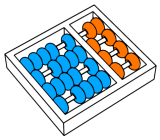


Solução para o problema da árvore t-spanner de peso mínimo com programação linear inteira

E. A. S. Vasconcellos *F. K. Miyazawa*

Relatório Técnico - IC-PFG-22-01

Projeto Final de Graduação

2022 - Agosto

UNIVERSIDADE ESTADUAL DE CAMPINAS
INSTITUTO DE COMPUTAÇÃO

Solução para o problema da árvore t-spanner de peso mínimo com programação linear inteira

Eduardo Augusto Simão Vasconcellos* Flávio Keidi Miyazawa*

Resumo

Situações que envolvem necessidade de economia de recursos ou computação de forma eficiente sobre caminhos em determinada rede ocorrem com frequência, e a forma de abordar esse tipo de problema pode ser crucial para obter resultados satisfatórios. É do estudo desses cenários que nascem problemas de minimização em spanners. Este trabalho descreve a implementação de um algoritmo de solução exata para um problema particular sobre spanners: o problema da árvore t-spanner de peso mínimo. O desafio em questão é encontrar, dado um grafo G e um inteiro t, uma árvore geradora de G na qual a distância entre cada par de vértices seja no máximo t vezes a distâncias entre eles em G. Os resultados da implementação e melhorias através de heurísticas são então discutidas e analisadas.

1 Introdução

Não são incomuns contextos nos quais se deseja economizar recursos ou computar caminhos ou rotas de forma rápida e eficiente. Um exemplo de destaque é o roteamento em redes de comunicação, mais especificamente quando se tem alguma restrição de velocidade ou espaço de armazenamento [1]. Uma das possíveis abordagens para o problema consiste em encarar a rede como um grafo e tentar construir um spanner desse grafo, obtendo uma sub-rede com significativamente menos conexões que a original, mas sem que as rotas entre cada par de pontos na sub-rede tenham uma distância muito maior do que na rede inicial. Esta seção define os conceitos de grafos necessários para a compreensão do trabalho e apresenta uma breve contextualização do problema da árvore t-spanner de peso mínimo.

1.1 Fundamentos de grafos

Um *grafo* G consiste em um par ordenado $(V(G), E(G))$ em que $V(G)$ é o seu conjunto de *vértices* e $E(G)$ o seu conjunto de *arestas*. Cada aresta de G está associada, por uma *função de incidência* ψ, a um par não ordenado de vértices, que são os seus *extremos*. Neste texto, representamos a aresta e tal que $\psi(e) = \{u, v\}$ por $e = uv$, deixando a função de incidência implícita. Um *digrafo* D – também conhecido como *grafo direcionado* – é um grafo em que as arestas possuem direção. As arestas de um digrafo são usualmente chamadas de *arcos*,

*Instituto de Computação, Universidade Estadual de Campinas, 13083-852 Campinas, SP

enquanto seu conjunto de arcos, de $A(D)$. Representaremos um arco direcionado de u a v por $a = uv$, sendo u e v a *cabeça* e a *cauda* de a, respectivamente. Ao remover a orientação dos arcos de um digrafo D, obtemos seu *grafo subjacente*. A fim de evitar redundâncias, consideramos no restante desta seção que todas as definições apresentadas para grafos valem também em digrafos.

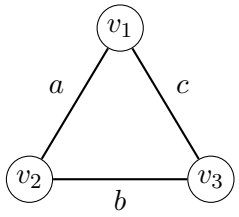


Figura 1: Grafo com $V(G) := \{v_1, v_2, v_3\}$ e $E(G) := \{a, b, c\}$. Na notação com função de incidência implícita: $v_1v_2 = a$; $v_2v_3 = b$; e $v_1v_3 = c$.

Seja G um grafo. Dois vértices $u, v \in V(G)$ são *adjacentes* se existe $e \in E(G)$ cujos extremos são u e v, isto é, se existe $e = uv$. Duas arestas são *adjacentes* se possuem um extremo em comum. Em um grafo qualquer, o *grau* de um vértice v, denotado por $\delta(v)$, é dado pelo número de vezes em que ele é extremo de arestas. Em digrafos, o *grau de entrada* $\delta^-(v)$ de um vértice v é o número de vezes em que ele é cabeça de arcos. Analogamente, seu *grau de saída* $\delta^+(v)$ é dado pelo número de vezes em que é cauda de arcos.

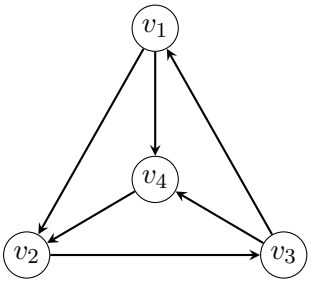


Figura 2: Digrafo em que $\delta^-(v_4) = 2$ e $\delta^+(v_4) = 1$.

