UNIVERSIDADE FEDERAL DA FRONTEIRA SUL
CAMPUS CHAPECÓ
CURSO DE CIÊNCIA DA COMPUTAÇÃO

PEDRO HENRIQUE PIAIA DARIVA

UMA ANÁLISE DE MÉTODOS DE SEPARAÇÃO DE GRAFOS QUANDO APLICADOS AO PROBLEMA DA ÁRVORE T-SPANNER DE PESO MÍNIMO

CHAPECÓ
2025

PEDRO HENRIQUE PIAIA DARIVA

UMA ANÁLISE DE MÉTODOS DE SEPARAÇÃO DE GRAFOS QUANDO APLICADOS AO PROBLEMA DA ÁRVORE T-SPANNER DE PESO MÍNIMO

Trabalho de Conclusão de Curso apresentado ao Curso de Ciência da Computação da Universidade Federal da Fronteira Sul (UFFS), como requisito para obtenção do título de Bacharel em Ciência da Computação.

Orientador: Prof. Dr. Andrei de Almeida Sampaio Braga

CHAPECÓ
2025

Dariva, Pedro Henrique Piaia

UMA ANÁLISE DE MÉTODOS DE SEPARAÇÃO DE GRAFOS QUANDO APLICADOS AO PROBLEMA DA ÁRVORE T-SPANNER DE PESO MÍNIMO / Pedro Henrique Piaia Dariva – 2025.

40 f.

Orientador: Prof. Dr. Andrei de Almeida Sampaio Braga

Trabalho de Conclusão de Curso (Graduacão) – Universidade Federal da Fronteira Sul, Curso de Ciência da Computação, Chapecó, SC, 2025.

1. Programação Linear Inteira 2. Grafos 3. Árvores 4. *Spanners* I. Braga, Dr. Andrei de Almeida Sampaio, orient. II. Universidade Federal da Fronteira Sul. III. Título.

PEDRO HENRIQUE PIAIA DARIVA

UMA ANÁLISE DE MÉTODOS DE SEPARAÇÃO DE GRAFOS QUANDO APLICADOS AO PROBLEMA DA ÁRVORE T-SPANNER DE PESO MÍNIMO

Trabalho de Conclusão de Curso apresentado ao Curso de Ciência da Computação da Universidade Federal da Fronteira Sul (UFFS), como requisito para obtenção do título de Bacharel em Ciência da Computação.

Orientador: Prof. Dr. Andrei de Almeida Sampaio Braga

Este Trabalho de Conclusão de Curso foi avaliado e aprovado pela banca avaliadora em: 12/12/2025.

BANCA AVALIADORA

Prof. Dr. Andrei de Almeida Sampaio Braga - UFFS

Prof. Dr. Marco Aurélio Spohn - UFFS

Prof. Dr. Samuel da Silva Feitosa - UFFS

DEDICATÓRIA

Dedico este trabalho a todas as pessoas que contribuíram para a minha trajetória até o presente momento, bem como a você, caro leitor.

“Just man up and learn what malloc and free are.”
(Terry A. Davis)

AGRADECIMENTOS

Agradeço a minha mãe, Magda Regina Piaia por todo o investimento, dedicação, suporte e carinho. Agradeço também ao meu orientador Andrei de Almeida Sampaio Braga por acreditar em mim durante todo o processo e por ser meu mentor durante a maior parte de minha formação. Por fim, agradeço aos amigos pela companhia constante e pelos votos sinceros de sucesso. Nada disso teria sido possível sem vocês.

RESUMO

O problema da árvore *t-spanner* de peso mínimo consiste em encontrar uma árvore geradora onde a distância entre qualquer par de vértices no subgrafo seja no máximo t vezes a distância no grafo original. Após essa condição, busca-se pelo somatório dos pesos das arestas que seja o menor possível. Esse problema tem variados usos na área de redes de computadores devido à sua propriedade de gerar estruturas concisas mas eficientes. Entretanto, mesmo utilizando técnicas como a programação linear inteira, o tempo de execução para instâncias maiores diminui o seu uso como um recurso prático. Nesse contexto, pode-se usar a técnica de separação por pontos de articulação, que se resume em dividir o grafo em diferentes blocos, determinados por seus pontos de articulação, e após encontrar a resposta do problema para cada uma, uni-las para formar a resposta final. Essa técnica ainda não foi testada e pode proporcionar ganho de desempenho ao buscar uma resposta ótima para o problema. Em virtude disso, entende-se necessário testar essa possibilidade. Ademais, levando em consideração os casos onde não existam pontos de articulação, é interessante testar também uma técnica de separação similar para os grafos cordais baseado em cliques separadores balanceados. A análise dos experimentos indicou que a decomposição por pontos de articulação contribuiu com a redução de cerca de 80% do tempo de execução no caso médio, em contrapartida com a abordagem baseada em cliques separadores apresentou piora do tempo de execução.

Palavras-Chave: Programação Linear Inteira, Grafos, Árvores, *Spanners*.

ABSTRACT

The minimum t-spanner tree problem consists of finding a spanning tree where the distances between any pair or vertices are at most t times that of the original graph. After that, one searches for a solution where the sum of edge weights is minimized. This problem has many uses when it comes to computer networks due to its property of creating concise but efficient graph structures. Nevertheless, even when using techniques like integer linear programming, the execution time for large instances diminishes its use in a practical sense. In this context, techniques such as the separation by articulation points, that involves separating different blocks of the grapph, determined by its articulation points, and after finding the answer for each block, they are combined to find the proper solution. This technique has not been properly tested and can result in the gain of performance when searching for an optimal solution for the problem. By virtue of this fact, it is deemed necessary to properly test this possibility. Furthermore, taking into consideration the cases where articulation points do not exist, tests are made with a similar technique for chordal graphs, based on balanced clique separators. Analysis of the experiments indicated that decomposition by articulation points contributed to a reduction of approximately 80% in execution time on average, in contrast to the approach based on the balanced clique separator, which resulted in a worsening of execution time.

Keywords: Integer Linear Programming, Graphs, Trees, Spanners.

LISTA DE FIGURAS

2.1 Grafo simples 15
2.2 Caminho 16
2.3 Árvore Geradora Mínima 17
2.4 Grafo cordal 18
2.5 Pontos de articulação 18
2.6 Árvore *t-spanner* de peso mínimo. 20
6.1 Diagrama de dispersão - Ponto de Articulação 34
6.2 Perfil de Desempenho - Ponto de Articulação 35
6.3 Perfil de Desempenho - Clique Separador Balanceado 36

LISTA DE TABELAS

6.1 Tabela de comparação por número de vértices - Ponto de Articulação 33
6.2 Tabela de comparação por número de blocos - Ponto de Articulação 33
6.3 Tabela de comparação por tamanho médio dos blocos - Ponto de Articulação 33
6.4 Tabela de comparação - Clique Separador Balanceado (Modelo) 35
6.5 Tabela de comparação - Clique Separador Balanceado (MST) 36

LISTA DE ALGORITMOS

1 Algoritmo adaptado de Hopcroft e Tarjan 28
2 Algoritmo para encontrar centroides 29
3 Algoritmo de formação de blocos 30

SUMÁRIO

1 INTRODUÇÃO **13**
1.1 RESULTADOS 14

2 FUNDAMENTAÇÃO TEÓRICA **15**
2.1 CONCEITOS DE TEORIA DOS GRAFOS 15
2.2 PROGRAMAÇÃO LINEAR INTEIRA 18
2.3 *SPANNERS* 19

3 TRABALHOS RELACIONADOS **21**

4 METODOLOGIA **23**

5 MÉTODO DE RESOLUÇÃO **25**
5.1 MODELO DE PROGRAMAÇÃO LINEAR INTEIRA 25
5.2 SEPARAÇÃO POR PONTO DE ARTICULAÇÃO 27
5.3 SEPARAÇÃO POR CLIQUE SEPARADOR BALANCEADO 28

6 RESULTADOS **31**
6.1 INSTÂNCIAS 31
6.2 AMBIENTE COMPUTACIONAL 32
6.3 SEPARAÇÃO POR PONTOS DE ARTICULAÇÃO 32
6.3.1 SEPARAÇÃO POR CLIQUE SEPARADOR BALANCEADO 35

7 CONCLUSÃO **38**

REFERÊNCIAS **39**

1. INTRODUÇÃO

Em diversas situações na área de redes de computadores, utilizam-se grafos para representar topologias, onde frequentemente é necessário atingir um ponto de equilíbrio entre o custo estrutural dessas redes e seu desempenho. Para tal, existem o conceito de *t-spanners*, que possuem a propriedade de serem esparsos além de minimizarem as distâncias entre os vértices da topologia (PELEG; SCHäFFER, 1989). Problemas de *t-spanners* são apontados quando é necessário a flexibilidade para lidar com uma ampla gama de topologias (PELEG, 2002).

