hamiltoniano em um grafo, assim como sua origem e motivação relacionada as redes ópticas.

Neste trabalho foi explorado os limites de heurísticas construtivas para o problema MBV. Foram desenvolvidas oito heurísticas construtivas para abordar esse problema. As quatro primeiras, BEP, EEP, CEP e CEEP, são algoritmos baseados em estratégias gulosas. As outras quatro, R-BEP, R-EEP, R-CEP e R-CEEP, são variantes das primeiras e incorporam critérios randomizados. As heurísticas propostas exploram as pontes e articulações existentes nos grafos para construir as soluções.

Os experimentos realizados mostraram que tanto os algoritmos gulosos quanto os algoritmos randomizados propostos neste trabalho obtiveram melhores resultados em comparação à heurística construtiva apresentada em (MORENO; FROTA; MARTINS, 2018).

Com base nos valores mínimos obtidos pelos algoritmos aqui propostos, os Algoritmos gulosos BEP e CEP geraram resultados superiores ao algoritmo construtivo de (MORENO; FROTA; MARTINS, 2018) em 92,7% dos conjuntos de instâncias. Já os Algoritmos EEP e CEEP apresentaram resultados melhores em 97,6%, empatando nos 2,4% restantes (um único caso cada). Já os quatro algoritmos randomizados apontaram resultados melhores ou iguais do que os demais em 100% dos grupos de instâncias.

Todos os algoritmos propostos neste trabalho se mostraram bastantes rápidos e eficientes, atingindo soluções muito próximas aos valores ótimos (gap inferior a 6,5% nas instâncias grandes).

Uma proposta para trabalhos futuros é a continuação desse estudo, avaliando variações

do procedimento de Expansão de Ramificações, apresentado na Seção 4.2. Outra proposta é a integração dos algoritmos construtivos já desenvolvidos a algoritmos baseados em meta-heurísticas, como o ILS ou GRASP, visando aprimorar ainda mais os bons resultados encontrados.

Também seria proveitoso trabalhar na questão do MBV em conjunto com o problema que busca minimizar o grau das ramificações, conhecido como “*Problem of Minimizing the Degree Sum of Branch Vertices*” ou DMS (CERULLI; GENTILI; IOSSA, 2009).

Outra questão interessante é adaptar as ideias propostas neste trabalho para o Problema d -MBV (MORENO; FROTA; MARTINS, 2018), a versão do MBV que busca limitar o número de vértices com grau maior ou igual a uma constante d na árvore geradora construída.

REFERÊNCIAS

ALMEIDA, A. S. **Minimizando Ramificações em Árvore Geradoras**. Niterói, 2015.

ALMEIDA, A. S.; NOGUEIRA, L. T.; SÁ, V. G. P. Minimizando Ramificações em Árvore Geradoras. In: ANAIS do XLVI SBPO. Salvador: SOBRAPO, 2014. P. 2977–2985.

BOAVENTURA NETTO, P. O. **Grafos: Teoria, Modelos, Algoritmos**. São Paulo: Blucher, 2012.

BONDY, J.A; MURTY, U.S.R. **Graph Theory**. London: Springer, 2008.

CARRABS, F. et al. Lower and upper bounds for the spanning tree with minimum branch vertices. **Computational Optimization and Applications**, v. 56, n. 2, p. 405–438, 2013.

CERULLI, R.; GENTILI, M.; IOSSA, A. Bounded-degree spanning tree problems: models and new algorithms. **Computational Optimization and Applications**, v. 42, n. 3, p. 353–370, 2009.

CORMEN, T.H. et al. **Introduction to Algorithms**. Cambridge: 3rd ed. MIT Press, 2009.

DIESTEL, R. **Graph Theory**. Berlin, Heidelberg: 5rd. ed. Springer, 2017.

GAREY, M. R.; JOHNSON, D. S. **Computers and Intractability: A Guide to the Theory of NP-Completeness**. San Francisco: WH Freeman e Company, 1979.

GARGANO, L. et al. Spanning Trees with Bounded Number of Branch Vertices. In: AUTOMATA, Languages and Programming. [S. l.]: Springer Berlin, Heidelberg, 2002. P. 355–365.

HOPCROFT, J.; TARJAN, R. Algorithm 447: efficient algorithms for graph manipulation. **Communications of the ACM**, v. 16, p. 372–378, 1973.

KRUSKAL, J.B. On the Shortest Spanning Subtree of a Graph and the Traveling Salesman Problem. **Proceedings of the American Mathematical Society**, v. 7, p. 48–50, 1956.

- LANDETE, M.; MARÍN, A.; SAINZ-PARDO, J.L. Decomposition methods based on articulation vertices for degree-dependent spanning tree problems. **Computational Optimization and Applications**, v. 68, n. 3, p. 749–773, 2017.
- MARÍN, A. Exact and heuristic solutions for the Minimum Number of Branch Vertices Spanning Tree Problem. **European Journal of Operational Research**, v. 245, n. 3, p. 680–689, 2015.
- MELO, R.A.; SAMER, P.; URRUTIA, S. An effective decomposition approach and heuristics to generate spanning trees with a small number of branch vertices. **Computational Optimization and Applications**, v. 65, n. 3, p. 821–844, 2016.
- MERABET, M.; MOLNAR, M. **Generalization of the Minimum Branch Vertices Spanning Tree Problem**. [S. l.], 2016.
- MORENO, J.; FROTA, Y.; MARTINS, S. An exact and heuristic approach for the d-minimum branch vertices problem. **Computational Optimization and Applications**, v. 71, n. 3, p. 829–855, 2018.
- MORENO, J.; PLASTINO, A.; MARTINS, S. Heurística Iterated Local Search para Resolver o Problema Minimum Branch Vertices. In: ANAIS do XLVIII SBPO. Vitória, ES: SOBRAPO, 2016. P. 1896–1907.
- MUKHERJEE, B. **Optical Communication Networks**. New York: McGraw Hill, 1997.
- PRIM, R.C. Shortest connection networks and some generalizations. **The Bell System Technical Journal**, v. 36, p. 1389–1401, 1957.
- SAHASRABUDDHE, L.H.; MUKHERJEE, B. Light trees: optical multicasting for improved performance in wavelength routed networks. **IEEE Communications Magazine**, v. 37, n. 2, p. 67–73, 1999.
- SAHASRABUDDHE, L.H.; SINGHAL, N.; MUKHERJEE, B. Light-Trees for WDM Optical Networks: Optimization Problem Formulations for Unicast and Broadcast Traffic. **Proc.Int.Conf.on Communications, Computers, and Devices (ICCD-2000)**, IIT Kharagpur, India, 2000.
- SCHMIDT, J. M. A simple test on 2-vertex- and 2-edge-connectivity. **Information Processing Letters**, v. 113, p. 241–244, 2013.
- SILVA, R.M.A. et al. An edge-swap heuristic for generating spanning trees with minimum number of branch vertices. **Optimization Letters**, v. 8, n. 4, p. 1225–1243, 2014.

SILVESTRI, S.; LAPORTE, G.; CERULLI, R. A branch-and-cut algorithm for the minimum branch vertices spanning tree problem. **Computers and Operation Research**, v. 81, p. 322–332, 2017.

STERNE, T. E.; BALA, K. **MultiWaveLength Optical Network: A Layered Approach**. Boston: Addison-Wesley, 1999.

SZWARCFITER, J.L. **Teoria Computacional de Grafos**. Rio de janeiro: Elsevier, 2018.