Um *passeio* de v_1 a v_n em um grafo G é dado por uma sequência $v_1, v_2, \ldots, v_n$ de vértices de G de forma que $v_iv_{i+1} \in E(G)$ para todo $1 \leq i \leq n-1$. Se os vértices forem todos distintos, o passeio é denominado *caminho*. Se $v_nv_1 \in E(G)$ para algum caminho $v_1, v_2, \ldots, v_n$, então $v_1, v_2, \ldots, v_n, v_1$ é um *ciclo* em G. Seja $\mathcal{C}$ o conjunto de todos caminhos de u a v em G. A *distância* $d(u, v)$ entre u e v é definida como o número de arestas do caminho de $\mathcal{C}$ com menor número de arestas.

Grafos nos quais existe pelo menos um caminho entre todos pares de vértices são ditos *conexos*, enquanto grafos nos quais existe exatamente um caminho entre todos os pares de vértices são chamados de *árvores*. Pode ser conveniente evidenciar um vértice v da árvore, chamando-o de *raiz*; nesse caso dizemos que o grafo é uma *árvore enraizada em* v. Em digrafos, como os caminhos são unilaterais, costuma-se fazer mais sentido considerar

arborescências, que são árvores enraizadas nas quais todos os arcos apontam "para fora" da raiz.

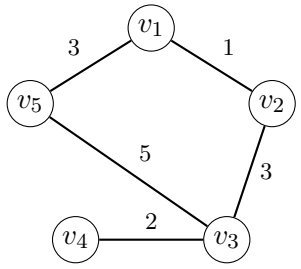


Figura 3: Grafo ponderado em que $P = v_1, v_2, v_3, v_4$ é um caminho, $C = v_1, v_2, v_3, v_5, v_1$ é um ciclo, $d(v_1, v_3) = 4$, e $d(v_3, v_5) = 5$.

Às vezes, convém associar valores, conhecidos como *pesos*, às arestas de um grafo. Grafos cujas arestas possuem pesos são denominados *grafos ponderados* – ou *digrafos ponderados*, no caso de arcos com pesos. Representamos grafos ponderados como um par (G, w), no qual $G = (V, E)$ é o grafo propriamente dito, e w é uma *função peso* com domínio igual ao conjunto E. Quando trabalhamos com grafos ponderados, geralmente consideramos a distância entre dois vértices não mais a quantidade de arestas do caminho com menor número de arestas, mas sim a soma dos pesos das arestas de um caminho com peso mínimo.

Um *subgrafo* H de um grafo G, $H \subseteq G$, é um grafo tal que $V(H) \subseteq V(G)$, $E(H) \subseteq E(G)$, e ψ_H é restrição de ψ_G. Subgrafos com $V(H) = V(G)$ são chamados de *subgrafos geradores*. Grande parte dos problemas estudados em grafos consistem em encontrar um subgrafo que possui determinadas características; é o caso do problema estudado neste trabalho.

1.2 Árvore t-spanner de peso mínimo

Seja (G, w, t) uma tripla formada por um grafo ponderado $G = (V, E)$, sua função peso $w : E \to \mathbb{R}_{\geq 0}$ e um *fator de dilatação* $t \geq 1$, $t \in \mathbb{R}$. Um *t-spanner* de G é um subgrafo gerador $H \subseteq G$ tal que, para qualquer par $u, v \in V(G)$, $d(u, v)$ em H é no máximo t vezes $d(u, v)$ em G. Se H é árvore, o chamamos de *árvore t-spanner*. Um dos principais temas de pesquisa quando se trata de spanners é o *problema do t-spanner de peso mínimo*; o foco do estudo deste trabalho é uma restrição dele: o *problema da árvore t-spanner de peso mínimo* – referido daqui pra frente como PATSPM –, que visa encontrar, dada uma tripla (G, w, t), uma árvore t-spanner de peso mínimo (supondo que G admita uma árvore t-spanner). Em 1995, Cai e Corneil provaram que esse problema é NP-difícil [2].

O conceito de spanner surgiu em 1987, em um estudo sobre algoritmos assíncronas em sistemas distribuídos de Peleg e Ullman [3, 4], mas o termo "spanner" só veio a ser utilizado pela primeira vez dois anos depois, em artigo de Peleg e Schäffer [5] que mostra, dentre outros resultados, que determinar se um grafo possui um 2-spanner com M ou menos arestas, $M \geq 1$, é NP-completo.