Geralmente busca-se por uma rede mais esparsa e com menor fator de dilatação possível (PELEG; SCHäFFER, 1989). No caso, existe certa relação entre o quanto um grafo é esparso e o desempenho do roteamento de pacotes na rede. Além disso, topologias mais densas tendem a requerer mais recursos para manter seu funcionamento ideal. Por meio dessas características, existe possibilidade de aplicar *spanners* para sistemas distribuídos, como exemplificado pelo *Arrow Distributed Directory Protocol*, descrito por Demmer e Herlihy (1998). Ponto peculiar desse protocolo é o uso de um *spanner* esparso como estrutura para a comunicação entre diferentes nodos e repasse de arquivos na rede.

Devido à natureza combinatória de encontrar *t-spanners* ótimos, depende-se de técnicas como a programação linear inteira, que permite encontrar soluções ótimas de maneira mais ágil do que a busca completa. Entretanto, o tempo de execução para instâncias maiores continua elevado. Para o problema da árvore *t-spanner* de peso mínimo, o modelo de programação linear inteira apresentado por Braga (2018) é uma ferramenta robusta para tal tarefa. Como é demonstrado no trabalho de Sousa (2021), é possível, seguindo alguns critérios, separar o grafo em blocos biconexos, calcular individualmente suas respostas e uni-las novamente sem perda de generalidade. Algo similar pode ser feito para os grafos cordais, uma vez que podem ser separados por cliques separadores balanceados, princípio que Dragan e Köhler (2014) demonstram em seu trabalho.

Como demonstrado no trabalho de Braga (2018), calcular respostas para instâncias menores é muito mais rápido do que para maiores instâncias. O que leva a indagação, se é possível utilizarmos das características de terminados tipos de grafos para conseguir melhoria no tempo de execução. Pelo que se tem conhecimento, a aplicação direta de métodos de separação no contexto do problema da árvore *t-spanner* de peso mínimo ainda não foi estudada. O objetivo do presente trabalho é demonstrar como os métodos de separação por ponto de articulação e por clique separador balanceado impactam o desempenho do modelo de programação linear inteira, quando aplicados no contexto da construção de uma árvore t-spanner de peso mínimo. De maneira mais específica, ambos os métodos são implementados e integrados ao modelo, sendo realizados experimentos computacionais para cada abordagem, cujos resultados são analisados e comparados.

1.1 Resultados

Ao avaliarmos o desemprenho de ambos os métodos de separação, os resultados indicam que o método por pontos de articulação fornece ganhos significativos de desempenho, com redução de cerca de 80% no tempo de execução. Observa-se também que para instâncias grandes o método apresenta um desempenho melhor, tornando-o uma escolha robusta para instâncias com grande numero de vértices. Por outro lado, a técnica baseada em cliques separadores balanceados mostrou-se sistematicamente menos eficiente que o modelo de Braga (2018), sobretudo em grafos de maior escala, devido a baixa reutilização das soluções parciais e do custo computacional adicional adquirido ao calcular as mesmas. Dessa forma, conclui-se a viabilidade prática da abordagem por pontos de articulação e as limitações do método baseado em cliques separadores balanceados.

2. FUNDAMENTAÇÃO TEÓRICA

No presente capítulo são apresentados os conceitos mais pertinentes para o entendimento desse trabalho. Entre eles, os conceitos básicos de teoria dos grafos (Seção 2.1), Programação Linear Inteira (Seção 2.2), *Spanners* e Árvores *t-spanner*(Seção 2.3).

2.1 Conceitos de Teoria dos Grafos

Segundo Jungnickel (2013), um grafo G é definido por $G = (V, E)$ de forma que V, sujeito a $V \neq \emptyset$, representa o conjunto dos vértices pertencentes ao grafo. O conjunto E, representando as arestas, é definido como um subconjunto de V, no formato de um par não ordenado $V \times V$. Se existe um aresta $e \in E$ com suas pontas em u e v, dizemos que v é vizinho de u, além disso, a notação $e = vu$ é usada para se referir a uma aresta. Utilizamos os símbolos $V(G)$ e $E(G)$ para um grafo qualquer G, que representam o conjunto dos vértices e das arestas respectivamente, enquanto $|V|$ e $|E|$ a quantidade de itens de cada um deles. Um exemplo de grafo é mostrado na Figura 2.1.

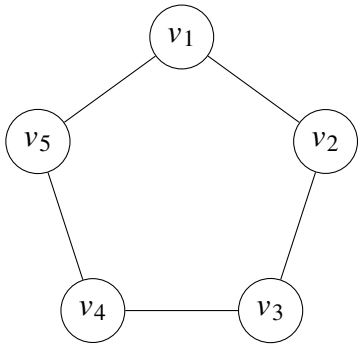

Figura 2.1: Grafo G composto por $V = \{v_1, v_2, v_3, v_4, v_5\}$ e $E = \{v_1v_2, v_2v_3, v_3v_4, v_4v_5, v_5v_1\}$

De acordo com Diestel (2018), para um grafo $G = (V, E)$, dizemos que $H = (V', E')$ é um subgrafo de G, se $V' \subseteq V$ e $E' \subseteq E$. Desse modo, podemos escrever $H \subseteq G$. Caso $V' = V$, o subgrafo é denominado gerador. Sempre que H contiver todas as arestas $uv \in E(G)$ onde $u, v \in V'$, consideramos o subgrafo como induzido. Além disso, um subgrafo H de G é considerado próprio, se $H \neq G$.

Um passeio em um grafo G é definido como uma sequência de vértices $v_1, v_2, \ldots, v_n$, tal que toda aresta v_iv_{i+1}, onde $i = 1, \ldots, n-1$, pertença ao grafo. Em caso onde não houver repetição de vértices, podemos montar a sequência de vértices em um subgrafo $P \subseteq G$, considerado um caminho. Um exemplo de um caminho é visível na Figura 2.2. De maneira similar, um ciclo c é representado por um caminho onde $v_1 = v_n$. Caso não existam ciclos no grafo, ele é considerado acíclico. Em um grafo, é possível que existam muitos caminhos

diferentes, entretanto, é útil pensarmos no menor caminho possível dentre todos. Entre os vértices de um grafo G, $d_G(v, u)$ representa a distância percorrida pelo menor caminho entre v e u, a distância é medida a partir da quantidade de arestas presentes no menor caminho. Caso não exista um caminho entre v e u a distância é interpretada como infinita.

Um par de vértices v e u é dito conectado caso exista um caminho entre eles. Se progredirmos o conceito para o conjunto maior de vértices, podemos dizer que um subgrafo H é conexo, se para cada vértice $v \in V(H)$ existe um caminho para qualquer outro. Seja H um subgrafo conexo maximal de G, então H é dito como uma componente conexa do grafo G. Se $H = G$ consequentemente G é um grafo conexo.

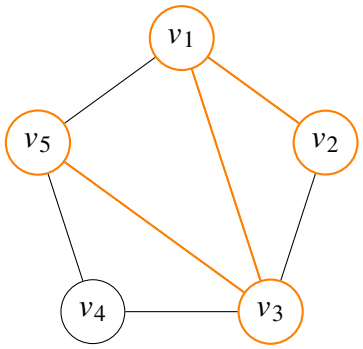

Figura 2.2: Grafo G com o caminho $P = v_5, v_3, v_1, v_2$

Em alguns tipos de problema, é proveitoso atribuir direções às arestas de um grafo, esses grafos são chamados de grafos direcionados ou digrafos. Um dígrafo $G = (V, E)$ é definido de maneira similar a um grafo não direcionado, V permanece um conjunto finito representando os vértices, enquanto E é subconjunto de V no formato de um par ordenado $V \times V$, representando as arestas. As arestas de um dígrafo também são chamadas de arcos. Dessa forma, para o arco $e = vu$ só pode se movimentar do vértice v para u. Logo, v é dito como o vértice de começo e u como o vértice final.

Para um grafo $G = (V, E)$, se ele for acíclico e conexo, ele é classificado como uma árvore. Uma árvore pode ter folhas, que são definidas como qualquer vértice v que tenha apenas um vizinho. É possível selecionar um vértice do grafo e declará-lo como uma raiz, dizemos que G é enraizado em v. Para todo grafo G existe um subgrafo H que contém todos os vértices de G, mas contém somente arestas suficientes para que H seja conexo e acíclico, H então é dito como uma árvore geradora do grafo G.

Uma arborescência é uma árvore direcionada enraizada, de forma que todos os arcos apontam para a direção contrária de sua raiz. Se a arborescência for adquirida a partir de uma árvore, ela tem a garantia de descrever os menores caminhos do vértice raiz para qualquer vértice intermediário ou folha.

