São José dos Campos - SP
LV SBPO
LV Simpósio Brasileiro de Pesquisa Operacional
São José dos Campos, SP - 6 a 9 de novembro de 2023

Explorando os Limites de Heurísticas Construtivas para o Problema da Árvore Geradora com Número Mínimo de Ramificações*

Anderson Beltrão
Universidade Federal Fluminense
Instituto de Computação
anderson_beltrao@id.uff.br

Rodrigo Lamblet Mafort
Universidade do Estado do Rio de Janeiro
Instituto Politécnico
rodrigo.mafort@iprj.uerj.br

Fábio Protti
Universidade Federal Fluminense
Instituto de Computação
fabio@ic.uff.br

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 quatro novas heurísticas construtivas para o problema, inspiradas no Algoritmo de Prim: dois métodos gulosos e duas variantes randomizadas. Os resultados obtidos para as instâncias de referência se mostraram superiores aos anteriormente encontrados na literatura.

PALAVRAS CHAVE. Grafos, Árvore Geradora, Ramificações, Heurísticas.
Tópicos (Heurísticas e Algorítimos em Grafos)

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 four novel constructive heuristics for the problem, inspired by Prim's Algorithm: two greedy methods and two randomized variations. The experimental results obtained on benchmark instances show a significant improvement over the previously reported state-of-the-art solutions in the literature.

KEYWORDS. Graphs, Spanning Trees, Branch Vertices, Heuristics.
Paper topics (Heuristics and Algorithms in Graphs)

*A. Beltrão, R. L. Mafort e F. Protti foram parcialmente financiados pela CAPES (CAPESMPROEX29), pelo CNPQ (304117/2019-6 e 166688/2020-7) e pela FAPERJ (201.083/2021).

DOI: 10.59254/sbpo-2023-174910
http://live-proceedings.pantheonsite.io/p/174910

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* chamados WDMs (*Wavelength Division Multiplexers*) 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-completo.

Entre as heurísticas construtivas para gerar soluções para o MBV, citamos os trabalhos de [Cerulli et al., 2009], [Almeida et al., 2014], [Marín, 2015], [Melo et al., 2016] e [Moreno et al., 2016, 2018], que aliam 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 et al., 2016, 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, meta-heurísticas.

Na literatura, várias abordagens foram propostas para resolver o MBV. Em [Cerulli et al., 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. 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 et al., 2016]. Esse mesmo trabalho apresentou heurísticas construtivas eficazes para o problema que levam em consideração a estrutura do problema para obter boas soluções viáveis. Em [Silvestri et al., 2017], os autores desenvolveram uma formulação híbrida contendo variáveis direcionadas e não direcionadas, que foi resolvida por um algoritmo *branch-and-cut*. Por fim, o trabalho de [Moreno et al., 2016, 2018] apresentou 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 e 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 et al., 2018], onde foram apresentados métodos exatos e heurísticos que refinam as heurísticas inicialmente apresentadas em [Moreno et al., 2016].

Vale observar que o MBV é um caso específico do d-MBV onde $d = 2$. Isto é, dada uma árvore geradora T de um grafo G, uma *ramificação* de T é um vértice v tal que $d_T(v) > 2$, onde $d_T(v)$ denota o grau do vértice v na árvore T. Formalmente, o MBV é definido como se segue:

Entrada: Grafo $G = (V_G, E_G)$ não direcionado.
Objetivo: Encontrar uma árvore geradora T de G com número mínimo de ramificações.

DOI: 10.59254/sbpo-2023-174910
http://live-proceedings.pantheonsite.io/p/174910

Neste trabalho, propomos quatro novas heurísticas construtivas para o MBV. As duas primeiras, BEP e EEP, são algoritmos baseados em estratégias gulosas. As outras duas, R-BEP e R-EEP são variantes das primeiras e incorporam critérios randomizados. Em seguida, comparamos os resultados destes algoritmos com a melhor heurística construtiva da literatura (de acordo com o conhecimento dos autores deste trabalho), pertencente ao trabalho de [Moreno et al., 2018].