Os estudos sobre spanners são frequentemente pautados na busca por spanners *esparsos*,

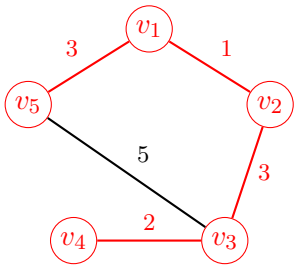


Figura 4: Em destaque, uma árvore 2-spanner de peso mínimo do grafo da Figura 3.

isto é, spanners com poucas arestas – ou pouco peso total, em grafos ponderados –; daí que surge, naturalmente, o interesse em árvores spanners. Movido por esse interesse, Cai provou diversos resultados sobre a complexidade de problemas envolvendo spanners esparsos, como os do t-spanner de peso mínimo e da árvore t-spanner de peso mínimo [6, 7]. Ainda na década de 90, Althöfer et al. desenvolveram um algoritmo guloso para obter spanners esparsos em grafos ponderados arbitrários [8].

Entre o final da década de 90 e 2010, diversos resultados para problemas sobre spanners foram obtidos. Um desses resultados de particular relevância foi a descoberta de limites inferiores e superiores para a quantidade de arestas de um 3-spanner mínimo, provados por Duckworth e Zito [9]. Outros autores preferiram focar em classes específicas de grafos, estabelecendo tempos polinomiais para encontrar t-spanners em alguns casos restritos [10, 11, 12]. Nesse mesmo período, foram publicados também trabalhos que utilizam programação linear para obter soluções aproximadas para problemas de t-spanners, como o de Berman et al. em 2013, que apresenta um algoritmo de aproximação com complexidade $O(\sqrt{n}\log n)$ para o problema do t-spanner de peso mínimo em digrafos [13].

Este trabalho se utiliza de programação linear inteira na obtenção de uma solução exata para o problema da árvore t-spanner de peso mínimo, conforme modelagem proposta por Braga em 2019 [14].

2 Objetivos

Esse trabalho tem como objetivo estudar e implementar uma solução exata para o problema da árvore t-spanner de peso mínimo, utilizando programação linear. O desempenho do programa foi avaliado para diferentes tipos e tamanhos de instância. Além disso, foram estudadas possibilidades de heurísticas a fim de melhorar o tempo de execução do algoritmo, bem como o impacto de sua aplicação.

3 Formulação linear inteira para o PATSPM

A formulação proposta por Braga se baseia em arborescências construídas a partir de cada vértice do grafo original. Estabelecemos restrições que forçam com que o grafo subjacente de todas as arborescências seja o mesmo, de forma que a sobreposição de duas arborescências

evidencie o caminho entre as raizes das duas. O primeiro passo é, dado um grafo G, considerar um digrafo D tal que $V(D) = V(G)$ e $A(D) = \bigcup_{uv \in E(G)} \{uv, vu\}$, como exemplificado na Figura 5. Construímos uma arborescência enraizada em cada vértice de D. Para isso, para cada $v \in V(D)$, consideramos uma variável binária y_{ij}^v para cada $ij \in A(D)$; essa variável indica se a aresta ij faz parte da arborescência enraizada em v – valor um caso fizer, zero caso contrário. Utilizamos então as seguintes restrições:

$$\sum_{i \in \delta^-(j)} y_{ij}^v = 1 \qquad \forall v \in V(D), \quad \forall j \in V(D) \setminus \{v\} \tag{1}$$

$$\sum_{i \in \delta^-(v)} y_{iv}^v = 0 \qquad \forall v \in V(D) \tag{2}$$

$$y_{ij}^v \in \{0, 1\} \qquad \forall v \in V(D), \quad \forall ij \in A(D) \tag{3}$$

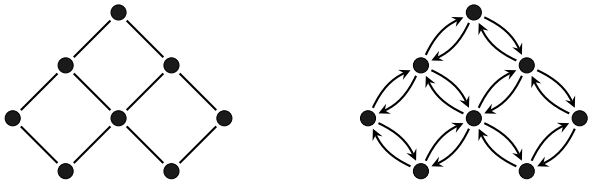


Figura 5: Grafo G original e digrafo D associado.

Aa restrições (1) forçam que todos vértices com exceção da raiz possuam grau de entrada um, enquanto as restrições (2) fazem com que a raiz tenha grau de entrada zero. Seja $H^v \subseteq D$ subgrafo gerador de D tal que $A(H^v) = \{ij \mid y_{ij}^v = 1\}$. As restrições (1) e (2) obrigam H^v a ter uma das duas formas exemplificadas na Figura 6.

