Algoritmos exatos para problemas de spanner em grafos

Hugo Vinicius Vaz Braga

Tese apresentada
ao
Instituto de Matemática e Estatística
da
Universidade de São Paulo
para
obtenção do título
de
Doutor em Ciência da Computação

Programa: Pós-Graduação em Ciência da Computação

Orientador: Prof. Dr. rer. nat. Yoshiko Wakabayashi

Durante o desenvolvimento deste trabalho o autor recebeu auxílio financeiro da FAPESP (Proc. 2013/22875-9)

São Paulo, fevereiro de 2019

Algoritmos exatos para problemas de spanner em grafos

Esta versão da tese contém as correções e alterações sugeridas pela Comissão Julgadora durante a defesa da versão original do trabalho, realizada em 14/12/2018. Uma cópia da versão original está disponível no Instituto de Matemática e Estatística da Universidade de São Paulo.

Comissão Julgadora:

- Profª. Drª. Yoshiko Wakabayashi (orientadora) - IME-USP
- Prof. Dr.Carlos Eduardo Ferreira - IME-USP
- Prof. Dr. Cláudio Nogueira de Meneses - UFABC
- Profª. Drª. Karla Roberta Pereira Sampaio Lima - EACH-USP
- Prof. Dr. Eduardo Cândido Xavier - UNICAMP

Agradecimentos

Começo fazendo um agradecimento em especial à minha orientadora, Profª Yoshiko Wakabayashi, por todos os ensinamentos e dedicação que teve para comigo. Ter tido a oportunidade de trabalhar com uma pessoa tão qualificada e premiada, e que aceitou orientar um estudante iniciante na área, gera um sentimento de gratidão desde o momento em que você aceitou ser minha orientadora. Certamente Yoshiko foi fundamental para a realização deste trabalho.

Agradeço à minha família, em especial aos meus pais, por todo o apoio ao longo deste doutorado. Em momentos importantes, como na reta final, a presença de vocês foi importante na continuidade deste trabalho. É claro que não posso deixar de mencionar meus irmãos Júnior, Clovis e Marcelle, além da minha avó Eufélia.

Agradeço aos colegas de laboratório, em especial ao Victor, Gabriel e Tiago. Gostaria de agradecer também aos professores com os quais tive a possibilidade de iniciar os estudos na área de otimização combinatória.

Agradeço em especial ao Prof. Alexandre Freire pela orientação e suporte na implementação do algoritmo de *branch-and-price*. A falta de experiência na área de geração de colunas foi minimizada pela ajuda do professor Alexandre. Outro agradecimento em especial ao Prof. Manoel Campêlo pela contribuição na elaboração dos algoritmos exatos para o problema de árvore t-spanner.

Agradeço aos amigos de estatística Douglas, Karina, Guilherme, Aline e Rodrigo (goiano) por terem me recebido tão bem em São Paulo.

Agradeço à Fundação de Amparo à Pesquisa do Estado de São Paulo (FAPESP) pelo financiamento deste trabalho por meio do processo 2013/22875-9. Graças ao suporte da FAPESP, pude me dedicar de forma exclusiva à elaboração deste trabalho.

Resumo

BRAGA, H. **Algoritmos exatos para problemas de spanner em grafos**. 2018. 96 f. Tese (Doutorado) - Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo, 2018.

Seja (G, w, t) uma tripla formada por um grafo conexo $G = (V, E)$, uma função peso não-negativa w definida em E e um número real $t \geq 1$, chamado de *fator de dilatação*. Um *t-spanner* de G é um subgrafo gerador H de G tal que para cada par de vértices u, v, a distância entre u e v em H é no máximo t vezes a distância entre u e v em G. Quando H é uma árvore, dizemos que H é uma *árvore t-spanner*. Nesta tese focamos o *problema da árvore t-spanner de peso mínimo* (cuja sigla em inglês é MWTSP), definido a seguir. Dada uma tripla (G, w, t), encontrar uma árvore t-spanner em G de peso mínimo. É sabido que MWTSP é NP-difícil para cada $t > 1$ fixo. Propomos duas formulações lineares inteiras para MWTSP, baseadas em arborescência, de tamanho polinomial no tamanho de G. A formulação que possui variáveis representando distâncias entres os pares de vértices apresentou resultados melhores na prática.

Também abordamos o *problema de t-spanner de peso mínimo* (cuja sigla em inglês é MWSP), cuja entrada é a mesma do MWTSP e cujo objetivo é encontrar um t-spanner de peso mínimo. Mesmo para grafos com peso unitário, MWSP é NP-difícil para cada $t \geq 2$ fixo. Tratamos este problema de duas formas. Propomos uma formulação linear inteira para o MWSP que possui um número exponencial de restrições, mas cujo problema da separação — para o programa linear relaxado correspondente — é polinomial no tamanho de G. Apresentamos também uma implementação de um algoritmo de *branch-and-price* para o MWSP baseado numa formulação linear inteira proposta por Sigurd e Zachariasen (2004). Exibimos resultados de experimentos realizados com as duas formulações para o MWTSP e também com o algoritmo de *branch-and-price* para o MWSP.

Palavras-chave: grafo, spanner, geração de coluna, algoritmo exato.

Abstract

BRAGA, H. **Exact algorithms for spanner problems in graphs**. 2018. 96 f. Tese (Doutorado) - Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo, 2018.

Let (G, w, t) be a triplet consisting of a connected graph $G = (V, E)$ with a nonnegative weight function w defined on E, and a real number $t \geq 1$. A *t-spanner* of G is a spanning subgraph H of G such that for each pair of vertices u, v, the distance between u and v in H is at most t times the distance between u and v in G. If H is a tree then we call it a *tree t-spanner* of G. We address the *Minimum Weight Tree Spanner Problem* (MWTSP), defined as follows. Given a triplet (G, w, t), find a minimum weight tree t-spanner in G. It is known that MWTSP is NP-hard for every fixed $t > 1$. We propose two ILP formulations for MWTSP, based on arborescences, of polynomial size in the size of G. The formulation that has variables representing distances between pairs of vertices has shown to be better in practice.

We also address the *Minimum Weight t-Spanner Problem* (MWSP) that has the same input as MWTSP and seeks for a minimum weight t-spanner in G. Even for unweighted graphs, it is known that MWSP is NP-hard for every fixed $t \geq 2$. We approach this problem in two ways. We propose an ILP formulation for MWSP that has an an exponential number of restrictions but we show that the separation problem — for the corresponding relaxed linear program — can be solved in polynomial time in the size of G. We also present a branch-and-price algorithm for MWSP based on an ILP formulation proposed by Sigurd and Zachariasen (2004). We show results on the computational experiments with both formulations for MWTSP and also with the branch-and-price algorithm that we implemented for MWSP.

Keywords: graph, spanner, column generation, exact algorithm.