O restante deste trabalho se encontra estruturado da seguinte maneira: a Seção 2 descreve as heurísticas propostas. A Seção 3 apresenta os resultados obtidos por tais heurísticas. Por fim, as conclusões são discutidas na Seção 4.

2. Heurísticas Construtivas

Neste capítulo são apresentadas quatro novas heurísticas para a solução do MBV. Os algoritmos são semelhantes entre si, mas cada um apresenta particularidades em relação aos critérios usados na construção das árvores geradoras. Essas heurísticas foram inspiradas no algoritmo de [Prim, 1957], que busca uma árvore geradora mínima de um grafo conexo, não direcionado e valorado. O algoritmo de [Prim, 1957] 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 2.1 apresenta essa etapa de pré-processamento do grafo.

2.1. 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 e Tarjan, 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 2.2.

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

DOI: 10.59254/sbpo-2023-174910
http://live-proceedings.pantheonsite.io/p/174910

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 a 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 da 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 código do algoritmo de pré-processamento pode ser encontrado em Beltrão [2023].

2.2. Processo de Expansão de Ramificações

Como mencionado anteriormente, o MBV não possui nenhum critério que busque minimizar o grau das ramificações. 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 chama de `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 arvore, enquanto o segundo visa incorporar novas arestas sem adicionar ciclos ou criar novas ramificações. O processo de expansão de ramificações está documentado em Beltrão [2023].

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 nelas 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.

2.3. Algoritmos Gulosos

As duas primeiras heurísticas propostas nesse trabalho são algoritmos gulosos, cujas descrições estão nas subseções a seguir.

2.3.1. *Branch Expanding Prim*

A primeira etapa do Algoritmo BEP (*Branch Expanding Prim*) 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

DOI: 10.59254/sbpo-2023-174910
http://live-proceedings.pantheonsite.io/p/174910

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 vértice 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 2.2) 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 inócuo 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 1 apresenta o método descrito.

A Figura 1 ilustra a execução do Algoritmo BEP em um grafo conexo com $n = 8$ e $m = 11$ (Figura 1(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

DOI: 10.59254/sbpo-2023-174910
http://live-proceedings.pantheonsite.io/p/174910

Algoritmo 1: BEP - *Branch Expanding Prim*

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

1 $(T, B_T, Pontas) \leftarrow$ `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**
 `// Não existe uma nova ponta para ser explorada: necessário converter um vértice da árvore em ramificação.`
16 Seja $v \in V_T$ um vértice que maximiza $C(v) = |\{u \in adj_G(v) \mid u \notin V_T \vee$
 $(CConexa(u,T) \neq CConexa(v,T) \wedge (u \in B_T \vee d_T(u) = 1))\}|$
17 **se** $C(v) > 0$ **então**
 `//` Transformar v em ramificação permite estabelecer até $C(v)$ novas conexões:
18 $B_T \leftarrow B_T \cup \{v\}$
19 `ExpandirRamificação(`G, T, v`)`
20 **para** $u \in adj_G(v)$ **faça**
21 **se** $d_T(u) = 1 \wedge d_G(u) > 1$ **então**
22 $Pontas \leftarrow Pontas \cup \{u\}$
23 **senão**
 `// 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)