Outra abstração proveitosa é adicionar peso às arestas de um grafo, tal grafo é denominado como um grafo ponderado. Define-se um grafo ponderado pelo par (G, w), onde

$w : E \rightarrow \mathbb{R}$ mapeia os pesos de cada aresta. Assim, ao calcular-se as distância soma-se os pesos das arestas. Para um grafo ponderado conexo $G = (V, E)$, a árvore geradora de peso mínimo é descrita como um subgrafo gerador $H = (V, T)$ que é acíclico, conexo e onde o peso de T é o menor possível, como detalha a Figura 2.3. O peso de T é descrito por:

$$w(T) = \sum_{e \in T} w(e)$$

Seja um Q um subconjunto dos vértices do grafo $G = (V, E)$, se para todo vértice $v \in Q$, v for vizinho de todas os outros, esse conjunto de vértices denominado uma clique. Para um grafo conexo $G = (V, E)$, um separador S é caracterizado como um subconjunto dos vértices de G, de tal forma que, o grafo obtido ao remover S de G tenha mais do que uma componente. Tal separador é considerado balanceado se o tamanho da maior componente de $G \setminus S$ seja menor do que $\frac{|V|}{2}$. Dessa forma, um clique separador balanceado é um subconjunto de vértices S, onde S é uma clique, bem como um separador balanceado.

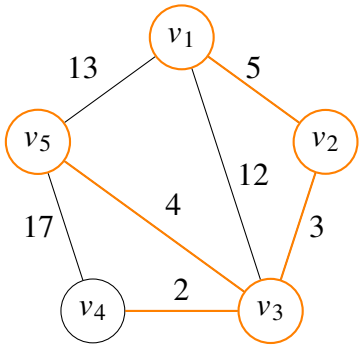

Figura 2.3: Em destaque, a árvore geradora de peso mínimo de um grafo

Um grafo G é denominado cordal, se qualquer subgrafo induzido de G que seja um ciclo tenha exatamente 3 vértices. Fato interessante dos grafos cordais apresentado por Gilbert, Rose e Edenbrandt (1984), é que todo grafo cordal contem uma clique maximal que é um separador balanceado. Dessa forma permitindo a divisão do grafo em blocos distintos. Os grafos das Figuras 2.3 e 2.4 são exemplos de grafos cordais.

Um vértice de corte ou um ponto de articulação, é todo e qualquer vértice v de um grafo $G = (V, E)$, de forma que, ao remover v de G, o grafo gerado possua mais componentes conexas do que o original. Todo ponto de articulação, por via de regra, é um separador mínimo de G. Por meio dos pontos de articulação, é possível agrupar certos grupos de vértices em subgrafos conexos distintos. Se gerarmos um novo grafo H onde cada um desses grupos de vértices é comprimido em um único vértice, H é garantidamente uma árvore.

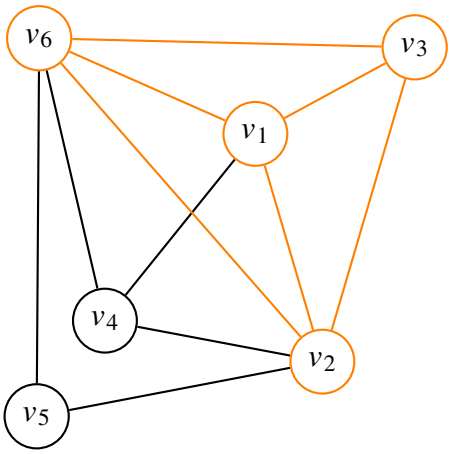

Figura 2.4: Exemplo de grafo cordal, destacando o clique separador maximal

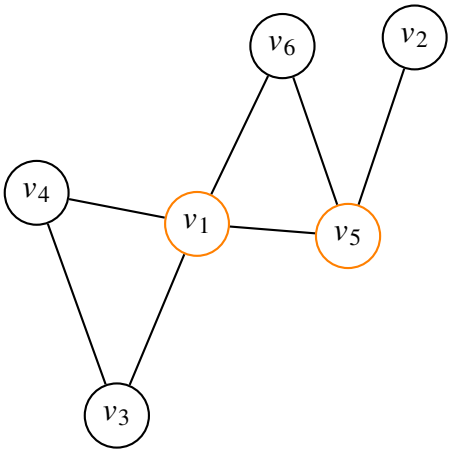

Figura 2.5: Grafo com pontos de articulação

2.2 Programação Linear Inteira

Ao se tratar de otimização combinatória, é uma área de estudo que necessita manipulação eficiente de recursos, tendo em vista a complexidade dos problemas que ela abrange. Programação Linear Inteira, uma técnica utilizada nessa área, desempenha papel crucial para a resolução de tais de problemas de maneira ágil e exata (NEMHAUSER; WOLSEY, 1999). Para usar programação linear inteira é necessário a criação de um modelo, composto por uma função de objetivo, que aponta para o modelo qual solução buscar; as condições, que limitam o espaço de soluções viáveis; e as variáveis, que definem a instância do problema. Um modelo geral assume a forma de:

$$\max \sum cx;\ ax \leq b;\ x \in \mathbb{Z}$$

Um problema clássico de programação linear inteira é o problema da mochila. O problema consiste em uma coleção de itens, cada um com um valor e um peso atribuído. O

objetivo é selecionar um conjunto desses itens que maximize o valor total dos itens selecionados, enquanto o somatório dos pesos fica abaixo da capacidade da mochila (WOLSEY, 1998).

Assim, $n \in \mathbb{N}$ se refere ao número de itens e $W \in \mathbb{R}$ o peso máximo que a mochila suporta. Para $i = 1, \ldots, n,\ x_i \in \mathbb{B},\ c_i, v_i \in \mathbb{R}$, onde x_i valendo 0 ou 1, representando a escolha de um item ou não para o conjunto de resposta, c_i é o seu peso, e v_i é o seu valor. Para garantir que W não seja ultrapassado, é necessário que a condição $x_1c_1 + x_2c_2 + \cdots + x_nc_n \leq W$ seja verdadeira. Então busca-se pela solução onde a soma $x_1v_1 + x_2v_2 + \cdots + x_nv_n$ seja máxima. De maneira consolidada:

$$\max \sum_{i}^{n} x_i v_i$$

$$\text{sujeito a} \sum_{i}^{n} x_i c_i \leq W$$

$$x_i \in \{0, 1\} \qquad \forall i = 1, \ldots, n$$

Para um modelo de programação linear inteira, é possível por meio de um resolvedor buscar pela solução ótima.

2.3 *Spanners*

Para um grafo $G = (V, E)$, denotamos um de seus subgrafos geradores H de *t-spanner*, se e somente se, para todo par de vértices $u, v \in V(G)$ a distância $d_H(u, v)$ é no máximo t vezes a distância $d_G(u, v)$, onde t é denominado o fator de dilatação. O problema foi primeiro apresentado por Chew (1986) e, posteriormente, o termo *spanner* foi cunhado por Peleg e Schäffer (1989). Na forma mais clássica do problema, G é um grafo ponderado e conexo. Assim convém o conceito do problema do *t-spanner* de peso mínimo, de minimizar o somatório total das arestas escolhidas para o *spanner*.

Ao impor um fator de dilatação t maior, permitimos que o subgrafo gerador resultante tenha maiores distâncias par a par entre seus vértices. Em troca, caso seja proveitoso, diminui-se a quantidade de arestas no subgrafo gerador. Em geral, existe uma relação inversamente proporcional entre o fator de dilatação e a quantidade de arestas do grafo: se mantivermos t fixo, então existe maior chance de encontrar uma resposta que não tenha todas as arestas do grafo; em contrapartida, se diminuirmos a quantidade de arestas, o t mínimo para uma resposta viável tende a aumentar. Nesse contexto, podemos pensar em exigir que o subgrafo gerador seja uma

árvore, dessa forma mantemos a conectividade e resumimos ao máximo as arestas do subgrafo. Ao adicionar tal condição, geramos então o problema da árvore *t-spanner* de peso mínimo.

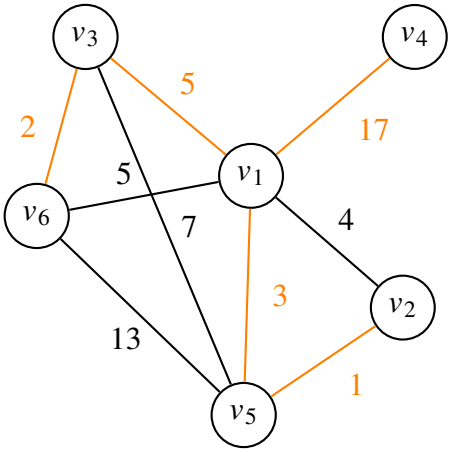

Figura 2.6: Árvore *t-spanner* de peso mínimo onde $t = 2$

3. TRABALHOS RELACIONADOS

Durante esse capítulo, são apresentados trabalhos relacionados ao problema da árvore *t-spanner* de peso mínimo, bem como *spanners* no geral. Foco especial é direcionado aos trabalhos que utilizam programação linear inteira como abordagem principal.

Desde as primeiras abordagens dos problemas de *spanner*, busca-se por maneiras de encontrar *spanners* mais esparsos, ou seja, com menor quantidade de arestas possível. Naturalmente, esse processo leva ao problema da árvore *t-spanner* de peso mínimo (PELEG; SCHäFFER, 1989).