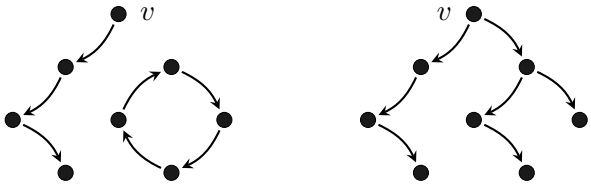


Figura 6: Possíveis formatos de H^v.

Note que as restrições (1) e (2) garantem que v sempre pertencerá a uma componente conexa de H^v, mas não que H^v seja em si conexo. Em resumo, H possui k componentes conexas, sendo elas 1 árvore e $k-1$ ciclos. Portanto, esse conjunto de restrições ainda não é suficiente para construir uma arborescência para cada vértice. Isso é resolvido pelas restrições (4), que introduzem também novas variáveis binárias x_e, uma para cada aresta de $E(G)$; x_e vale um caso e pertença à solução do problema da árvore t-spanner de peso mínimo, e zero caso contrário.

$$x_e = y_{ij}^v + y_{ji}^v \qquad \forall v \in V(D), \quad \forall e = ij \in E(G) \tag{4}$$

$$x_e \in \{0, 1\} \qquad \forall e \in E(G) \tag{5}$$

Para entender o efeito dessas novas restrições, consideremos uma solução $(\tilde{x}, \tilde{y})$ do sistema proposto até aqui. Seja $H^v \subseteq D$ tal que $A(H^v) = \{ij \mid y^v_{ij} = 1\}$, e $\widetilde{H}^v \subseteq G$ o grafo subjacente de H^v. De imediato, podemos depreender do conjunto de restrições (4) a seguinte relação:

$$\widetilde{H}^v = \widetilde{H}^u, \quad \forall v, u \in V(D)$$

Queremos mostrar que, para todo $v \in V(D)$, H^v é uma arborescência enraizada em v, isto é, H^v tem forma semelhante ao segundo exemplo da Figura 6. Suponhamos que isso não seja verdade. Seja então u tal que H^u possui algum ciclo C. Note que C existe também em $\widetilde{H}^v$. Sabemos que u não pertence a C, caso contrário a restrição (2) seria violada. Seja então $w \neq u$ tal que $w \in V(C)$. Por raciocínio análogo ao que aplicamos para u, w não pode pertencer a nenhum ciclo de H^w, e portanto também não pertence a nenhum ciclo em $\widetilde{H}^w$. Mas $\widetilde{H}^u = \widetilde{H}^w$, então w não pertence a ciclos em $\widetilde{H}^u$ e H^u, de forma que chegamos a uma contradição. Concluímos que H^v é uma arborescência enraizada em v para todo $v \in V(D)$. A fim de simplificação da notação, a partir deste ponto chamaremos $\widetilde{H}^v$ por H.

Como mencionado no início da seção, o objetivo final da formulação é ajudar a evidenciar as distâncias entre cada par de vértices. Podemos observar esse efeito na Figura 7. Os únicos arcos que possuem orientação diferente em H^v e H^w são as que pertencem ao caminho entre v e w na solução – em H^v esses arcos formam um caminho direcionado de v a w, enquanto em H^w, de w a v.

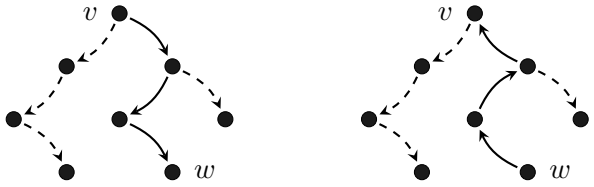


Figura 7: Sobreposição de H^v e H^w.

Com isso, estamos prontos para montar as restrições finais do modelo. Definiremos, para cada aresta $f = uv \in E(G)$, um novo conjunto de variáveis binárias z^f_e para toda aresta $e \in E(G)$. A ideia é que $z^f_e = 1$ indica que a aresta e pertence ao caminho entre u e v em H, enquanto $z^f_e = 0$ indica que a aresta não pertence a esse caminho. Note que aqui consideramos apenas os caminhos entre vértices adjacentes, visto que se as distâncias entre vértices adjacentes no grafo original respeitar o fator de dilatação, então naturalmente as distâncias entre vértices não adjacentes também o respeitarão.