DOI: 10.59254/sbpo-2023-174910
http://live-proceedings.pantheonsite.io/p/174910

na Figura 1(b): a ponte foi adicionada ao grafo e a ramificação obrigatória, hachurada. Em seguida, as ramificações marcadas anteriormente passam pelo processo da Expansão (detalhado na Subseção 2.2). Após essa etapa inicial, o algoritmo identifica e marca todo vértice w que satisfaz $d_T(w) = 1$ e $d_G(w) > 1$ (Figura 1(c)). Os vértices que atendem à condição apresentada foram marcados em cinza e correspondem ao conjunto $Pontas$. Nas iterações seguintes (Figura 1(d) - (g)), 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. Esse vértice é então adicionado à solução em construção. Durante esse processo, um caso merece atenção: na Figura 1(e), o vértice v selecionado não possui um vizinho u que satisfaça as condições necessárias e, portanto, foi simplesmente descartado do conjunto $Pontas$. Ao longo do processamento, esse mesmo cenário ocorreu com os vértices 5, 6 e 8. A árvore geradora resultante é apresentada na Figura 1(h) e corresponde a uma solução ótima para a instância em estudo.

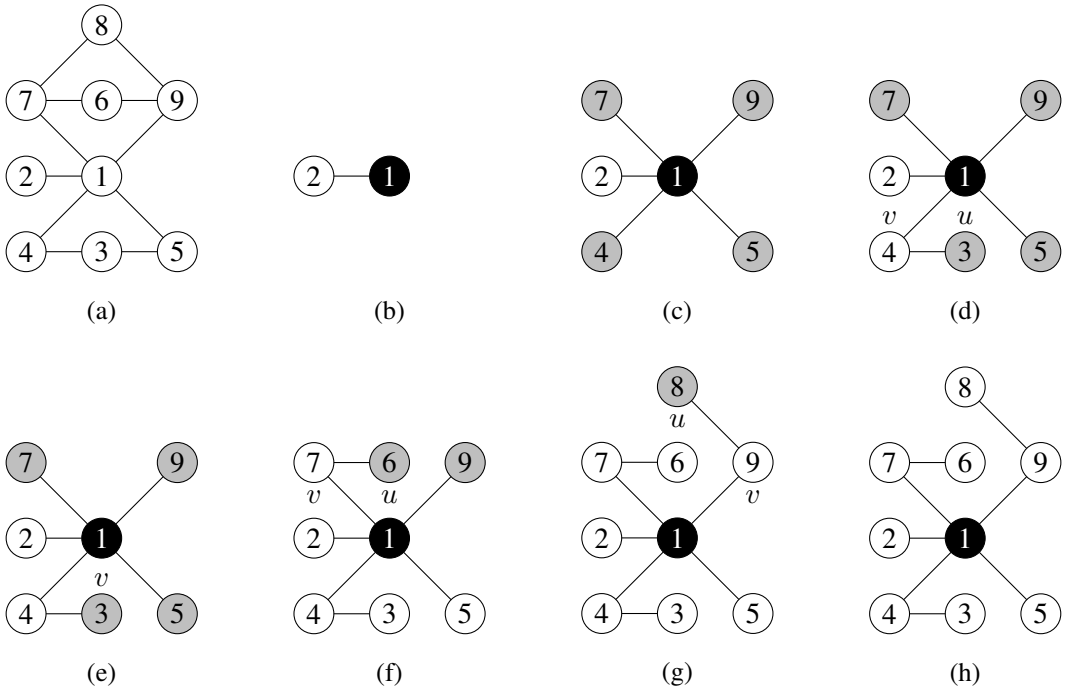


Figura 1: Exemplo de Aplicação do Algoritmo 1

2.3.2. *Edge Expanding Prim*

O Algoritmo EEP (*Edge Expanding Prim*) é uma variação do Algoritmo BEP, e seu 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 1).

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 está considerando 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 $(a, b) \in E_G$ que satisfazem as seguintes condições: (i) $a \in Pontas$; (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,

DOI: 10.59254/sbpo-2023-174910
http://live-proceedings.pantheonsite.io/p/174910

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 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 não admitem essas conexões.

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 de BEP pode ser estendida para o Algoritmo EEP, uma vez que as demais premissas seguem válidas. O Algoritmo EEP é detalhado em Beltrão [2023].