Sumário

Lista de Abreviaturas

APSP	Caminhos mínimos entre todos os pares de vértices (*All Pairs Shortest Path*)
B&B	*Branch-and-bound*
BDMSTP	Árvore Geradora Mínima com Diâmetro Limitado (*Bounded Diameter Minimum Spanning Tree Problem*)
BeFS	Procurar o melhor primeiro (*Best-First Search*)
CR	Formulação com rótulos
CSPP	Problema do caminho mínimo restrito (*Constrained Shortest Path Problem*)
EPTAS	Esquema eficiente de aproximação em tempo polinomial (*Efficient Polynomial-Time Approximation Scheme*)
MMST	Árvore spanner de menor fator de dilatação (*Minimum Max-Stretch spanning Tree*)
MST	Árvore Geradora Mínima (*Minimum Spanning Tree*)
MWSP	Problema de t-spanner de peso mínimo (*Minimum Weight t-Spanner Problem*)
MWTSP	Problema da árvore t-spanner de peso mínimo (*Minimum Weight Tree t-Spanner Problem*)
PL	Programação/Programa Linear
POC	Problema de Otimização Combinatória linear
PTAS	Esquema de aproximação em tempo polinomial (*Polynomial-Time Approximation Scheme*)
RMP	Problema mestre reduzido (*Reduced Master Problem*)
SR	Formulação sem rótulos
TSP	Problema do caixeiro viajante (*Travelling Salesman Problem*)

Lista de Símbolos

$E(G)$	Conjunto de arestas do grafo G
$V(G)$	Conjunto de vértices do grafo G
$A(D)$	Conjunto de arcos do digrafo D
$\text{dist}_G(u, v)$	Distância entre os vértices u e v no grafo G
$\text{dist}_G(u, v, c)$	Distância entre os vértices u e v no grafo G medida pela função c
$\mathbb{R}^{\lvert E\rvert}$	Espaço vetorial real $\lvert E\rvert$-dimensional
$\mathcal{X}^F$	Vetor de incidência de F
$\mathbb{B}^n$	Espaço vetorial binário n-dimensional
Δ	Grau máximo
$\mathbb{N}$	Conjunto dos números naturais
$\mathbb{Z}$	Conjunto dos números inteiros
$\mathbb{Z}_{\geq 0}$	Conjunto dos números inteiros não negativos
$\mathbb{R}$	Conjunto dos números reais
$\mathbb{R}_{\geq 0}$	Conjunto dos números reais não negativos

Lista de Figuras

Lista de Tabelas

Lista de Algoritmos

Capítulo 1

Introdução

Dado um grafo $G = (V, E)$, e um número real $t \geq 1$, um *t-spanner* de G é um subgrafo gerador H de G tal que para quaisquer pares de vértices u,v, a distância entre u e v em H é no máximo t vezes a distância entre u e v em G. Este conceito de *t-spanner* pode ser generalizado para o caso em que às arestas do grafo G estão associados pesos não negativos, ou seja, é dada uma função peso $w : E \to \mathbb{R}_{\geq 0}$, e o conceito de distância é o que comumente entendemos para grafos com pesos (ou comprimentos) nas arestas. Quando o subgrafo H é uma árvore, dizemos que H é uma *árvore t-spanner*. Não encontramos uma tradução que consideramos apropriada para o termo *spanner*, e decidimos manter o termo original, sem grafá-lo em itálico.

Problemas sobre spanners surgem em vários cenários, em diferentes áreas, como computação distribuída [Awe85, PU89], redes de comunicação [PU88, PR99, OP05] e robótica [ACC+96, MB11a, MB11b]. No Capítulo 3, definimos formalmente os principais problemas sobre spanners e apresentamos os resultados sobre algoritmos conhecidos e complexidade computacional desses problemas. Além disso, apresentamos um histórico do estudo de spanner, desde o surgimento do conceito até os dias atuais. Antes de apresentar esses problemas, no Capítulo 2, apresentamos conceitos básicos de teoria dos grafos, combinatória poliédrica e otimização combinatória para estabelecer a terminologia.

Nesta tese focamos dois problemas sobre spanners: *problema da árvore t-spanner de peso mínimo* (cuja sigla em inglês é MWTSP), e *problema de t-spanner de peso mínimo* (cuja sigla em inglês é MWSP). Em ambos os problemas, a entrada é uma tripla (G, w, t). No primeiro problema o objetivo é encontrar uma árvore t-spanner em G de peso mínimo; e no segundo, o objetivo é encontrar um t-spanner em G de peso mínimo.

Ambos os problemas são NP-difíceis. O problema MWTSP já é NP-difícil mesmo para t fixo, quando $t > 1$ [CC95]. O segundo problema, MWSP, já é NP-difícil mesmo quando os grafos têm peso unitário, e t é fixo, $t \geq 2$ [Cai94].

O problema da árvore t-spanner de peso mínimo (MWTSP) tem sido largamente investigado no caso em que os pesos são unitários. Para este caso há na literatura alguns algoritmos e resultados sobre a complexidade computacional para classes específicas de grafos. Identi-

ficamos uma lacuna com relação à existência de um algoritmo exato para o caso de pesos arbitrários, e em vista disso, optamos por desenvolver um tal algoritmo.

No Capítulo 4, propomos duas formulações lineares inteiras para o MWTSP. Estas são as primeiras formulações lineares inteiras específicas para o problema da árvore t-spanner. Diferentemente de formulações propostas para o problema de grafos t-spanner, as formulações que propusemos são específicas para árvores t-spanner (no sentido de não serem formulações para grafos spanners com a exigência adicional de aciclicidade). Além disso, as formulações lineares inteiras propostas por nós para o MWTSP são polinomiais no tamanho de G, diferentemente das formulações existentes para o MWSP. Ambas as formulações são baseadas em arborescências que "se sobrepõem". Esta ideia facilita encontrar o caminho entre dois vértices quaisquer nas arborescências (e, por consequência, na árvore que representa uma solução). As duas formulações diferem quanto à forma como a distância entre os pares de vértices é armazenada na formulação. Em uma das formulações, esta distância é calculada de maneira explícita. Numa segunda formulação, existem variáveis que representam a distância entre pares de vértices. Esta segunda formulação mostrou-se melhor nos experimentos computacionais realizados.

Nas formulações lineares inteiras existentes para o MWSP, geralmente os autores definem variáveis para cada caminho t-spanner [SZ04, DK11a]. Neste caso, se todas as colunas forem consideradas desde o início, a formulação terá um número exponencial (no tamanho de E) de colunas. Neste caso, alguns autores como Sigurd e Zachariasen [SZ04] adotam o método de geração de colunas. Na formulação linear inteira que propomos no Capítulo 5 para o MWSP, adotamos variáveis associadas às arestas e aos pares de arestas. Nesta formulação, o número de variáveis é polinomial no tamanho de G, mas o número de restrições é exponencial no tamanho de E. Para tratar as restrições de tamanho exponencial, do programa linear relaxado correspondente, precisamos resolver o problema da separação. No problema da separação, dado um conjunto de inequações e uma (possível) solução, queremos saber se existe uma inequação violada pela solução. Em outras palavras, desejamos saber se a solução é viável ou obter um certificado que corresponda a uma inequação violada. A separação das restrições cujo tamanho é exponencial (do programa linear relaxado) pode ser feita em tempo polinomial no tamanho de V, visto que nós mostramos que o problema da separação corresponde ao problema de corte mínimo, que pode ser resolvido por um algoritmo de tempo polinomial. Apesar desta proposta diferente (quando comparada às formulações lineares inteiras existentes), os experimentos iniciais mostraram que os resultados não foram bons (e não serão apresentados). O fortalecimento da formulação proposta por nós, através da descoberta de inequações válidas, é uma alternativa para melhorar os resultados.