Introduzimos agora duas definições pensadas com o intuito de ajudar a distinguir se a aresta ij pertence ao caminho de u a v em H.

$$d^{uv}_{ij} := y^u_{ij} - y^v_{ij}$$
$$s^{uv}_{ij} := y^u_{ij} + y^v_{ij}$$

Os valores de d^{uv}_{ij} e s^{uv}_{ij} correspondentes às possíveis combinações de y^u_{ij} e y^v_{ij} estão expostos na Tabela 1.

y_{ij}^u	y_{ij}^v	d_{ij}^{uv}	s_{ij}^{uv}	y_{ji}^u	y_{ji}^v	d_{ji}^{uv}	s_{ji}^{uv}
1	0	1	1	0	1	-1	1
0	1	-1	1	1	0	1	1
1	1	0	2	0	0	0	0
0	0	0	0	1	1	0	2
0	0	0	0	0	0	0	0

Tabela 1: Possíveis valores para d_{ij}^{uv}, s_{ij}^{uv}, d_{ji}^{uv} e s_{ji}^{uv}.

As quatro primeiras linhas representam os casos em que $ij \in E(H)$, sendo as duas primeiras os casos em que ij pertence ao caminho de u a v em H – lembre que os arcos possuem sentidos opostos em H^u e H^v se e somente se pertencem a esse caminho. Com isso em mente, considere as inequações abaixo:

$$d_{ij}^{uv} \leq z_e^{uv} \leq s_{ij}^{uv} \qquad \forall uv \in E(G), \quad \forall e = ij \in E(G) \tag{6}$$

$$d_{ji}^{uv} \leq z_e^{uv} \leq s_{ji}^{uv} \qquad \forall uv \in E(G), \quad \forall e = ij \in E(G) \tag{7}$$

Da Tabela 1 e das inequeções (6) e (7), $z_e^{uv} = 1$ se e somente se a aresta e pertence ao caminho entre u e v em H. Assim, podemos representar $d_H(u, v)$ por:

$$d_H(u, v) = \sum_{e \in E(G)} w_e z_e^{uv}$$

Com isso, falta somente estabelecermos uma restrição para limitar $d_H(u, v)$ de acordo com o fator de dilatação e uma função objetivo a ser minimizada, que nada mais é do que a somatória dos pesos das arestas de H. A formulação completa está exposta abaixo.

$$min \sum_{e \in E(G)} w_e x_e$$

$$\sum_{i \in \delta^-(j)} y_{ij}^v = 1 \qquad \forall v \in V(G), \quad \forall j \in V(G) \setminus \{v\}$$

$$\sum_{i \in \delta^-(v)} y_{iv}^v = 0 \qquad \forall v \in V(G)$$

$$x_e = y_{ij}^v + y_{ji}^v \qquad \forall v \in V(G), \quad \forall e = ij \in E(G)$$

$$y_{ij}^u - y_{ij}^v \leq z_e^{uv} \leq y_{ij}^u + y_{ij}^v \qquad \forall uv \in E(G), \quad \forall e = ij \in E(G)$$

$$y_{ji}^u - y_{ji}^v \leq z_e^{uv} \leq y_{ji}^u + y_{ji}^v \qquad \forall uv \in E(G), \quad \forall e = ij \in E(G)$$

$$\sum_{e \in E(G)} w_e z_e^{uv} \leq t \cdot w_{uv} \qquad \forall uv \in E(G)$$

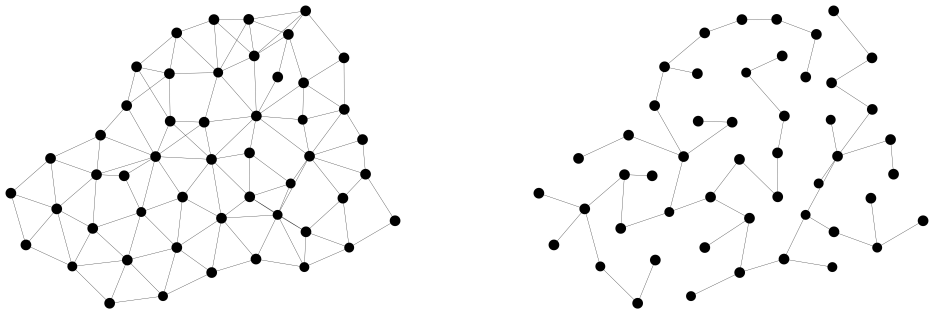
$$x_e \in \{0, 1\} \qquad \forall e \in E(G)$$

$$y_{ij}^v \in \{0, 1\} \qquad \forall v \in V(G), \quad \forall ij \in A(D)$$

$$z_e^f \in \{0, 1\} \qquad \forall f \in E(G), \quad \forall e \in E(G)$$

4 Detalhes da implementação

O modelo foi implementado utilizando o solver *Gurobi Optimizer* [15] em C++. A ferramenta fornece uma interface para a construção de modelos de programação linear, bem como a implementação de algoritmos otimizados para esse tipo de problema. Além disso, foi utilizada também uma biblioteca de C++ chamada *lemon* [16], que oferece diversos tipos e estruturas pensadas para a representação de grafos e estruturas similares. Para proporcionar uma visualização mais confortável das instâncias e resultados produzidos, usou-se a ferramenta *Graphviz* [17], um software para visualização de grafos. A Figura 8 mostra um exemplo de visualização por meio do Graphviz de uma instância resolvida.[1]



(a) Grafo de entrada G.