Na tese de Braga (2018), com o objetivo de abordar o problema da árvore *t-spanner* de peso mínimo e o problema do *t-spanner* de peso mínimo, o autor apresenta as primeiras formulações de programação linear inteira específicas para o problema, além de testes computacionais em grafos aleatórios.

Dando ênfase ao problema da árvore *t-spanner* de peso mínimo, foram apresentados dois modelos, um com rótulos e o outro sem. Ambos os modelos são baseados na ideia de arborescências, que permitem o mesmo a ser eficiente e de tamanho polinomial referente a G. O principal diferencial entre os modelos é a maneira como se lida com as distâncias, essas podem ou não ser armazenadas em variáveis próprias, descrevendo o modelo sem rótulos e modelo com rótulos, respectivamente. Além disso, como demonstrado por Braga (2018), o uso de rótulos no modelo ocasiona melhoria de desempenho.

Posteriormente, Sousa (2021) reavaliou os modelos apresentados por Braga (2018) e introduziu duas novas formulações, uma delas aprofundou mais a região viável do problema. Além disso, é necessário ressaltar a ideia de decomposição de blocos que o autor apresentou. Separando o grafo em blocos por meio de vértices de articulação, é possível avaliar cada bloco separadamente e depois unir o resultado de cada bloco. Sousa (2021) ressaltou que há possibilidade de tal técnica resultar em melhor desempenho.

Com o objetivo de implementar uma solução para o problema da árvore *t-spanner* de peso mínimo e possivelmente atingir melhorias por meio de heurísticas, Vasconcellos e Miyazawa (2022) realizam testes computacionais com diferentes tipos de grafos se baseando na formulação de Braga (2018). Os resultados destes demonstraram maior performance do modelo de arborescências ao se tratar de classes específicas de grafos, como os grafos planares e bipartidos.

Um dos primeiros algoritmos gulosos para o problema do problema do *t-spanner* de peso mínimo é apresentado por Althöfer et al. (1993), se baseando no algoritmo de Kruskal para o problema da árvore geradora de peso mínimo. Comparações de diferentes heurísticas foram feitas por Chimani e Stutzenstein (2021), testando-as em vários tipos de grafos diferentes, o que realçou o algoritmo de Althöfer et al. (1993) como uma das heurísticas mais performáticas.

No trabalho de Dragan e Köhler (2014), foi demonstrado, para o problema do *t-spanner* de peso mínimo, um algoritmo de aproximação com um conceito de separação por discos balanceados, com generalização para grafos cordais. Um ponto particular desse trabalho foi a especificidade aos grafos cordais, como mencionado pelo autor, um grafo cordal pode ser quebrado em blocos de subgrafos cordais por meio de um clique separador, e esses transformados em árvores *spanners*.

Miranda e Sinnl (2019) trataram de outro problema relacionado, o problema da árvore *t*-spanner*. Esse problema é uma variação do problema da árvore *t-spanner* de peso mínimo, que encontra o menor fator de dilatação t possível para um grafo específico, tal que para esse fator exista uma árvore *t-spanner*. Os autores desenvolveram duas formulações de programação linear inteira, as primeiras na literatura a resolver o problema de maneira exata.

Ao se tratar dos testes computacionais realizados pelos autores, demonstra-se uma conclusão interessante, de que ao menos 40% das arestas presentes na árvore geradora de peso mínimo fazem parte da solução ótima no caso médio. Logo, são fortes candidatas para a solução final.

Com relação aos trabalhos apresentados nessa seção, se faz necessário apontar um padrão. A maioria das heurísticas abordadas tem seu desempenho marcado pela classe de grafo para a qual elas são aplicadas. Não somente isso, mas o tamanho das entradas para os testes computacionais é razoavelmente baixo. Por via de regra, dizer que existe correlação entre a classe de um grafo e um resultado ótimo do algoritmo é apropriado. Além disso, Sousa (2021), Dragan e Köhler (2014) demonstram que é possível repartir o grafo original em blocos. Existe então a possibilidade de que, se o grafo for dividido, haja melhorias ao seu desempenho. Tudo isso, motiva uma investigação aprofundada das peculiaridades de cada classe e torna relevante uma avaliação do comportamento dos métodos de separação para ganho de desempenho. No presente trabalho, são exploradas tais questões.

4. METODOLOGIA

No presente capítulo, são elucidados os passos realizados no desenvolvimento desse trabalho. Cada etapa representa uma visão geral dos objetivos cumpridos.

Implementação do modelo de Braga (2018). Como primeiro passo, foi implementar o modelo desse autor. Como o objetivo do presente trabalho é estudar possível melhoria de desempenho, foi escolhida a linguagem de programação C++ devido à sua ampla biblioteca de *templates* (*STL*) e sua velocidade em comparação com outras linguagens. Em compromisso com essa escolha, também foi feito o uso do Gurobi (2025), resolvedor de programação linear inteira com licença gratuita para estudantes.

Implementação dos métodos de separação por ponto de articulação e por clique separador balanceado. Em relação ao método por pontos de articulação, foi implementado o algoritmo de busca em profundidade modificado, descrito por Hopcroft e Tarjan (1973), para encontrá-los. Já para o método de clique separador balanceado, o artigo de Blair e Peyton (1993) delimita um algoritmo útil para tal tarefa. Além disso, foi utilizado o algoritmo de Giustina, Prezza e Venturini (2019) para encontrar o clique separador balanceado de cada instância.

Integração dos métodos de separação com o modelo. Devido à natureza de cada método de divisão, foi necessário realizar a integração adequada com o modelo. A separação por ponto de articulação foi a mais fácil, afinal, é somente necessário unir os resultados de cada bloco, para obter a solução final. Já para o método por clique separador balanceado, foi necessário repassar os resultados dos blocos novamente pelo modelo para garantir uma resposta exata.

Coleta e gerarão de instâncias de teste. Foi realizado um levantamento das instâncias testadas em trabalhos relacionados, como o trabalho de Sousa (2021). Entretanto, para os propósitos do nosso trabalho essas não foram as mais apropriadas para uso. Para a realização dos experimentos, foi gerado um número razoável de instâncias para cada uma das classes, nomeadamente grafos com pontos de articulação e grafos cordais. Ademais, foi utilizada a ferramenta *NetworkX* que contém vários métodos úteis para a geração de grafos das classes citadas (HAGBERG; SCHULT; SWART, 2008).

Realização dos experimentos. Cada um dos métodos foi testado para suas respectivas instâncias: o modelo original para todas as instâncias, o modelo integrado com o método do clique separador balanceado para os grafos cordais e o modelo integrado com o método de separação por pontos de articulação para grafos que possuam essa característica. O modelo original foi utilizado para a comparação dos resultados. Para os casos de teste, o tempo de execução foi o principal critério de comparação.

Análise dos resultados obtidos nos experimentos. Após a realização dos experimentos, foi realizada a compilação dos resultados e a comparação do desempenho dos diferentes métodos. Isso foi feito com o intuito de elucidar possíveis tendências presentes no desempenho do algoritmo. Ao se tratar do método por pontos de articulação, também foi comparado o tamanho médio de cada bloco, em quantos blocos uma instância fora dividida e tempo médio de execução por classe. Enquanto no método para os grafos cordais, somente o tempo de execução foi levado em consideração.

5. MÉTODO DE RESOLUÇÃO

O presente capítulo descreve em detalhe o modelo de programação linear inteira de Braga (2018) junto de ambos os métodos de separação, além dos algoritmos utilizados para sua implementação e integração.

5.1 Modelo de Programação Linear Inteira

O modelo de Braga se embasa na ideia de enraizar arborescências em cada vértice do grafo, as quais, quando sobrepostas, permitem o cálculo das distâncias entre os pares de vértices possíveis contidos no grafo. Dentre as formulações apresentadas em seu trabalho, foi escolhida a formulação "com rótulos", à qual dedica variáveis a essas distâncias e apresentou um desempenho melhor do que sua contrapartida (BRAGA, 2018).