Sigurd e Zachariasen [SZ04] propuseram um algoritmo exato para o MWSP, com o intuito de verificar quão boa era a heurística gulosa proposta por Althöfer et al [ADD+93]. Os autores propõem uma formulação linear inteira cujo número de variáveis é exponencial (no tamanho de E). Em decorrência disso, os autores adotam um método de *branch-and-bound* (B&B) aliado à geração de colunas para a resolução do problema. O problema do

pricing relacionado à geração de colunas é o *problema do caminho mínimo restrito*. No Capítulo 6, abordamos o algoritmo exato de *branch-and-price* proposto Sigurd e Zachariasen e uma implementação que propusemos para este algoritmo. Discutimos os detalhes desta implementação que fizemos, como os pré-processamentos realizados, as heurísticas utilizadas para calcular limitantes primais e duais, assim como as decisões que afetam os nós numa árvore de B&B, como a estratégia de ramificação e de busca do nó a ser explorado, e o processamento realizado em cada nó da árvore. A implementação de Sigurd e Zachariasen não está disponível, e os autores não comentam sobre a utilização de pré-processamentos e heurísticas para o cálculo de limitantes primais e duais.

No Capítulo 7, apresentamos os experimentos realizados com as formulações lineares inteiras propostas no Capítulo 4 para o *problema da árvore t-spanner de peso mínimo* (MWTSP). Apresentamos também os experimentos computacionais para uma adaptação feita na implementação do algoritmo de *branch-and-price* para que pudesse ser utilizada para o MWTSP. Também apresentamos os resultados computacionais dos testes relativos à implementação do algoritmo de *branch-and-price* para o *problema de t-spanner de peso mínimo* (MWSP).

No Capítulo 8, finalizamos a tese apresentando as considerações finais juntamente com algumas sugestões de trabalhos futuros.

Capítulo 2

Preliminares

A notação que usamos na tese é a comumente adotada na maioria dos textos de teoria dos grafos, combinatória poliédrica e otimização combinatória, que são as disciplinas que permeiam este trabalho. Apresentamos apenas alguns conceitos básicos para estabelecer a terminologia, e em cada seção sugerimos ao leitor a consulta de outros textos, caso alguns conceitos não estejam definidos aqui.

2.1 Conceitos básicos sobre grafos

Um *grafo* é um par ordenado (V, E), onde V é um conjunto de elementos chamados *vértices*, e E é um conjunto de pares não ordenados de elementos distintos de V, chamados *arestas*. Se G é o nome de um grafo definido por um par (V, E), então escrevemos $G = (V, E)$. Algumas vezes, referimos simplesmente a um grafo G, e usamos a notação $V(G)$ e $E(G)$, respectivamente, para denotar o seu conjunto de vértices e de arestas. Muitas vezes, uma aresta $e = \{u, v\}$ é denotada por uv (ou vu). Se $e = uv$, os vértices u e v são chamados *extremos* da aresta e. Em geral, mesmo não estando explícito, denotamos por n o número de vértices do grafo em consideração.

Um *subgrafo* G' de $G = (V, E)$ é um grafo, onde $V(G') \subseteq V$ e $E(G') \subseteq E$. Usamos a notação $G' \subseteq G$ para indicar que G' é um subgrafo de G. Se $G' \subseteq G$ e $V(G') = V$, então dizemos que G' é um subgrafo *gerador* de G. Para $W \subseteq V$, dizemos que G' é um subgrafo *induzido* por W, se $V(G') = W$ e para cada par $u, v \in W$, temos que $uv \in E(G')$ se e só se $uv \in E$. Neste caso, escrevemos $G' = G[W]$. Para $F \subseteq E$, dizemos que G' é um subgrafo *induzido* por F, e escrevemos $G' = G[F]$, se $E(G') = F$ e $V(G')$ é o conjunto dos vértices de $V(G)$ que são extremos das arestas em F.

Num grafo $G = (V, E)$, dois vértices u, v são *adjacentes* ou *vizinhos* em G se $uv \in E$. O conjunto dos vizinhos de um vértice u é o conjunto $N_u(G) = \{v \in V \mid uv \in E\}$. O *grau* de um vértice $v \in V$ é a quantidade de arestas adjacentes a v em G.

Seja $P = (v_1, a_1, v_2, a_2, \ldots, a_{k-1}, v_k)$, com $k \geq 1$, uma sequência alternada de vértices (v_i) e arestas (a_i) de G. Esta sequência é chamada de *caminho* se os vértices são dois a

dois distintos e para $1 \leq i < k$, $a_i = v_i v_{i+1}$. O *comprimento* de P é k, a quantidade de suas arestas. Como um caminho fica bem determinado pela sequência de seus vértices, comumente denotamos um caminho apenas pela sequência de seus vértices: no caso do caminho P, este se simplificaria para $(v_1, v_2, \ldots, v_k)$. Dizemos que P é um caminho de v_1 a v_k (ou entre v_1 e v_k). A *distância* entre dois vértices u e v num grafo G, denotada por $\text{dist}_G(u, v)$, é definida como o comprimento de um caminho de menor comprimento entre u e v em G.

Se $P = (v_1, v_2, \ldots, v_k)$ é um caminho, e existe em G a aresta $v_k v_1$, então a sequência $(v_1, v_2, \ldots, v_k, v_1)$ define um *circuito* em G.

Dizemos que G é um grafo *conexo* se em G existem caminhos entre todos os seus pares de vértices. Um conjunto de vértices I de um grafo G é *independente* se para cada $u, v \in I$, tem-se que $uv \notin E$.

Definimos a seguir algumas operações que são comumente realizadas em grafos: *remoção de vértices*, *remoção de arestas* e *contração de arestas*. A *remoção de um vértice* u de um grafo G resulta em um subgrafo $G' = (V \setminus \{u\}, E')$, onde $E' = E \setminus \{uv \in E \mid v \in N_u(G)\}$. A *remoção de uma aresta* e de G resulta em um subgrafo $G' = (V, E \setminus \{e\})$. A *contração de uma aresta* uv de G resulta em um grafo G' que se obtém de G removendo-se os vértices u e v e acrescentando um novo vértice, digamos w, e fazendo-o adjacente aos vértices de $(N_u(G) \cup N_v(G)) \setminus \{u, v\}$. Neste caso, $|V(G')| = |V| - 1$.

Um conjunto $C \subseteq V$ é uma *clique* de G se os vértices de C são dois a dois adjacentes. Algumas vezes, também chamamos de clique o subgrafo de G induzido por C.

Um *grafo direcionado* ou *digrafo* é um par ordenado (V, A), onde V é um conjunto de elementos chamados vértices, e A é um conjunto de pares ordenados de vértices distintos de V, chamado *arcos*. Também representamos um arco $a = (u, v) \in A$ simplesmente por uv (aqui a ordem dos vértices é importante); e dizemos que o vértice u é o de *origem* e o vértice v é o de *destino*. Também usamos a notação $V(D)$ e $A(D)$, respectivamente, para denotar o conjunto de vértices e de arcos de D.