2.4. Algoritmos Randomizados

As duas 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 e EEP foram modificados para considerar uma componente randômica nas decisões. Essa componente randômica permite que os algoritmos retornem árvores diferentes das encontradas pelas versões determinísticas (BEP e EEP).

Entretanto, visando preservar a tendência natural dos métodos apresentados anteriormente, optamos por uma estratégia capaz de combinar os critérios já definidos e uma fonte de randomização: o uso de 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 e EEP, respectivamente, resultando nos Algoritmos R-BEP e R-EEP.

2.4.1. *Randomized Branch Expanding Prim*

A versão randomizada do BEP é o Algoritmo R-BEP (*Randomized Branch Expanding Prim*). Durante o tratamento do conjunto $Pontas$, em vez do algoritmo escolher 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

DOI: 10.59254/sbpo-2023-174910
http://live-proceedings.pantheonsite.io/p/174910

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, o algoritmo verifica 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 então ele 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 de os 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 do algoritmo. 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 no Algoritmo BEP. Além disso, as mudanças realizadas nos critérios de seleção não interferem com a corretude do método, uma vez que ainda preservam as condições originais. O R-BEP está detalhado em Beltrão [2023].

2.4.2. *Randomized Edge Expanding Prim*

A versão randomizada do EEP é o Algoritmo R-EEP (*Randomized Edge Expanding Prim*). Considerando que o EEP seleciona diretamente a aresta que será inserida na árvore, o R-EEP difere do R-BEP ao usar apenas uma roleta.

A inclusão da roleta no R-EEP é realizada logo após estabelecer o conjunto de arestas candidatas $ACandidatas$. A probabilidade de cada aresta $(a, b) \in ACandidatas$ é definida como $prob((a, b)) = \max\{d_G(c) + d_G(d), \forall (c, d) \in ACandidatas\} - (d_G(a) + d_G(b) + fator)$, onde $fator = 2$ se $b \notin V_T$, e $fator = 1$ caso contrário. Após construir a roleta, o algoritmo seleciona uma aresta (v, u) e continua exatamente o mesmo procedimentos adotado no EEP.

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. O Algoritmo R-EEP está descrito em Beltrão [2023].

3. Experimentos computacionais

Os experimentos realizados consistem na execução dos quatro 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. 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 et al., 2018]. A heurística construtiva de [Marín, 2015] também apresenta bons resultados, mas como utiliza busca local, preferimos não compará-la com o método aqui proposto.

DOI: 10.59254/sbpo-2023-174910
http://live-proceedings.pantheonsite.io/p/174910

Cabe ressaltar que o trabalho de [Moreno et al., 2018] não avalia os resultados obtidos apenas pela sua heurística construtiva gulosa. Para possibilitar uma comparação entre ela com as deste trabalho, reproduzimos os testes com o algoritmo ILS como proposto por [Moreno et al., 2018], mas interrompendo o algoritmo imediatamente após o algoritmo construtivo guloso, permitindo identificar os resultados de seu método construtivo isoladamente.

3.1. 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.

Nas tabelas apresentadas, 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 et al., 2018]. A quinta e a sexta coluna, respectivamente **BEP** e **EEP**, apresentam os resultados médios obtidos pelos algoritmos gulosos propostos neste trabalho. A sétima e oitava coluna representam as médias considerando a melhor solução obtidas pelas 100 execuções dos algoritmos R-BEP e R-EEP para cada uma das instâncias. As duas últimas colunas apresentam os *gaps* dos algoritmos R-BEP e R-EEP, considerando a diferença percentual entre cada algoritmo e a solução ótima para cada instância, sendo calculado da seguinte maneira: $gap = \frac{Algoritmo\ Randômico - Ótimo}{Ótimo} \times 100$.

A Tabela 1 apresenta os resultados para as instâncias médias, enquanto a Tabela 2 demonstra os resultados para as instâncias grandes. Para melhor identificação, marcamos em fonte *bold* os melhores resultados obtidos em cada grupo. Nas colunas **BEP** e **EEP**, marcamos em azul os valores que são iguais ou superiores aos obtidos pelo método de Moreno et al., para melhor comparação dos algoritmos determinísticos.