Para gerar as arborescências de um grafo $G = (V, E)$, é necessário convertê-lo em um dígrafo $D = (V, A)$, tal que $A = \{(i, j), (j, i) \mid \forall ij \in E(G)\}$. Então criamos para cada vértice $v \in V(D)$ uma arborescência, por meio da variável binária z_{ij}^v que representa se a aresta $(i, j) \in A$ pertence à arborescência enraizada em v.

$$\sum_{\substack{ij \in A \\ j=k}} z_{ij}^v = 1 \qquad \forall v \in V,\ \forall k \in V \setminus \{v\} \tag{5.1}$$

$$\sum_{iv \in A} z_{iv}^v = 0 \qquad \forall vi \in V \tag{5.2}$$

$$x_e = z_{ij}^v + z_{ji}^v \qquad \forall e = ij \in A, \forall v \in V \tag{5.3}$$

$$z_{ij}^v \in \{0, 1\} \qquad \forall ij \in A, \forall v \in V$$

$$x_e \in \{0, 1\} \qquad \forall e = E$$

As restrições (5.1) e (5.2) são responsáveis, respectivamente, por assegurar que todo o vértice $i \neq v$ tenha grau de entrada igual a um e que não exista nenhum vértice que entre na raiz v. Isso garante que exista no máximo um caminho de um vértice a outro. Contudo, não é suficiente para garantir que as arestas escolhidas formem uma componente conexa ou que as arborescências possam ser sobrepostas. A restrição (5.3) então determina que, se uma aresta for escolhida, ela deve estar presente em todas as arborescências. Caso contrário, ela não faz parte da solução para o problema; ela também garante que todas as arborescências formam somente uma componente conexa, além da propriedade de sobreposição.

$$u_i^v - u_j^v + (k_{ij} + w_{ij})z_{ij}^v + (k_{ij} - w_{ij})z_{ji}^v \leq k_{ij} \qquad \forall ij \in A, \forall v \in V \qquad (5.4)$$

$$u_i^v + (k_{vi} + w_{vi})z_{vi}^v <= k_{vi} \qquad \forall ij \in A, \forall v \in V \qquad (5.5)$$

$$u_v^v = 0 \qquad \forall v \in V \qquad (5.6)$$

$$u_j^i = u_i^j \qquad \forall ij \in V \qquad (5.7)$$

$$u_j^i \leq t \cdot w_{ij} \qquad \forall ij \in V \qquad (5.8)$$

$$u_i^v \in \mathbb{R}^+ \qquad \forall vi \in V$$

Em relação às distâncias, declaramos a variável $u_i^v \in \mathbb{R}^+$ que representa a o peso do caminho da raiz v até o vértice i e a constante $k_{ij} = t * d_G(i, j)$ que representa o limite da distância para que a resposta respeite o fator de dilatação t. Ao unir as restrições (5.4), (5.5) e (5.6), é garantido que as arborescências aderem ao fator de dilatação. Em aderência as propriedades de sobreposição e de dilatação são impostas as restrições (5.7) e (5.8).

$$\min \sum_{e \in E} x_e w_e \qquad \forall e \in E$$

s.a

$$\sum_{e \in E} x_e = |V| - 1 \qquad \forall e \in E \qquad (5.9)$$

$$\sum_{\substack{ij \in A \\ j=k}} z_{ij}^v = 1 \qquad \forall v \in V,\ \forall k \in V \setminus \{v\}$$

$$\sum_{iv \in A} z_{iv}^v = 0 \qquad \forall ij \in A, \forall v \in V$$

$$x_e = z_{ij}^v + z_{ji}^v \qquad \forall e = ij \in A, \forall v \in V$$

$$u_i^v - u_j^v + (k_{ij} + w_{ij})z_{ij}^v + (k_{ij} - w_{ij})z_{ji}^v \leq k_{ij} \qquad \forall ij \in A, \forall v \in V$$

$$u_i^v + (k_{vi} + w_{vi})z_{vi}^v <= k_{vi} \qquad \forall ij \in A, \forall v \in V$$

$$u_v^v = 0 \qquad \forall v \in V$$

$$u_j^i = u_i^j \qquad \forall ij \in V$$

$$u_j^i \leq t \cdot w_{ij} \qquad \forall ij \in V$$

$$u_i^v \in \mathbb{R}^+ \qquad \forall vi \in V$$

$$z_{ij}^v \in \{0, 1\} \qquad \forall ij \in A, \forall v \in V$$

$$x_e \in \{0, 1\} \qquad \forall e = E$$

Como última etapa é necessário adicionar uma restrição para limitar as arestas e forçar a resposta final a ser acíclica – restrição (5.9), além de definir a função objetivo que se resume ao somatório do peso das arestas escolhidas. A união daquilo que foi apresentado representa a implementação do modelo com rótulos de Braga (2018).

5.2 Separação por Ponto de Articulação

Grafos que contêm pontos de articulação podem ser convertidos em uma árvore bloco-articulação, onde as componentes biconexas presentes no grafo original umas vez separadas por um vértice de corte formam os nodos da árvore. Essa propriedade é útil, visto que a separação ocorre sem perda da generalidade e permite a junção de árvores geradoras obtidas para cada bloco de maneira simples (SOUSA, 2021). Entretanto no contexto do problema da árvore *t-spanner* de peso mínimo essa separação não garante a propriedade do fator de dilatação, afinal se considera somente o fator de dilatação dos blocos separadamente. Em função disso, é necessário garantir se o grafo resultante após a junção é de fato *t-spanner*, o que durante os experimentos computacionais se demonstrou suficiente.

Existem diversos algoritmos que geram a árvore bloco-articulação de um determinado grafo, o mais comum na literatura é aquele demonstrado por Hopcroft e Tarjan (1973), o que além de calcular os pontos de articulação também provê os blocos. Repassa-se então as componentes para o modelo de Braga (2018), e aplica-se um algoritmo de distâncias mínimas na união dos resultados. Um pseudocódigo e apresentado no Algoritmo 1.

O algoritmo utiliza o tempo de descoberta dos vértices durante a busca por profundidade, mantendo um vetor do tempo de entrada e um vetor com o menor tempo dentre os próximos vértices na ordem da busca. Durante a busca os vértices visitados são armazenados em uma pilha, que ao se detectar um ponto de articulação, é esvaziada até tal. Os vértices removidos da pilha constituem uma componente biconexa, que é armazenada no conjunto de blocos. A detecção ocorre quando a busca chega a um vértice u, onde um dos seus vizinhos já marcados v tem seu tempo de entrada menor do que o tempo mínimo de entrada de v. Em outras palavras, isso significa que não é possível chegar a outro vértice fora da subárvore u sem passar por v no caminho, assim u é um ponto de articulação. Isso também se aplica no caso onde o grafo ou partes dele tenham a estrutura de uma árvore degenerada, onde todos os vértices exceto a raiz e os nodos folha são pontos de articulação.

Algoritmo 1: Algoritmo adaptado de Hopcroft e Tarjan

Entrada: Grafo conexo $G = (V, E)$; vetor booleano $apt[1 \ldots |V|]$; vértices u e p; tempo inicial t; pilha st; conjunto de componentes biconexas B

```
Function DFS(u, p):
    in_u ← low_u ← t + 1;
    t ← t + 1;
    st.push(u);
    foreach v ∈ N_G(u) do
        if v ≠ p then
            if in_v = 0 then
                DFS (v, u);
                low_u ← min(low_u, low_v);
                if low_v ≥ in_u then
                    apt_u ← (in_u > 1) ∨ (in_v > 2);
                    S ← ∅ ;
                    l ← ∞ ;
                    while l ≠ v do
                        l ← st.pop();
                        S ← S ∪ {l};
                    end
                    B ← B ∪ S;
                end
            end
        end
        else
            low_u ← min(low_u, low_v);
        end
    end
```

5.3 Separação por Clique Separador Balanceado

No contexto dos grafos cordais, existem propriedades que se assemelham ao descrito na seção anterior. A tentativa de separar o grafo pelo clique separador balanceado vem de uma junção da proposta de separador apresentada por Sousa (2021) e a heurística apresentada por Dragan e Köhler (2014).

Uma árvore de cliques, como o nome implica, transforma as cliques contidas dentro de um grafo em nodos, permitindo então o uso das propriedades de uma árvore no grafo. Todo grafo cordal pode ser convertido em uma árvore de cliques. Como descrito por Blair e Peyton (1993), é possível utilizar uma busca por cardinalidade máxima modificada para encontrar árvore de cliques geradora maximal, bem como a ordem de eliminação perfeita, o que já propicia o teste se o grafo é de fato cordal.

Utilizando a estrutura da árvore gerada pode-se encontrar um vértice central, o qual cumpra o papel de um clique separador balanceado. Um centroide é um vértice que, quando retirado da árvore, todas as novas subárvores geradas terão tamanho menor do que a metade da quantidade de vértices da árvore original. Para encontrar tal vértice utilizamos o algoritmo descrito por Giustina, Prezza e Venturini (2019). Quando aplicado a uma árvore de cliques é necessário modificar o algoritmo para considerar a quantidade de vértices do grafo original e não da árvore de cliques. Um pseudocódigo é apresentado no Algoritmo 2.

Algoritmo 2: Algoritmo para encontrar centroides

Entrada: Grafo conexo $G = (V, E)$; Árvore Geradora Máxima de Cliques $T = (D, F)$; vetor de cliques $q[1 \ldots |D|]$, vetor de inteiros auxiliar $dp[1 \ldots |D|]$; vértices x e p; conjunto de centroides C

Function `DFS_CENT`(x, p)**:**

- $dp_x \leftarrow |q_x|$;
- $b \leftarrow verdadeiro$;
- **foreach** $i \in N_T(x)$ **do**
  - **if** $i \neq p$ **then**
    - `DFS_CENT` (i, x);
    - $dp_x \leftarrow dp_x + dp_i$;
    - **if** $dp_i > \frac{|V|}{2}$ **then**
      - $b \leftarrow falso$;
    - **end**
  - **end**
- **end**
- **if** $|V| - dp_x > \frac{|V|}{2}$ **then**
  - $b \leftarrow falso$;
- **end**
- **if** b **then**
  - $C \leftarrow C \cup x$;
- **end**

É necessário pontuar que o método descrito não necessariamente encontra o clique separador balanceado para o grafo original, mas cumpre o papel de encontrar um candidato bom para a separação. Outro fator é a possibilidade de existirem dois centroides diferentes para uma árvore específica, para os propósitos do método de separação a escolha de um sobre o outro foi desconsiderada.