Para consultar outros conceitos da teoria dos grafos, sugerimos Bondy e Murty [BM08], Bollobás [Bol98] ou Diestel [Die17].

2.1.1 Algumas classes de grafos

A seguir, definiremos algumas classes especiais de grafos, que serão mencionados neste trabalho.

O *complemento* de um grafo G é um grafo H sobre o mesmo conjunto de vértices tal que dois vértices distintos de H são adjacentes se e só se não são adjacentes em G.

Um grafo G é *bipartido* se V admite uma bipartição $X \cup Y$ tal que todas as arestas de G têm um extremo em X e outro em Y. Neste caso, para explicitar a bipartição, escrevemos que $G = (X \cup Y, E)$.

Um *cografo* é um grafo que consiste de um único vértice, ou é a união disjunta de dois cografos, ou é o complemento de um cografo. Um grafo *split* é um grafo que pode ser

particionado em uma clique e um conjunto independente. Um grafo *intervalo* é um grafo cujos vértices estão em correspondência um-a-um com uma coleção finita de intervalos da reta real, ou seja, cada vértice corresponde a um intervalo; e dois vértices são adjacentes se os correspondentes intervalos têm intersecção não vazia.

Um *grafo de permutação* é um grafo cujos vértices representam os elementos de uma permutação, e cujas arestas representam pares de elementos que são invertidos pela permutação.

Um grafo é *planar* se pode ser desenhado no plano sem que haja intersecção de suas arestas, exceto em seus extremos. Um grafo planar é *exoplanar* se pode ser desenhado no plano de modo que todos os seus vértices pertençam à fronteira da face externa.

Um grafo bipartido $G = (X \cup Y, E)$ é *convexo* sobre X se existe uma ordenação de X tal que para cada $y \in Y$, os vértices adjacentes a y são consecutivos na ordenação de X.

Um grafo $G = (V, E)$ é de *comparabilidade* se admite que suas arestas possam ser orientadas de modo a produzir um digrafo $D = (V, A)$ que é uma orientação transitiva, ou seja, sempre que $xy \in A$ e $yz \in A$, então $xz \in A$. Um grafo é de *cocomparabilidade* se seu complemento é um grafo de comparabilidade.

Um vértice v num grafo G é *simplicial* se os vizinhos de v formam uma clique em G. Um grafo G é 1-*split* se ao removermos todos os vértices simpliciais de G resta somente uma clique.

Um grafo G é *cordal* se para todo circuito de comprimento pelo menos quatro tem uma corda, que é uma aresta que não faz parte do circuito mas conecta dois vértices do circuito. Para um circuito de comprimento par, uma corda neste circuito é ímpar se os dois vértices adjacentes pela corda são separados no circuito por uma distância ímpar. Um grafo é *fortemente cordal* se é cordal e para cada circuito de comprimento par e maior ou igual a seis existe uma corda ímpar.

Chamamos de *hipercubo* um grafo cujos vértices e cujas arestas correspondem, respectivamente, aos vértices e às arestas de um hipercubo.

Um grafo H é *menor* de um grafo G se H pode ser obtido a partir de G por meio da remoção de vértices, remoção de arestas ou contração de arestas.

Para ver definições de outras classes de grafos, ou relações entre essas classes, o leitor poderá consultar as referências [BLS99, BDLL04, GYZ13].

2.2 Conceitos básicos sobre poliedros

Sejam $x_1, x_2, ..., x_m$ vetores em $\mathbb{R}^n$. Dizemos que um vetor x é uma *combinação linear* de $x_1, x_2, ..., x_m$, se $x = \sum_{i=1}^{m} \lambda_i x_i$, onde $\lambda_1, \lambda_2, ..., \lambda_m \in \mathbb{R}$. Se, adicionalmente, temos que $\sum_{i=1}^{m} \lambda_i = 1$, então dizemos que x é uma *combinação afim* de $x_1, x_2, ..., x_m$. Uma combinação afim como a acima, com $\lambda_1, \lambda_2, ..., \lambda_m \in \mathbb{R}_{\geq 0}$, é chamada de *combinação convexa* de $x_1, x_2, ..., x_m$.

Dizemos que vetores $x_1, x_2, ..., x_m \in \mathbb{R}^n$ são *linearmente independentes* se o sistema $\sum_{i=1}^{m} \lambda_i x_i = \vec{0}$, $\lambda_i \in \mathbb{R}$, tem como única solução $\lambda_i = 0$ para $i = 1, \ldots, m$. Dizemos que vetores $x_1, x_2, ..., x_m \in \mathbb{R}^n$ são *afim-independentes* se o sistema $\sum_{i=1}^{m} \lambda_i x_i = \vec{0}$, $\lambda_i \in \mathbb{R}$, $\sum_{i=i}^{m} \lambda_i = 0$ tem como única solução $\lambda_i = 0$ para $i = 1, \ldots, m$. O *fecho convexo* de um conjunto $S \in \mathbb{R}^n$ é o conjunto dos vetores em $\mathbb{R}^n$ que podem ser escritos como combinação convexa dos vetores em S. O *fecho inteiro* de um conjunto $S \in \mathbb{R}^n$ é o conjunto dos vetores em $\mathbb{R}^n$ que podem ser escritos como combinação convexa dos vetores inteiros em S.

Um *poliedro* P em $\mathbb{R}^n$ é um conjunto da forma $\{x \in \mathbb{R}^n \mid Ax \leq b\}$, para alguma matrix $A \in \mathbb{R}^{m \times n}$ e para algum vetor $b \in \mathbb{R}^m$. O poliedro P é chamado de *politopo* se P é o fecho convexo de um conjunto finito de pontos. A *dimensão* de P é o número máximo de vetores afim-independentes em P menos um.

Dado um poliedro da forma $Q = \{(u, x) \in \mathbb{R}^p \times \mathbb{R}^q : Au + Bx \leq b\}$, para matrizes $A \in \mathbb{R}^{m \times p}$ e $B \in \mathbb{R}^{m \times q}$ e para algum vetor $b \in \mathbb{R}^m$, a *projeção* de Q no espaço da variável x (ou seja, no espaço $\mathbb{R}^q$) é definida como

$$Proj_x(Q) = \{x \in \mathbb{R}^q \mid \exists u \in \mathbb{R}^p \text{ tal que } (u, x) \in Q\}.$$

Para definições e resultados básicos sobre combinatória poliédrica, o leitor pode consultar [FW96]; e sobre projeções, indicamos [Bal05].

2.3 Otimização combinatória

Problemas de Otimização Combinatória Linear (POC) podem ser definidos por meio de de uma tripla $(E, w, \mathcal{S})$, onde E é um conjunto finito e não vazio, $w : E \to \mathbb{R}_{\geq 0}$ é uma função peso, e $\mathcal{S}$ representa soluções viáveis formadas por subconjuntos de E. O objetivo é encontrar uma solução $S \in \mathcal{S}$ que maximiza ou minimiza o valor da solução, dado por $f(S) = \sum_{e \in S} w_e$. A função $f(S)$ é chamada de *função objetivo*. Neste trabalho, para uma função objetivo $w : E \to \mathbb{R}_{\geq 0}$ e para $e \in E$, usamos a notação w_e em vez de $w(e)$. Vamos assumir que estamos trabalhando com problemas de minimização. Observe que estamos apenas considerando funções objetivo lineares, visto que os problemas de otimização ao longo desta tese terão apenas este tipo de função objetivo. Isto não quer dizer que todos os problemas de otimização terão esta classe específica de função objetivo.