(b) 5-spanner de G.

Figura 8: Visualização do Graphviz para a solução de uma instância de 50 vértices.

Toda a parte computacional do trabalho foi realizada utilizando um processador Intel® Core™ i3-4005U, com quatro núcleos de processamento de 1.7GHz e 4GB de RAM. O sistema operacional utilizado foi o Linux Ubuntu 18.04, de 64 bits, com interface gráfica GNOME 3.28.2. Vale destacar a restrição de memória do computador, que trouxe limitações inconvenientes para os experimentos computacionais e impossibilitou a execução em instâncias grandes.

4.1 Algoritmo

A implementação do programa de fato tem, resumidamente, três etapas. Primeiro, é feito o tratamento da entrada, bem como um processamento inicial para obter todas as estruturas e variáveis necessárias relativas à instância – é aqui que, por exemplo, construímos o grafo e o digrafo associados à entrada e computamos as distâncias entre os vértices. Isso feito, são realizadas novas iterações pelos vértices, arestas e arcos para criar as variáveis e restrições da formulação linear. Por fim, é chamado o método de otimização do Gurobi para resolver o modelo.

A função de otimização do Gurobi, por sua vez, também tem três etapas. Inicialmente, o modelo passa por um processo de *pré-resolução*, no qual o modelo é transformado em um modelo equivalente que possui as seguintes características: (1) o modelo pré-resolvido

[1] Os pesos das arestas foram omitidos da imagem para evitar poluição visual.

é insolúvel se e somente só o modelo original o for; (2) o modelo pré-resolvido não possui limite se e somente se o modelo original também não possuir; (3) uma solução ótima para o modelo pré-resolvido possui valor igual a uma solução ótima para o modelo original; e (4) qualquer solução do modelo pré-resolvido pode ser mapeada para uma solução do modelo original. O objetivo dessa etapa é criar um modelo menor e mais fácil de resolver. Uma exploração minuciosa sobre como esse procedimento é realizado foge do escopo deste projeto, e portanto não será explorada aqui. Aqui também ocorrem tentativas de obter soluções heurísticas para o problema, de forma a já estabelecer algum limitante superior para a solução final. Foram utilizadas heurísticas providas pelo resolvedor, que buscam alguma solução viável em um período limitado de tempo.

A segunda etapa da otimização do Gurobi é a de *root relaxation*, que resolve uma versão do problema original sem a restrição de integralidade. Por fim, a etapa final utiliza um algoritmo de *branch and cut* para resolver o sistema. Cada nó do algoritmo é resolvido pelo método *simplex*.

4.2 Complexidade

Tanto no tratamento da entrada quanto no preparo da variáveis e restrições da formução, é realizada uma quantidade fixa de iterações pelos vértices e arestas da instância de entrada, levando a uma complexidade de $O(|V| + |E|)$. Vale notar aqui que não houve uma preocupação em fazer esse processamento da forma mais eficiente possível, visto que o foco do projeto estava no desempenho do modelo de programação linear inteira.

O Gurobi utiliza o método simplex para a resolução do sistema linear, que tem, no pior caso, complexidade exponencial, mas performa em tempo polinomial na maior parte dos casos.

5 Experimentos computacionais

A fim de analisar o comportamento do modelo para diferentes tipos de grafos, o programa foi executado em tipos de instâncias variadas. Foram definidas três categorias para testes computacionais: *grafos planares*, *grafos bipartidos*, e grafos gerados de forma aleatória. Para cada uma delas, foram experimentados tamanhos crescentes de entradas, até que a quantidade de memória impossibilitasse a otimização do modelo. Para cada par de tipo e tamanho de instância, o programa foi executado com os fatores de dilatação: 5, 10, 15 e 20.

Os tempos de execução para alguns dos experimentos realizados estão expostos nas tabelas 2, 3 e 4. Cada linha das tabelas corresponde a uma combinação de quantidade de vértices e de arestas, enquanto cada coluna representa uma das quatro possibilidades de fator de dilatação citadas. As células destacadas em verde indicam que um resultado heurístico foi encontrado antes da otimização exata, estabelecendo um limite máximo prévio. Células vermelhas indicam instâncias declaradas insolúveis pelo resolvedor do Gurobi.