Após encontrar um centroide, é feita uma busca em profundidade a partir de cada vértice no grafo original, a qual é descrita pelo Algoritmo 3. Quando é encontrado um vértice que faz parte do separador a busca termina e os vértices visitados são declarados como um conjunto. Esses conjuntos então podem ter suas respostas calculadas por quaisquer métodos relevantes. No caso, foram realizados testes com o modelo de Braga (2018) e a árvore geradora de peso mínimo. A separação feita não garante que, ao se juntar as soluções parciais obtidas para

cada parte, obtenha-se uma solução ótima para o grafo como um todo. Por isso, é necessário repassar aquilo já calculado como solução inicial para o modelo e resolvê-lo novamente.

Algoritmo 3: Algoritmo de formação de blocos

Entrada: Grafo conexo $G = (V, E)$; conjunto de vértices Q, representando o clique separador balanceado; vetor booleano $m[1 \ldots |V|]$; vértice x; conjunto de vértices B representando o bloco atual

```
Function DFS_BLK(x):
    if m_x = falso then
        B ← B ∪ x;
        if x ∉ Q then
            m_x ← verdadeiro ;
            foreach i ∈ N_G(x) do
                DFS_BLK (i);
            end
        end
    end
```

6. RESULTADOS

Nesse capítulo serão apresentados os resultados obtidos com os experimentos computacionais, assim como uma análise dos mesmos. Além disso, as caraterísticas do ambiente de execução e os critérios de geração de instâncias para cada método de separação.

6.1 Instâncias

Para a realização dos experimentos computacionais foram utilizadas instâncias geradas aleatoriamente. De maneira similar à geração apresentada por Braga (2018) e Sousa (2021) foram utilizados os seguintes parâmetros: a quantidade de vértices de cada grafo n e a probabilidade geral p, que é utilizada de maneira variada dependendo do método de geração. Ao se tratar das distribuições de peso w das arestas, para cada aresta é dado um peso gerado aleatoriamente. É necessário pontuar que, para cada instância gerada, são atribuídos diferentes valores do fator de dilatação t, ou seja, para um mesmo grafo são testados vários fatores de dilatação diferentes. Além disso, são geradas 10 instâncias para cada combinação de n e p, o que equivale à 960 instâncias no total. Devido às diferenças de cada classe de grafo, o conjunto de valores de t muda dependendo do método de separação. Em relação ao tempo limite de execução, foi declarado como no máximo 3600 segundos. De forma mais específica os valores dos parâmetros utilizados são distribuídos em:

- $n \in \{25, 30, 40, 50, 60, 70\}$
- $p \in \{0.3, 0.5, 0.7, 0.8\}$
- $w \in [1, 1000]$
- Para os métodos de separação:
  - Por ponto de articulação: $t \in \{2, 3, 4, 10\}$
  - Por clique separador balanceado: $t \in \{3, 4, 10, 20\}$

Para instâncias do método de separação por pontos de articulação, usou-se a estratégia de gerar um grafo inicial com o seu tamanho variando de 30% a 80% do tamanho final do grafo. Em diferentes rodadas, os vértices remanescentes são selecionados aleatoriamente para gerar um grafo adicional, que é unido ao grafo inicial até que o tamanho esperado seja atingido. Dessa forma é mais fácil gerar grafos que tenham a estrutura desejada. O método *connected_watts_strogatz_graph* disponibilizado pela biblioteca *NetworkX*, foi utilizado para a geração do grafo inicial e dos grafos adicionais (HAGBERG; SCHULT; SWART, 2008). Vale

ressaltar que no caso das instâncias com pontos de articulação, foi estipulado que para cada instância, a mediana do tamanho dos blocos deve ser superior a 3. A razão disso é minimizar a quantidade de blocos de solução trivial, sem excluí-los totalmente.

Para as instâncias da divisão por clique separador balanceado, utilizou-se do método *extended_barabasi_albert_graph*, devido sua capacidade de gerar grafos densos ou esparsos dependendo da probabilidade fornecida, dessa forma permitindo maior variação entre a estrutura dos grafos gerados. Entretanto é necessário garantir que o grafo seja cordal e conexo. Para isso, foi aplicado o método *complete_to_chordal_graph*, o qual garante a cordalidade das instâncias. Caso o grafo resultante não fosse conexo, então o processo era repetido. Ambos métodos são disponibilizados pela biblioteca *NetworkX*.

6.2 Ambiente computacional

Para a resolução do modelo utilizamos o resolvedor matemático Gurobi (2025) - versão 12.0.3 e para implementação dos algoritmos a linguagem de programação C++, devido seu desempenho e sua extensiva biblioteca de estruturas de dados. Para gerar as instâncias foram criados programas na linguagem Python com a assistência da biblioteca *NetworkX* (HAGBERG; SCHULT; SWART, 2008). A máquina utilizada para a execução possui o processador Intel i7-8750H (12 cores) @ 4.100GHz e 16GB de mémoria RAM, assim como o sistema operacional Artix Linux - Kernel 6.17.7-artix1-1.

6.3 Separação por Pontos de Articulação

Nessa subseção são detalhados os resultados obtidos após a resolução das instâncias. Para cada instância foi registrado o tempo de execução para o modelo e para o método de separação, o valor encontrado, o número de blocos e pontos de articulação além o tamanho médio dos blocos. Além disso, a partir dos resultados foi calculada a diferença pareada, que calcula o percentual da diferença entre o tempo para o modelo matemático e o método de separação. Vale ressaltar que para todas as 960 diferentes execuções, os valores encontrados pelo modelo de programação linear inteira e o método de separação foi o mesmo, assim não foram incluídos nas métricas.

	Modelo Original		Separação por ponto de articulação		
N.º de Vértices	Tempo médio	Tempo total	Tempo médio	Tempo total	Melhoria Percentual (Tempo Total)
25	0,23	36,16	0,07	11,79	67,39%
30	0,58	93,45	0,23	36,10	61,37%
40	3,14	502,01	0,92	147,50	70,62%
50	4,34	695,19	0,07	165,65	76,17%
60	22,47	3595,48	1,04	694,56	80,68%
70	35,67	5706,81	4,34	1025,40	82,03%
Total:	11,07	10629,09	2,17	2081,00	80,42%

Tabela 6.1: Tempos de execução para o modelo original e para o método de separação em relação ao numero de vértices, assim como a melhora relativa ao tempo total

	Modelo Original		Separação por ponto de articulação		
N.º de Blocos	Tempo médio	Tempo total	Tempo médio	Tempo total	Melhoria Percentual
3	12,46	10203,20	2,37	1943,89	80,95%
4	2,96	284,52	1,14	109,18	61,63%
5	3,17	88,84	0,60	16,72	81,19%
6+	3,28	52,48	0,70	11,17	78,71%

Tabela 6.2: Relação entre o número de blocos e a melhoria no tempo de execução

	Modelo Original		Separação por ponto de articulação		
Tamanho Médio dos Blocos	Tempo médio	Tempo total	Tempo médio	Tempo total	Melhoria Percentual
[3, 10)	0,85	185,69	0,24	53,28	71,30%
[10, 14)	1,92	568,93	0,55	164,03	71,17%
[14, 20)	5,62	764,06	1,34	182,29	76,14%
[20, 24)	29,58	9110,36	5,46	1681,36	81,54%

Tabela 6.3: Relação entre o tamanho médio dos blocos e a melhoria no tempo de execução

Como é observado na Tabela 6.1, percebe-se que, conforme o tamanho das instâncias cresce, a melhoria em relação ao modelo original também aumenta, culminando em cerca de 80% ao compararmos o tempo total para todas as instâncias. Esse comportamento também é visto na Tabela 6.3, afinal o tamanho médio dos blocos esta diretamente relacionado ao tamanho da instância. Em contrapartida, as informações apresentadas pela Tabela 6.2, não aparentam nenhuma correlação relevante dentre o tempo de execução dos diferentes métodos e a quantidade dos blocos ao olharmos para a porcentagem de melhoria.

Como é evidenciado na Figura 6.1 nem sempre a separação resulta em ganho de desempenho previsível, na verdade é possível identificar casos em que ocorrem perdas atípicas no desempenho. Acredita-se que isso ocorre devido a elevada variância dentre o tempo de execução para o método original e o de separação. Um exemplo disso é a instância *i463_40_0.8_4.txt*

cujo tempo de execução aumentou de 0,19 segundos no modelo original para 38,86 segundos quando aplicado o método de separação.