Em uma comparação restrita aos algoritmos determinísticos (o método de [Moreno et al., 2018], o BEP e o EEP – colunas 4,5 e 6), nota-se que o BEP foi capaz de superar o construtivo de [Moreno et al., 2018] em quase todos os grupos de instâncias, exceto em três. Já o EEP foi capaz de apontar resultados melhores ou equivalentes em todos os grupos de instâncias estudadas.

Analisando agora os resultados de forma global, nota-se que o Algoritmo R-BEP, em comparação aos demais métodos, apresentou o melhor resultado em 34 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. Em segundo lugar fica o R-EEP, atingindo o melhor resultado em 19 grupos de instâncias.

4. Conclusão

Neste trabalho foram desenvolvidas quatro heurísticas construtivas para abordar o problema MBV. Os experimentos realizados mostraram que tanto os algoritmos gulosos quanto os algoritmos randomizados obtiveram melhores resultados em comparação à heurística construtiva apresentada em [Moreno et al., 2018].

O Algoritmo guloso BEP gerou resultados superiores ao algoritmo de [Moreno et al., 2018] em 92,7% dos conjuntos de instâncias. Já o Algoritmo EEP apresentou resultados melhores

DOI: 10.59254/sbpo-2023-174910
http://live-proceedings.pantheonsite.io/p/174910

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

Tabela 1: Resultados para *Medium Instances*.

em 97,6%, empatando nos 2,4% restantes (um único caso). Já os dois 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 eficazes, atingindo soluções muito próximas aos valores ótimos (gap inferior a 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 2.2. Outra proposta é a integração dos algoritmos construtivos já desenvolvidos em algoritmos baseados em meta-heurísticas, visando aprimorar ainda mais os bons resultados encontrados.

Referências

Almeida, A. S., Nogueira, L. T., e Sá, V. G. P. (2014). Minimizando Ramificações em Árvores Geradoras. In *Anais do XLVI SBPO*, p. 2977–2985, Salvador. SOBRAPO.

Beltrão, A. (2023). Heurísticas Gulosas e Randomizadas para o Problema da Árvore Geradora com Número Mínimo de Ramificações. Dissertação de Mestrado, Instituto de Computação, Universidade Federal Fluminense, Niterói.

Carrabs, F., Cerulli, R., Gaudioso, M., e Gentili, M. (2013). Lower and upper bounds for the spanning tree with minimum branch vertices. *Comput. Optim. and Appl.*, 56(2):405–438.

Cerulli, R., Gentili, M., e Iossa, A. (2009). Bounded-degree spanning tree problems: models and new algorithms. *Computational Optimization and Applications*, 42(3):353–370.

Gargano, L., Hell, P., Stacho, L., e Vaccaro, U. (2002). Spanning trees with bounded number of branch vertices. In *Automata, Languages and Programming*, p. 355–365. Springer, Berlin, Heidelberg.

Hopcroft, J. e Tarjan, R. (1973). Algorithm 447: efficient algorithms for graph manipulation. *Communications of the ACM*, 16:372 – 378.

Marín, A. (2015). Exact and heuristic solutions for the minimum number of branch vertices spanning tree problem. *European Journal of Operational Research*, 245(3):680–689.