Para um melhor efeito comparativo, o ideal seria que, para as diferentes classes de grafos,

fossem testadas instâncias de mesmo tamanho. Entretanto, não é uma tarefa fácil produzir grafos de classes distintas com mesmas quantidades de vértices e de arestas – em grafos planares, por exemplo, o grau dos vértices tende a ser pequeno, reduzindo a proporção entre arestas e vértices. Assim, tentou-se produzir instâncias que possuíssem quantidades de arestas parecidas para cada classe, visto que as arestas são as responsáveis pela maior parte da complexidade do modelo.

Dilatação / $\|V\|$ / $\|E\|$	5	10	15	20
7 / 14	0.02s	0.03s	0.04s	0.03s
15 / 43	0.54s	0.44s	0.39s	0.26s
20 / 112	8.73s	25.21s	13.04s	8.26s
25 / 179	48.81s	70.93s	85.65s	47.25s
30 / 193	260.60s	342.52s	499.64s	488.83s

Tabela 2: Tempos de solução em instâncias de grafos gerados aleatoriamente.

Dilatação / $\|V\|$ / $\|E\|$	5	10	15	20
20 / 43	0.69s	0.67s	0.48s	0.33s
50 / 116	7.55s	6.74s	6.44s	6.11s
69 / 180		22.21s	18.46s	18.56s
76 / 200		31.10s	30.73s	55.73s

Tabela 3: Tempos de solução em instâncias de grafos planares.

Dilatação / $\|V\|$ / $\|E\|$	5	10	15	20
20 / 37	0.96s	0.80s	0.94s	0.84s
30 / 100	2.54s	5.85s	8.18s	3.77s
35 / 153	7.97s	20.84s	19.19s	20.18s
40 / 200	12.90s	116.94	172.89s	350.29s

Tabela 4: Tempos de solução em instâncias de grafos bipartidos.

5.1 Análise dos resultados

Apesar da limitação dos experimentos imposta pelo hardware, podemos obter informações interessantes através da análise dos resultados obtidos sob algumas óticas distintas. No âmbito da comparação entre as classes, observa-se que as instâncias de grafos planares são resolvidas com muito mais facilidade do que as das outras duas classes, com essa diferença tornando-se mais significante conforme o tamanho das instâncias cresce. Dentre os grafos bipartidos e os construídos aleatoriamente, os bipartidos se mostraram mais fáceis de serem

resolvidos, ainda que longe dos planares. Uma explicação razoável para essa disparidade reside nas quantidades de caminhos existentes entre os pares de vértices: quanto mais restritiva a estrutura de uma classe, a tendência é de que exista uma menor gama de possibilidades de caminhos entre vértices.

Os resultados relativos a diferentes fatores de dilatação são curiosos. A princípio, poderia se imaginar que, via de regra, quanto menor o fator de dilatação, mais rápida seria a resolução da instâncias. Afinal, mais configurações de árvores estariam sendo descartadas pela restrição das distâncias. Apesar de os dados indicarem uma tendência nesse sentido, isso não foi verdadeiro para boa parte dos casos de teste. Levantamos duas possíveis razões desse comportamento. Em primeiro lugar, vale notar que, apesar de um fator de dilatação baixo possibilitar uma poda mais rápida do branch and bound pela restrição das distâncias, ele descarta menos soluções por meio de limites superiores. Isto é, ao examinar menos ramos da árvore, o algoritmo deixa de lado soluções que estabeleceriam limites superiores à função objetivo – o que por sua vez também possibilita uma poda mais efetiva de ramos do branch and bound. Esse efeito é ilustrado pelo fato de as duas instâncias insolúveis apresentadas na Tabela 3 apresentarem, por larga margem, os maiores tempos de execução de todos os testes.[2] Além disso, para fatores de dilatação grandes o suficiente, sua variação deixa de influenciar na poda do branch and bound, visto que a restrição das distâncias torna-se verdadeira para todos os casos.

Voltamos agora a atenção para a influência da aplicação de heurísticas no desempenho do programa. Podemos avaliar a qualidade da heurística por duas perspectivas: a quantidade de instâncias para as quais ela obtém algum efeito; e a qualidade desse efeito. No primeiro aspecto, a heurística utilizada pelo resolvedor se mostrou bastante inadequada, funcionando somente em instâncias menores, o que é significantemente ruim, dado que é justamente nas instâncias maiores em que mais se faz necessária a aplicação de heurísticas. Essa inadequação provavelmente poderia ser resolvida – ou ao menos mitigada – pela utilização de heurísticas especificamente pensadas para o problema da árvore t-spanner de peso mínimo, ao invés de heurísticas de uso geral. Com relação à efetividade da heurística nos casos em que de fato ela produziu resultados, os dados colhidos são insuficientes para tirar conclusões assertivas. Ainda assim, indícios de um impacto positivo podem ser observados nas terceira e quarta linhas da Tabela 4.