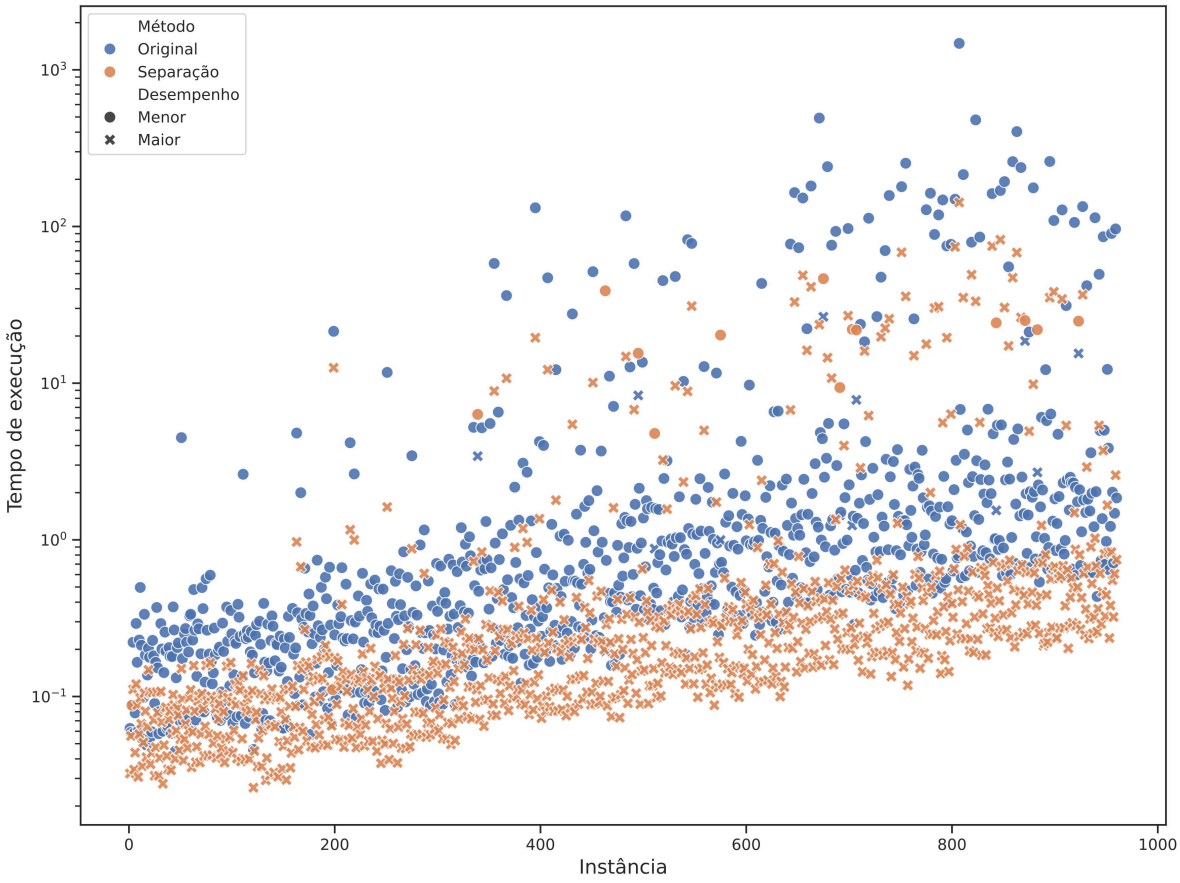

Figura 6.1: Tempos de execução de cada instância para o método original e a separação. O estilo dos pontos indica qual algoritmo apresentou melhor desempenho.

Uma provável explicação para esse comportamento é que, quando o modelo é executado para o grafo em sua totalidade existem maiores chances de poda das respostas subótimas e inviáveis. Dessa forma, o modelo tende a checar um maior conjunto de possibilidades ocasionando e com isso maior tempo de execução. Entretanto, a relação inversa também se aplica, já que ao separarmos o grafo, diminuímos a região viável bem como o tempo de execução. Em outras palavras, acredita-se que a diferença entre os tempos advém do quanto cada instância é afetada por essas propriedades.

Os dados apresentados sugerem que, além do fatores relacionados ao número de vértices, os demais fatores estruturais do grafo têm pouca influência no desempenho do método de separação. Uma vez que não se observa uma tendência clara ao comparar o desempenho em função desses. Torna-se mais adequado analisar, de forma geral, o tempo de execução de cada método frente ao conjunto de instâncias.

Ademais, como o apresentado no perfil de desempenho presente na Figura 6.2, ambos os métodos apresentam crescimento exponencial de tempo em relação ao número de vértices

das instâncias. Mas o crescimento é muito mais lento para à separação, em especial para os casos maiores, demonstrando ser um método robusto para tal circunstância. A partir disso, é possível afirmar que aplicar o método de separação por ponto de articulação ocasiona em notável melhoria de desempenho.

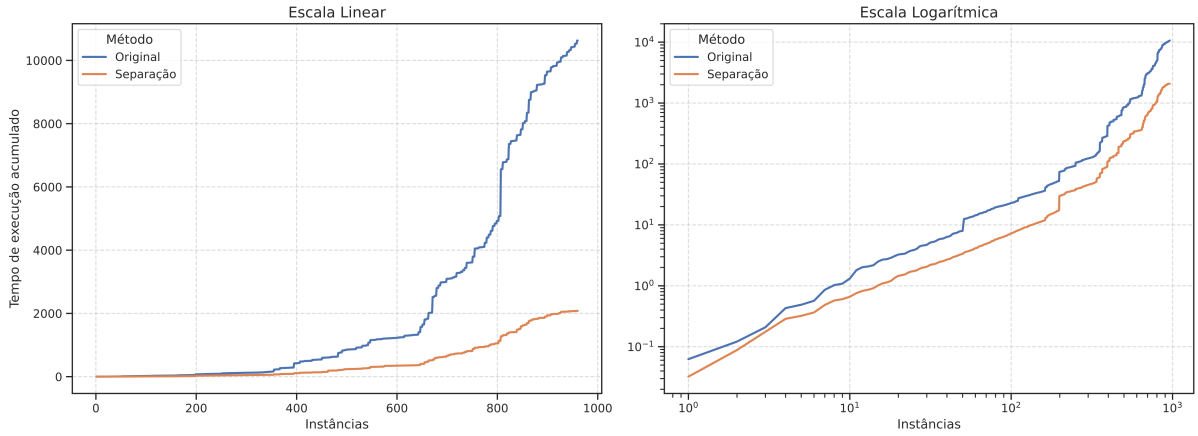

Figura 6.2: Tempo acumulado após a execução de cada instância

6.3.1 Separação por Clique Separador Balanceado

Nessa subseção são apresentados os resultados obtidos para as instâncias cordais. Para tal, foram realizados os testes com o modelo original de Braga (2018), e para o método de separação, foi utilizado tanto o modelo original quanto a *MST* para o cálculo das respostas parciais. Diferentemente da separação por ponto de articulação, devido as características do método cordal, não foi realizada a captura de demais dados estruturais além em das informações básicas das instâncias, resumindo-as ao tempo de execução para cada variação do método de resolução.

	Modelo Original		Separação/Modelo		
N.º de Vértices	Tempo médio	Tempo total	Tempo médio	Tempo total	Diferença Percentual
25	0,54	86,61	0,74	117,72	-35,93%
30	1,60	256,00	2,50	399,60	-56,10%
40	9,85	1575,43	14,81	2369,96	-50,43%
50	18,25	2920,21	26,97	4322,47	-48,02%
60	46,79	7486,14	82,00	13119,84	-75,25%
70	109,78	17565,48	172,51	75168,29	-327,93%
Total:	31,14	29889,86	49,74	95497,88	-219,50%

Tabela 6.4: Tempos de execução para o modelo original e o método por clique separador balanceado utilizando o modelo para calcular as respostas parciais em relação ao número de vértices, assim como a diferença relativa ao tempo total

	Modelo Original		Separação/MST		
N.º de Vértices	Tempo médio	Tempo total	Tempo médio	Tempo total	Diferença Percentual
25	0,54	86,61	0,52	83,80	3,24%
30	1,60	256,00	1,57	250,96	1,97%
40	9,85	1575,43	10,39	1662,26	-5,51%
50	18,25	2920,21	18,70	2992,24	-2,47%
60	46,79	7486,14	47,47	7594,48	-1,45%
70	109,78	17565,48	124,11	19857,23	-13,05%
Total:	31,14	29889,86	33,79	32440,98	-8,54%

Tabela 6.5: Tempos de execução para o modelo original e para o método por clique separador balanceado utilizando a MST para calcular as respostas parciais em relação ao número de vértices, assim como a diferença relativa ao tempo total

Como é demonstrado nas Tabelas 6.4 e 6.5, o método em abas suas variações teve pior desempenho que o modelo de Braga (2018) no geral. Em contra partida aos resultados expostos na subseção anterior, o método de separação apresentou perda significa de desempenho para as instâncias com maior numero de vértices. Isso ocorreu principalmente quando a separação utilizou o modelo para o cálculo das respostas parciais, o que ocasionou em um aumento muito além do esperado ao tempo, o que não ocorreu com a resolução por heurística, que apresentou melhora marginal no tempo de execução para casos com poucas vértices. O desempenho dos modelos pode ser averiguado na Figura 6.3.