Para cada POC definido pela tripla $(E, w, \mathcal{S})$, podemos associar um poliedro. Para definir um tal poliedro, precisamos definir alguns conceitos antes. Para um espaço vetorial real $|E|$-dimensional $\mathbb{R}^{|E|}$, seja $x \in \mathbb{R}^{|E|}$ um vetor cujos componentes são indexados pelos elementos de E. Para cada solução $S \in \mathcal{S}$, consideremos o *vetor de incidência* $\mathcal{X}^S = (\mathcal{X}^E_e)_{e \in E}$, definido como

$$\mathcal{X}_e^S = \begin{cases} 1, & \text{se } e \in S, \\ 0, & \text{caso contrário.} \end{cases}$$

Podemos então associar um poliedro $P \subseteq \mathbb{R}^{|E|}$ à tripla $(E, w, \mathcal{S})$ como sendo o fecho convexo dos vetores de incidência das soluções em $\mathcal{S}$, isto é

$$P := \text{conv}\{\mathcal{X}^S \in \mathbb{R}^{|E|}_{\geq 0} \mid S \in \mathcal{S}\}.$$

Dessa forma, o POC considerado pode ser definido como $\min\{w^\intercal x \mid x \in P\}$. O conjunto dos vértices de P corresponde ao conjunto dos vetores de incidência das soluções $S \in \mathcal{S}$ (veja [FW96]). Como P é um politopo, então sabemos que P pode ser descrito por meio de um sistema de inequações lineares. Desta forma, podemos modelar um POC como um problema de *Programação Linear* (PL). Ou seja, é possível modelar todo POC como um PL 0/1. Reciprocamente, como mencionado em [FW96], todo PL 0/1 pode ser formulado como um POC. De fato, dado um PL 0/1 da forma $\min\{w^\intercal x \mid Ax \leq b,\ x \in \{0,1\}^n\}$, onde $A \in \mathbb{R}^{m \times n}$ e $b \in \mathbb{R}^m$, podemos formular este PL como um POC definido sobre $(E, w, \mathcal{S})$, onde $E := \{1, 2, ..., n\}$ e $\mathcal{S} := \{S \subseteq E \mid A\mathcal{X}^S \leq b\}$.

Considere o programa linear $z = \min\{w^\intercal x \mid x \in P\}$, onde $P = \{x \in \mathbb{B}^n \mid Ax \leq b\}$, $A \in \mathbb{R}^{m \times n}$ e $b \in \mathbb{R}^m$. Seja $a_i x \leq b_i$, $1 \leq i \leq m$, a i-ésima restrição do conjunto de restrições de P. Um vetor $x \in P$ é chamado de *solução viável* de Q, e o valor de z é chamado de *valor ótimo* de Q. Uma solução viável x^* para a qual o valor ótimo é atingido é chamada de *solução ótima*. O *suporte* de um vetor x é o conjunto dos índices i tais que $x_i \neq 0$. Dizemos que uma solução viável x tem *suporte minimal* se não existe outro vetor viável x' cujo suporte esteja contido propriamente no suporte do vetor x. Para um vetor $x' \in \mathbb{B}^n$, uma restrição $a_i x \leq b_i$ de P é dita *ativa* em x' se $a_i x' = b_i$.

2.3.1 Problema da separação

Um resultado extremamente importante na área de otimização diz respeito à equivalência entre problema de otimização e *problema da separação*. Este assunto e outros correlatos foram seminalmente tratados por Grötschel, Lovász e Schrijver [GLS88]. No problema da separação, dado um poliedro e um ponto no mesmo espaço vetorial, queremos saber se existe uma inequação violada pelo ponto. Em outras palavras, desejamos saber se o ponto pertence ao poliedro ou obter um certificado (da não pertinência) que corresponda a uma inequação violada.

Mesmo quando um poliedro possui um número exponencial de inequações, é possível otimizar uma função objetivo neste poliedro, fazendo algumas restrições na entrada do problema. Apesar de não existir um algoritmo fortemente polinomial (ou seja, que não dependa da representação da entrada) para problemas de programação linear, usando o *método elip-*

sóide e fazendo algumas restrições na entrada, é possível resolver em tempo polinomial problemas de otimização desde que o problema de separação seja resolvido em tempo polinomial (veja Teorema 3.13 em [NW99]).

2.3.2 *Branch-and-bound*

O método de *branch-and-bound* (B&B) é utilizado para resolver (de maneira exata) problemas difíceis de otimização. Em decorrência dos problemas abordados nesta tese, vamos restringir as definições aos problemas de otimização combinatória linear (POC). Como definido na Seção 2.3, um POC é definido por uma tripla $(E, w, \mathcal{S})$, onde $\mathcal{S}$ representa soluções viáveis formadas por subconjuntos de E, e $w : E \to \mathbb{R}_{\geq 0}$ uma função peso. O objetivo é encontrar uma solução $S \in \mathcal{S}$ que minimiza $f(S) = \sum_{e \in S} w_e$. Como dito anteriormente, a função $f : \mathcal{S} \to \mathbb{R}_{\geq 0}$ é denominada de *função objetivo*, enquanto que $\mathcal{S}$ é conhecido como *espaço de busca*.

O método de B&B cria uma árvore de busca de subconjuntos do espaço de busca, sendo que estes subconjuntos também são chamados de *subproblemas*. Inicialmente, o B&B armazena uma solução viável x^* denominada *solução incumbente*. O algoritmo possui uma lista dos subproblemas (nós) ainda não analisados e, a cada iteração, ele seleciona (e remove da lista) um destes subproblemas. Vamos considerar que o subproblema selecionado seja $S \subseteq \mathcal{S}$. Se for possível encontrar uma solução viável $x \in S$ tal que $f(x) < f(x^*)$, então atualizamos a solução incumbente com x. Caso contrário, se for possível provar que não existe solução em S melhor do que a solução incumbente, então o subproblema S é *podado* (operação também conhecida como *podar por otimalidade* [Wol98]). Caso nenhum dos casos aconteça, subproblemas $S_1, S_2, ..., S_i \subset S$ são criados, sendo que estes correspondem aos nós filhos de S na árvore de busca, e armazenados na lista dos subproblemas ainda não analisados. Quando não houver mais nós na lista a serem analisados, a solução incumbente corresponderá a uma solução ótima.