DOI: 10.59254/sbpo-2023-174910
http://live-proceedings.pantheonsite.io/p/174910

n	m	Ótimo	Moreno et al	BEP	EEP	R-BEP	R-EEP	Gap% (R-BEP)	Gap% (R-EEP)
600	637	183,8	187,6	185,4	185,6	**184,6**	**184,6**	0,4	0,4
600	674	167,2	173,4	171,4	171,0	170,2	**170,0**	1,8	1,7
600	712	150,6	159,4	156,2	156,6	**154,0**	154,2	2,3	2,4
600	749	138,8	145,4	145,6	144,4	**142,6**	143,2	2,7	3,2
600	787	125,8	135,4	133,8	133,8	**131,6**	132,0	4,6	4,9
700	740	214,4	218,8	217,0	217,2	**216,2**	**216,2**	0,8	0,8
700	780	198,0	204,4	202,6	202,0	**201,0**	**201,0**	1,5	1,5
700	821	180,0	189,0	185,6	185,8	**184,0**	**184,0**	2,2	2,2
700	861	164,0	174,2	171,4	170,8	168,8	**168,6**	2,9	2,8
700	861	164,0	174,2	171,4	170,8	168,8	**168,6**	2,9	2,8
700	902	154,2	165,2	161,8	162,2	**159,0**	159,6	3,1	3,5
800	843	245,6	250,8	248,4	248,8	**247,8**	**247,8**	0,9	0,9
800	886	227,6	235,0	232,0	231,8	**230,6**	**230,6**	1,3	1,3
800	930	208,4	216,2	215,2	215,2	**212,6**	213,2	2,0	2,3
800	973	194,2	205,0	201,6	200,6	**198,8**	199,0	2,4	2,5
800	1017	176,2	189,0	184,8	184,4	**182,6**	182,8	3,6	3,7
900	944	279,6	285,2	282,8	282,6	**282,0**	**282,0**	0,9	0,9
900	989	259,2	268,8	264,8	264,4	**263,4**	**263,4**	1,6	1,6
900	1034	240,6	250,6	247,0	247,4	245,0	**244,6**	1,8	1,7
900	1079	223,2	235,8	232,4	231,0	229,6	**229,4**	2,9	2,8
900	1124	206,0	219,0	214,8	214,2	**213,0**	213,2	3,4	3,5
1000	1047	312,0	317,4	315,4	**314,6**	**314,6**	**314,6**	0,8	0,8
1000	1095	290,0	299,6	295,6	296,0	294,4	**294,2**	1,5	1,4
1000	1143	271,2	283,0	277,8	278,4	276,2	**276,0**	1,8	1,8
1000	1191	251,0	264,0	259,4	259,4	**257,2**	**257,2**	2,5	2,5
1000	1239	235,2	249,6	244,0	244,4	**242,2**	**242,2**	3,0	3,0

Tabela 2: Resultados para *Large Instances*.

Melo, R., Samer, P., e Urrutia, S. (2016). An effective decomposition approach and heuristics to generate spanning trees with a small number of branch vertices. *Comput. Optim. and Appl.*, 65 (3):821–844.

Merabet, M. e Molnar, M. (2016). Generalization of the minimum branch vertices spanning tree problem. Phd thesis, Nanyang Technological University, Singapore.

Moreno, J., Frota, Y., e Martins, S. (2018). An exact and heuristic approach for the d-minimum branch vertices problem. *Computational Optimization and Applications*, 71(3):829–855.

Moreno, J., Plastino, A., e Martins, S. (2016). Heurística Iterated Local Search para Resolver o Problema Minimum Branch Vertices. In *Anais do XLVIII SBPO*, p. 1896–1907, Vitória. SOBRAPO.

Prim, R. (1957). Shortest connection networks and some generalizations. *The Bell System Technical Journal*, 36:1389–1401.

Silva, R., Silva, D., Mateus, G., Gonçalves, J., Resende, M., e Festa, P. (2014). An edge-swap heuristic for generating spanning trees with minimum number of branch vertices. *Optimization Letters*, 8(4):1225–1243.

Silvestri, S., Laporte, G., e Cerulli, R. (2017). A branch-and-cut algorithm for the minimum branch vertices spanning tree problem. *Computers and Operation Research*, 81:322–332.

DOI: 10.59254/sbpo-2023-174910
http://live-proceedings.pantheonsite.io/p/174910