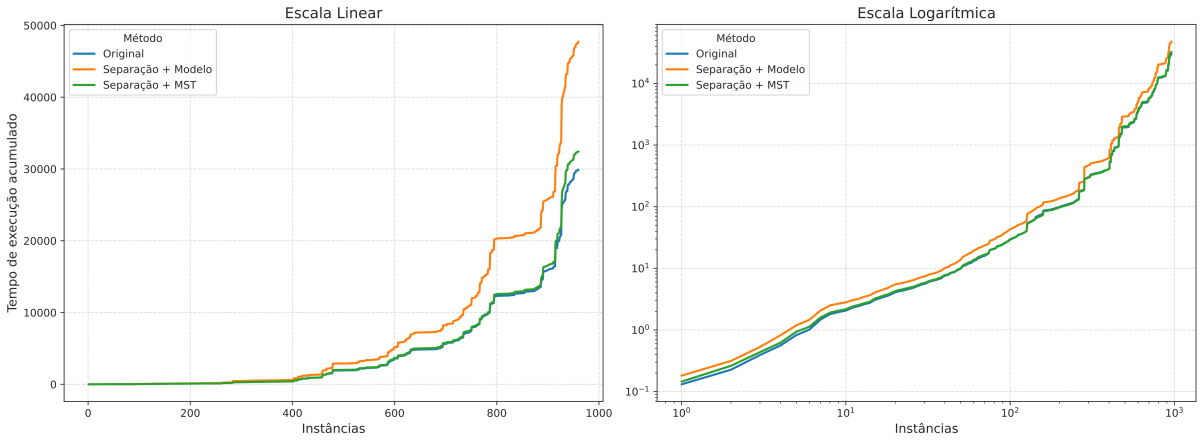

Figura 6.3: Tempo acumulado após a execução de cada instância

A provável explicação para tal comportamento, é que ao calcularmos a resposta local ótima, existe pouco aproveitamento das arestas escolhidas pelo modelo na resposta final, além do custo de tempo adicional devido sua utilização. O que também engaja com o fato da resposta aproximada ter tido melhor desempenho nos casos com tamanho menor, afinal existe menor quantidade de escolhas dentre as arestas, e então fica mais fácil a resposta final se assemelhar à heurística. Isso levanta a questão sobre a viabilidade de uma heurística capaz de oferecer bom

aproveitamento tanto para a resposta local quanto a global, ou se existe outra característica dos grafos cordais que permitam o cálculo da resposta de maneira mais rápida quando integradas ao modelo.

7. CONCLUSÃO

Nesse trabalho foram abordados dois diferentes métodos de separação de grafos no contexto do problema da árvore *t-spanner* de peso mínimo com o objetivo de averiguar seus efeitos no desempenho do modelo de programação linear inteira de Braga (2018).

Pela análise dos resultados, podemos afirmar que o método por ponto de articulação se destacou com ampla melhoria de desempenho. Apesar das inconsistências aparentes para alguns casos específicos, levando em consideração o contexto geral das instâncias, apresentou redução em média de cerca de 80% tempo de execução. Além disso, se demonstrou robusto frente a grafos com maior número de vértices. O método por clique separador balanceado, por sua vez apresentou uma piora geral para o tempo de execução. Entretanto existem diferentes formas de abordar a classe dos grafos cordais. O uso de heurísticas para o caso cordal demonstrou-se mais vantajoso do que o uso puro do modelo, como implementado no método por ponto de articulação. Até onde se sabe, utilizar os centroides de uma árvore de cliques para determinar bons candidatos para separadores balanceados é algo que ainda não foi devidamente estudado na literatura. Existe chance desse método levar a uma resposta ótima ou ser uma boa heurística para encontrar separadores.

Algo que ainda não foi averiguado é o impacto do uso de diferentes heurísticas para calcular respostas boas para os nós da árvore geradora de cliques maximal e repassar esses então ao modelo, além disso pode-se utilizar de outras estruturas como um grafo de cliques para conter diferentes partes do grafo. Também não foi testado utilizar a ordem de eliminação perfeita presente nos grafos cordais para gerar uma resposta ótima gradualmente. Devido ao bom resultado do método por pontos de articulação, é possível que haja similar ganho de desempenho para outras classes de grafos. Tais possibilidades são deixadas como trabalho futuro.

REFERÊNCIAS BIBLIOGRÁFICAS

ALTHÖFER, I. et al. On sparse spanners of weighted graphs. *Discrete Computational Geometry*, Springer New York, v. 9, n. 1, p. 81–100, dez. 1993.

BLAIR, J. R. S.; PEYTON, B. An introduction to chordal graphs and clique trees. In: GEORGE, A.; GILBERT, J. R.; LIU, J. W. H. (Ed.). *Graph Theory and Sparse Matrix Computation*. New York, NY: Springer New York, 1993. p. 1–29.

BRAGA, H. V. V. *Algoritmos exatos para problemas de spanner em grafos*. Tese (Dissertação de Doutorado) — Universidade de São Paulo, 2018.

CHEW, P. There is a planar graph almost as good as the complete graph. In: *Proceedings of the second annual symposium on Computational geometry - SCG '86*. Yorktown Heights, New York, United States: ACM Press, 1986. p. 169–177.

CHIMANI, M.; STUTZENSTEIN, F. *Spanner Approximations in Practice*. arXiv, 2021. ArXiv:2107.02018 [cs]. Disponível em: <http://arxiv.org/abs/2107.02018>.

DEMMER, M. J.; HERLIHY, M. P. The arrow distributed directory protocol. In: GOOS, G. et al. (Ed.). *Distributed Computing*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1998. v. 1499, p. 119–133.

DIESTEL, R. *Graph Theory*. 5th ed.. ed. Berlin: Springer, 2018. (Graduate Texts in Mathematics, 173).

DRAGAN, F. F.; KöHLER, E. An Approximation Algorithm for the Tree t-Spanner Problem on Unweighted Graphs via Generalized Chordal Graphs. *Algorithmica*, v. 69, n. 4, p. 884–905, ago. 2014.

GILBERT, J. R.; ROSE, D. J.; EDENBRANDT, A. A Separator Theorem for Chordal Graphs. *SIAM Journal on Algebraic Discrete Methods*, v. 5, n. 3, p. 306–313, set. 1984.

GIUSTINA, D. D.; PREZZA, N.; VENTURINI, R. A new linear-time algorithm for centroid decomposition. In: BRISABOA, N. R.; PUGLISI, S. J. (Ed.). *String Processing and Information Retrieval*. Cham: Springer International Publishing, 2019. p. 274–282.

Gurobi. *The Leader in Decision Intelligence Technology*. 2025. Acesso em: 15 de novembro de 2025. Disponível em: <https://www.gurobi.com/>.

HAGBERG, A. A.; SCHULT, D. A.; SWART, P. J. Exploring network structure, dynamics, and function using networkx. In: VAROQUAUX, G.; VAUGHT, T.; MILLMAN, J. (Ed.). *Proceedings of the 7th Python in Science Conference*. Pasadena: [s.n.], 2008. p. 11 – 15.

HOPCROFT, J.; TARJAN, R. Algorithm 447: efficient algorithms for graph manipulation. *Communications of the ACM*, v. 16, n. 6, p. 372–378, jun. 1973.

JUNGNICKEL, D. *Graphs, Networks and Algorithms*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013. v. 5. (Algorithms and Computation in Mathematics, v. 5).

MIRANDA, E. Álvarez; SINNL, M. Mixed-integer programming approaches for the tree t*-spanner problem. *Optimization Letters*, v. 13, n. 7, p. 1693–1709, 2019.

NEMHAUSER, G. L.; WOLSEY, L. A. *Integer and combinatorial optimization*. New York, NY Weinheim: Wiley, 1999. (Wiley-Interscience series in discrete mathematics and optimization).

PELEG, D. Low Stretch Spanning Trees. In: GOOS, G. et al. (Ed.). *Mathematical Foundations of Computer Science 2002*. Berlin, Heidelberg: Springer, 2002. v. 2420, p. 68–80.

PELEG, D.; SCHäFFER, A. A. Graph spanners. *Journal of Graph Theory*, v. 13, n. 1, p. 99–116, mar. 1989.

SOUSA, G. H. d. *Problema da árvore t-spanner de custo mínimo*. Dissertação (Mestrado) — Universidade Federal do Ceará, 2021.

VASCONCELLOS, E. A. S.; MIYAZAWA, F. K. *Solução para o problema da Árvore t-spanner de peso mínimo com programação linear inteira*. [S.l.], 2022.

WOLSEY, L. A. *Integer programming*. New York: Wiley, 1998. 1–7 p. (Wiley-Interscience series in discrete mathematics and optimization).