Existem três componentes importantes no B&B: a *estratégia de busca*, a *estratégia de ramificação* e a *estratégia de poda*. A estratégia de busca é responsável por selecionar na árvore de busca o próximo nó a ser analisado. Duas estratégias de busca conhecidas são a busca em largura e a busca em profundidade. Uma terceira abordagem, conhecida como *buscar o melhor primeiro*, será abordada na Seção 6.2.5. A estratégia de ramificação é responsável por definir como os subproblemas filhos serão criados a partir do subproblema sendo analisado. Ela também é responsável por saber qual variável fracionária será selecionada para fazer a ramificação. Por fim, a estratégia de poda é responsável pelas regras que vão definir quais nós não precisam ser analisados e podem ser descartados. Uma destas estratégias é a utilização de *geração de colunas*. Quando esta estratégia é utilizada juntamente ao B&B, o algoritmo é conhecido como *branch-and-price*.

Branch-and-price

Na geração de colunas, quando o PL possui muitas colunas, a maioria destas é deixada de fora da relaxação deste modelo para que este possa ser resolvido rapidamente. Uma justificativa para esta abordagem é que numa solução ótima, a maioria das colunas terá suas variáveis com valor zero. A versão reduzida (com menos colunas) da relaxação do PL é conhecida como *problema mestre reduzido* (*Reduced Master Problem* - RMP). Inicialmente, muitas vezes o PL é reformulado para melhorar os limites obtidos. Esta reformulação do PL inicial é conhecida como *problema mestre* (*master problem*). O RMP é modelado a partir do problema mestre. No algoritmo baseado em geração de colunas a ser apresentado na seção 6.1, o problema mestre não será necessário.

Para testar se o RMP contém as colunas necessárias para obter uma solução ótima, um subproblema conhecido como *pricing* é resolvido para descobrir se novas colunas podem entrar na base e melhorar o valor corrente da função objetivo. Para um problema de minimização, encontrar tal coluna significa encontrar aquela que possui *custo reduzido* menor do que zero. Cada vez que uma coluna é encontrada, ela é adicionada ao RMP e este é otimizado novamente. Quando não for encontrado mais colunas e a solução do RMP não for inteira, então realiza-se a operação de ramificação. A Figura 2.1 (adaptada de [Kel17]) ilustra o funcionamento do *branch-and-price*. O *pricing* é responsável por encontrar restrições que estão sendo violadas no dual do problema mestre. Estas restrições correspondem às colunas que devem ser inseridas no RMP.

Heurísticas

No contexto dos problemas de otimização, as *heurísticas* são algoritmos para encontrar soluções boas mas não necessariamente ótimas, de maneira rápida [NW99]. No método de B&B, as heurísitcas são interessantes não só para gerar uma solução incumbente inicial, antes de iniciar o método de B&B propriamente dito, mas também para melhorar a solução incumbente ao longo da excução do B&B [NW99, MJSS16].

Considerando um problema de otimização de minimização, uma heurística *primal* é aquela que gera uma solução viável do PL, fornecendo uma cota superior para uma solução ótima. Por outro lado, uma heurística *dual* gera uma cota inferior para o PL.

Dentre as referências sobre otimização combinatória, mencionamos [Wol98, BJN+98, NW99, DL05, MJSS16].

2.4 Algoritmos de aproximação

Seja $\Pi = (E, w, \mathcal{S})$ um POC, e seja $\mathcal{I}$ o seu conjunto de instâncias. Para cada instância $I \in \mathcal{I}$, seja opt(I) o valor de uma solução ótima para a instância I. Seja A um algoritmo polinomial para Π tal que, para cada instância $I \in \mathcal{I}$, $A(I)$ devolve uma solução viável de I. Como dito no início da Seção 2.3, para cada $S \in \mathcal{S}$, a função $f(S)$ devolve o valor

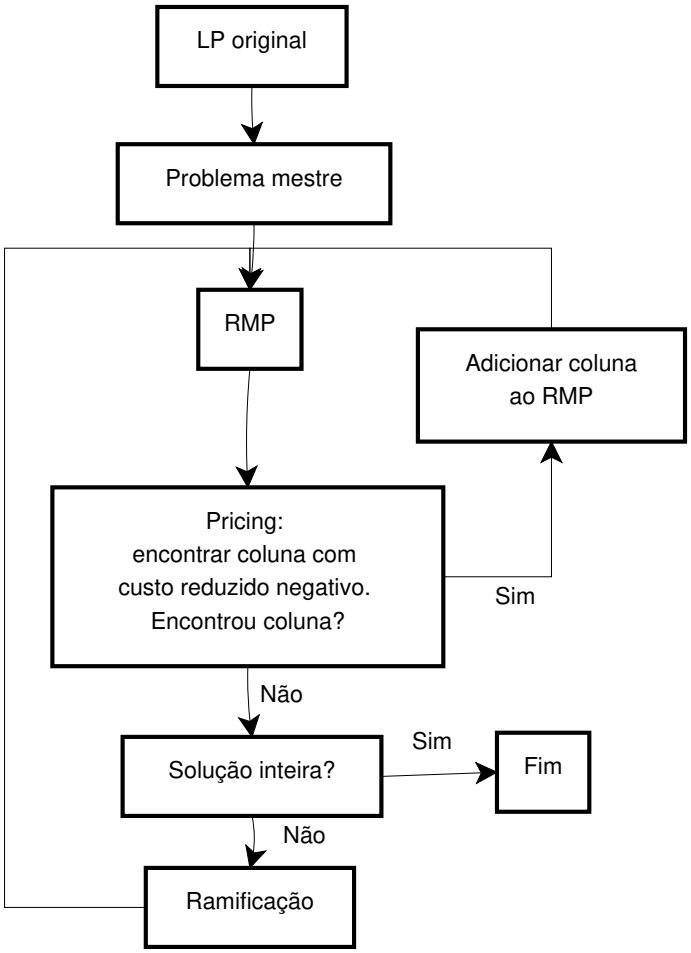

Figura 2.1: *Diagrama de funcionamento do* branch-and-price *para PL de minimização*

associado à solução S. Dizemos que o algoritmo A é uma α-aproximação para Π, problema de minimização, se, para toda instância $I \in \mathcal{I}$, temos que

$$f(A(I)) \leq \alpha \, \mathrm{opt(I)}.$$

Se Π é de maximização, a desigualdade acima deve ser trocada por $\geq$.

O termo α (pode ser uma constante ou uma função de I) é chamado *fator de aproximação* do algoritmo A. Dizemos que um problema Π de minimização tem um *fator de inaproximabilidade* β se Π não admite uma α-aproximação onde $\alpha \leq \beta$.

Um algoritmo A é um *esquema de aproximação de tempo polinomial* (*Polynomial-Time Approximation Scheme*), ou simplesmente PTAS, para um problema Π de minimização se, para cada instância de Π e para cada racional *fixo* $\epsilon > 0$, produz em tempo polinomial uma $(1+\epsilon)$-aproximação. Um PTAS pode não ser muito útil do ponto de vista prático quando o expoente do polinômio cresce muito à medida que o valor de ϵ diminui. Uma variante deste esquema corresponde ao *esquema eficiente de aproximação em tempo polinomial* (*Efficient Polynomial-Time Approximation Scheme* - EPTAS), cujo tempo de execução deve ser $O(n^c)$, onde n é o tamanho da instância e c é uma constante; mas neste caso, a constante da notação O pode depender de ϵ.