Por fim, vale ressaltar que, para todas as instâncias geradas, os pesos de todas as arestas foi escolhido aleatoriamente dentro do intervalo $(0, 1)$. Isso significa que, nos grafos testados, a distribuição dos pesos nas arestas tem caráter homogêneo, cenário que dificilmente apareceria em alguma aplicação prática. Essa variedade extrema nos pesos provavelmente contribui para uma solução mais rápida do programa, já que as arestas de peso alto serão preteridas na maior parte das vezes. Uma confguração que provavelmente se adequaria melhor a situações práticas seria, por exemplo, uma distribuição normal.

[2]Por não se tratar de uma medida do tempo dispendido até atingir uma solução viável, essa informação não consta na tabela.

6 Conclusão e trabalhos futuros

O trabalho proporcionou um rico estudo da modelagem do problema da árvore t-spanner de peso mínimo como um problema de programação linear inteira, gerando observações interessantes por meio dos experimentos computacionais. Ainda assim, o desenvolvimente foi prejudicado pelas limitações de tempo e do hardware utilizado. Assim, esta pesquisa pode ser expandida ou melhorada de diversas maneiras, algumas delas citadas abaixo:

- Testes em instâncias grandes;
- Análise de resultados para outras classes de grafos;
- Diferentes configurações para os pesos das arestas;
- Testes com mais valores distintos de fator de dilatação;
- Aplicação de mais e melhores heurísticas para o problema em questão, e análise de seu desempenho;
- Implementação de outras modelagens para o problema e comparação dos resultados.

Referências

[1] D. Peleg and E. Upfal, *A Trade-Off between Space and Efficiency for Routing Tables*. Journal of the ACM, **36** (6), 510–530 (1989).

[2] L. Cai and D. G. Corneil, *Tree Spanners*. SIAM Journal on Discrete Mathematics, **8** (3), 359–387 (1995).

[3] D. Peleg and J. D. Ullman, *An Optimal Synchronizer for the Hypercube*. Em Proceedings of the Sixth Annual ACM Symposium on Principles of Distributed Computing, PODC' 87, 77–85 (1987).

[4] D. Peleg and J. D. Ullman, *An Optimal Synchronizer for the Hypercube*. SIAM Journal on Computing, **18** (4), 740–747 (1989).

[5] D. Peleg and A. A. Schäffer, *Graph spanners*. Journal of Graph Theory, **13** (1), 99–116 (1989).

[6] L. Cai, *Tree spanners: spanning trees that approximate distances*. Tese de Doutorado, University of Toronto, Canada (1992).

[7] L. Cai, *NP-completeness of minimum spanner problems*. Discrete Applied Mathematics, **48** (2), 187–194 (1994).

[8] I. Althöfer, G. Das, D. Dobkin, D. Joseph and J. Soares, *On Sparse Spanners of Weighted Graphs*. Discrete & Computational Geometry, **9** (1), 81–90 (1993).

[9] W. Duckworth and M. Zito, *Sparse Hypercube 3-spanners*. Discrete Applied Mathematics, **103** (1), 289–295 (2000).

[10] M. S. Madanlal, G. Venkatesan and C. P. Rangan, *Tree 3-spanners on interval, permutation and regular bipartite graphs*. Information Processing Letters, **59** (2), 97–102 (1996).

[11] A. Brandstädt, V. Chepoi and F. Dragan, *Distance Approximating Trees for Chordal and Dually Chordal Graphs*. Journal of Algorithms, **30** (1), 166–184 (1999).

[12] S. P. Fekete and J. Kremer, *Tree spanners in planar graphs*. Discrete Applied Mathematics, **108** (1), 85–103 (2001).

[13] P. Berman, A. Bhattacharyya, K. Makarychev, S. Raskhodnikova and G. Yaroslavtsev, *Approximation algorithms for spanner problems and Directed Steiner Forest*. Information and Computation, **222**, 93–107 (2013).

[14] H. Braga, *Algoritmos exatos para problemas de spanner em grafos*. Tese de Doutorado, University of São Paulo, Brasil (2019).

[15] Gurobi Optimization, disponível em https://www.gurobi.com/

[16] Lemon Graph Library, disponível em https://lemon.cs.elte.hu/trac/lemon

[17] Graphviz, disponível em https://graphviz.org/