Ao leitor interessado em mais conceitos e resultados sobre algoritmos de aproximação, recomendamos consultar Ausiello et al [APMS+99] e Williamson e Shmoys [WS11].

Capítulo 3

Problemas sobre spanners, aplicações e histórico

Neste capítulo introduzimos o conceito de spanners em grafos, e mencionamos vários problemas e resultados sobre spanners. Também apresentamos diversas situações ou contextos onde problemas sobre spanners ocorrem. Posteriormente, formalizamos alguns problemas clássicos sobre spanners, dando enfoque aos problemas estudados nesta tese, e mencionamos os principais resultados sobre a complexidade computacional desses problemas. Por fim, apresentamos um histórico a respeito das investigações relativas aos problemas sobre spanner, destacando o foco da pesquisa em cada período assim como alguns resultados sobre algoritmos ou complexidade.

Dada uma tripla (G, w, t) formada por um grafo $G = (V, E)$, uma função peso $w : E \to \mathbb{R}_{\geq 0}$, e um número real $t \geq 1$, chamado de *fator de dilatação*, um *t-spanner* de G é um subgrafo gerador H de G tal que para quaisquer pares de vértices u,v, a distância (definida como a soma dos pesos das arestas que compõem um caminho de peso mínimo) entre u e v em H é no máximo t vezes a distância entre u e v em G. Mais formalmente,

$$\text{dist}_H(u, v) \leq t \cdot \text{dist}_G(u, v), \quad \forall\, u, v \in V. \tag{3.1}$$

Quando H é uma árvore, dizemos que H é uma *árvore t-spanner*.

Lembramos que, para uma função peso $w : E \to \mathbb{R}_{\geq 0}$ definida sobre o conjunto de arestas de um grafo $G = (V, E)$, para cada aresta $e \in E$, escrevemos w_e em vez de $w(e)$.

Se P é um caminho entre u e v em G cujo comprimento é no máximo $t \cdot \text{dist}_G(u, v)$, então P é chamado de *caminho t-spanner* entre u e v.

Para vértices u e v em um grafo G, denotamos por $\mathcal{P}_{u,v}(G)$ a coleção de caminhos t-spanner em G entre u e v. Dizemos que uma aresta $f \in E$ é uma *aresta t-essencial* de G se $G - f$ não é t-spanner de G.

Problemas sobre spanners surgem em vários cenários, em diferentes áreas, como computação distribuída [Awe85, PU89], redes de comunicação [PU88, PR99, OP05] e robótica [ACC+96]. Essa sua aplicabilidade em diversas áreas tem motivado ampla pesquisa e

interesse em áreas como teoria dos grafos, otimização combinatória, e mesmo geometria computacional (nas variantes em que as distâncias são euclidianas).

3.1 Aplicações de problemas sobre spanners

De modo geral, problemas sobre spanners surgem em cenários onde há a necessidade de economia de recursos ou existe a necessidade de computação rápida de informações sobre caminhos. Dentre esses cenários, destaca-se redes de comunicação [PU88, PR99, OP05, RTZ08], mais especificamente, quando se deseja fazer roteamento que consuma pouco espaço ou tempo. Neste cenário, busca-se uma relação de compromisso entre o tamanho da rede armazenada e o comprimento da rota mais longa para interligar quaisquer dois pares de pontos. Uma das ideias consideradas é basicamente considerar a rede original como um grafo, e construir um spanner desse grafo. Desta forma, ao obter uma subrede com um conjunto menor de arestas do que a rede original, basta utilizar apenas este subconjunto menor de arestas no roteamento. Neste caso, as rotas entre os pares de pontos terão comprimento no máximo t vezes pior do que os das rotas na rede original.

Outro cenário é o de construção de oráculos de distância [RTZ05, TZ05, EP16, Som16, Knu17]. No problema de *Caminhos Mínimos entre todos os Pares de Vértices* (cujo acrônimo em inglês é APSP), estamos interessados nas distâncias entre todos os pares de vértices. Como abordado em [TZ05], em muitas situações, não é necessário obter esta informação entre todos os pares de vértices. Além do mais, em alguns cenários, para um grafo $G = (V, E)$, armazenar uma estrutura cujo espaço é $O(|V|^2)$ pode representar um gasto excessivo.

Outra restrição com relação ao APSP consiste no tempo de pré-processamento (no APSP, o tempo é $O(|V| \cdot |E|)$). Este conjunto de fatores caracteriza a motivação para a utilização de oráculos de distância, cujo papel é devolver a informação aproximada da distância rapidamente. Nestes oráculos, o pré-processamento realizado é mais rápido quando comparado ao pré-processamento do APSP. Neste pré-processamento, é construída uma estrutura que possibilita que as consultas às distâncias sejam efetuadas rapidamente. Estas estruturas são menores e correspondem a um spanner.

Os spanners aparecem em computação distribuída, comumente na construção de algoritmos distribuídos para calcular caminhos mínimos aproximados [Elk05, EZ06, BKKL17]. Novamente aqui, a ideia é utilizar um algoritmo distribuído num spanner do grafo original. O protocolo distribuído apresentado em [DH98] oferece suporte a objetos móveis em uma rede distribuída. Este protocolo é implementado sobre uma árvore geradora da rede e seu *overhead* é proporcional ao fator de dilatação da árvore. Neste caso, um bom candidato para esta árvore geradora seria uma árvore t-spanner. Os spanners também desempenham o papel de *sincronizadores* em sistemas distribuídos [Awe85, PU89]. Eles são utilizados para simular um algoritmo distribuído síncrono em um ambiente assíncrono. Os autores mostraram que o custo (medido em termos de quantidade de mensagens e tempo) está diretamente

relacionado ao fator de dilatação e à quantidade de arestas de um grafo spanner da rede.

Spanners são utilizados no contexto de teste de propriedades [BGJ+09, BGJ+10, BBM+11], mais especificamente no teste de monotonicidade de funções. Para descrever a aplicação, precisamos apresentar antes algumas definições. A redução transitiva de uma relação binária R em um conjunto S é a relação mínima R' em S tal que o fecho transitivo de R' é o mesmo que o fecho transitivo de R. No caso de um digrafo $D = (V_d, A_d)$, a redução transitiva de D corresponde a um outro digrafo $D' = (V_d, A'_d), A'_d \subseteq A_d$, onde A'_d deve ter o tamanho mínimo possível tal que para cada par $u, v \in V_d$, se existe um caminho entre u e v em D então deve haver um caminho entre u e v em D'. Lembramos aqui que o Diagrama de Hasse é usado para representar a redução transitiva de um conjunto finito parcialmente ordenado. Mais formalmente, para um conjunto parcialmente ordenado $(S, \leq)$, no Diagrama de Hasse, cada elemento de S é representado como um vértice. Para cada $x, y \in S$, existe um segmento ascendente de x para y no diagrama se y cobre x, ou seja, não existe $z \in S$ tal que $x < z < y$. A restrição na existência dos segmentos ascendentes caracteriza a redução transitiva.

Com relação ao teste de monotonicidade de funções, seja V_n um conjunto parcialmente ordenado de n elementos e seja $D_n = (V_n, A)$ o Diagrama de Hasse para V_n. Uma função $f : V_n \to \mathbb{R}$ é monótona se para cada $xy \in A$, $f(x) \leq f(y)$. Dizemos que a função f é ϵ-distante de ser monótona se $min_{\text{monótona } g}|\{x : f(x) \neq g(x)\}| \geq \epsilon n$. Um testador de monotonicidade em D_n é um algoritmo que *aceita* se f é uma função monótona, e *rejeita* com alta probabilidade quando f é ϵ-distante de ser monótona. Trabalhos como [BGJ+09, BGJ+10, BBM+11] propõem testadores de monotonicidade cuja estrutura básica é um spanner. Além disso, foi observado que a complexidade do testador está relacionada com o tamanho do spanner.

Spanners são utilizados na área de robótica, mais especificamente na área de planejadores de roteiros [MB11a, MB11b, MB12, MB13, WBC15]. Os planejadores de roteiro (*roadmap planners*) em robótica são algoritmos que determinam rotas entre uma configuração inicial de um robô e uma configuração-objetivo (final), de modo a evitar colisões. Estes planejadores são divididos em duas fases: na primeira (pré-processamento), o planejador acumula conhecimento sobre o espaço; na segunda, o planejador responde perguntas sobre rotas. Uma das formas de melhorar a qualidade do caminho devolvido é construir, na fase de pré-processamento, mapas densos de roteiros. Esta abordagem incorre em desvantagens tais como custo de armazenamento e transmissão das rotas, além de um tempo maior para calcular as rotas. Com o intuito de contornar este problema, os autores em [MB11a] constroem um spanner do grafo que representa os roteiros. Tal abordagem é interessante, pois os computadores embarcados nos robôs geralmente possuem restrições de recursos. Um mapa denso pode ser construído em um computador central e um spanner deste mapa pode ser transmitido para os robôs. Arestas do mapa de rotas podem ser invalidadas em decorrência de obstáculos em tempo real. Quando se utiliza um spanner, recalcular a rota pode ser feita rapidamente.

Aplicações que representam relações tais como *internet* e *hiperlink* ou artigos e citações

trabalham com uma grande massa de dados. Em decorrência do tamanho dos grafos que representam estas relações, pesquisadores têm adotado o modelo *stream* de dados. Neste modelo, o algoritmo tem como entrada um fluxo de dados e deve processar este fluxo na mesma ordem em que os dados vão chegando. Uma outra restrição imposta pelo modelo é que o algoritmo deve usar uma quantidade limitada de memória. Para conciliar acurácia e gasto de memória, muitos autores estão utilizando spanners na construção dos algoritmos para o modelo *stream* de dados [AGM12, CMS13, McG14, KW14, ES16, BKKL17].

Spanners foram úteis para obter um primeiro PTAS para o problema do caixeiro viajante (o bem conhecido *Traveling Saleman Problem -TSP*) para grafos planares [Kle06]. Além disso, spanners foram utilizados no problema de controle de acesso, mais especificamente no gerenciamento de chaves numa hierarquia de acessos [ABFF09, BGJ+09]. Este problema surge no cenário onde os usários têm acesso a um conjunto de serviços e a todos os serviços que são descendentes na hierarquia de acesso.

3.2 Variantes de problemas sobre spanners

Definimos aqui os principais problemas sobre spanner em grafos, e mencionamos algoritmos e resultados conhecidos sobre suas complexidades computacionais. Para todos os problemas, a entrada é sempre uma tripla (G, w, t), onde $G = (V, E)$ é um grafo, $w : E \to \mathbb{R}_{\geq 0}$ é uma função peso, e $t \geq 1$ é um número real. No caso especial em que $w_e = 1$ para toda aresta e de G, dizemos que se trata do *caso unitário* (ou *caso cardinalidade*, já que se deseja minimizar a cardinalidade do conjunto de arestas do spanner). No que segue, mesmo que não esteja mencionado, convencionamos que $n = |V|$ e $m = |E|$.

3.2.1 Problema de t-spanner de peso mínimo

Dada uma tripla (G, w, t), o *problema de t-spanner de peso mínimo* (*Minimum Weight t-Spanner Problem* - MWSP) tem por objetivo encontrar em G um t-spanner que seja de peso mínimo.

No caso unitário, Cai e Corneil [CC95] referem-se a este problema como o *problema de t-spanner mínimo*. A versão de decisão deste problema é conhecida como *problema de t-spanner esparso*. Peleg e Schäffer [PS89] mostraram que o problema de t-spanner esparso é NP-completo quando $t = 2$. Este resultado foi estendido para todo inteiro (fixo) $t \geq 2$ por Cai [Cai94]. Quando $t = 2$, Kortsarz [Kor01] mostrou que é NP-difícil obter uma aproximação para o problema com fator $c \log n$, para algum $c < 1$ e, neste sentido, o algoritmo de aproximação proposto por Kortsarz e Peleg [KP94] com fator $\log(m/n)$ é o melhor possível. Para $t \geq 3$, também é NP-difícil obter uma aproximação com fator $c \log n$, para algum $c < 1$, mas neste caso c depende de t [Kor01]. Este fator foi melhorado em 2015 por Dinitz et al [DKR15] para $2^{\frac{\log^{1-\epsilon} n}{t}}$. Para $t \geq 3$, o algoritmo proposto por Althöfer et al [ADD+93] garante um fator de aproximação de $O(n^{\frac{2}{t-1}})$.

Trabalhos	Resultados
Arestas com peso unitário (t-spanner mínimo)	
Peleg e Schäffer '89 [PS89]	NP-c ($t = 2$)
Cai '94 [Cai94]	NP-c ($t \geq 2$)
Kortsarz '01 [Kor01]	INAPROX - $c \log n$ ($t = 2$, $c < 1$)
Kortsarz e Peleg '94 [KP94]	APROX - $O(\log m/n)$ ($t = 2$)
Dinitz et al '15 [DKR15]	INAPROX - $2^{\frac{\log^{1-\epsilon} n}{t}}$ ($t \geq 3$, $\epsilon > 0$)
Althöfer et al '93 [ADD+93]	APROX - $O(n^{\frac{2}{t-1}})$ ($t \geq 3$)
Arestas com peso arbitrário	
Chandra et al '92 [CDNS92]	APROX - $O(n^{\frac{2+\epsilon}{t-1}})$ ($t > 1$, $\epsilon > 0$)
Chechik et al '16 [CWN16]	APROX - $O(n^{1/k}(1/\epsilon)^{3+2/k})$ (t inteiro, $\epsilon > 0$) (k-spanner, onde $k = (2t-1)(1+\epsilon)$)
Sigurd e Zachariasen '04 [SZ04]	AE-BPC*
AE (Algoritmo Exato); BP (*Branch and Price*); BPC (*Branch and Price and Cut*)	
APROX (Alg. Aproximação); INAPROX (Fator de Inaproximabilidade); NP-c (NP-completo)	

Tabela 3.1: *Problema de t-spanner de peso mínimo*

Os autores em [ENS15, CWN16] melhoram a análise do algoritmo clássico proposto em [CDNS92, ADD+93], garantindo um fator de aproximação menor. Para um fator de dilatação $k = (2t-1)(1+\epsilon)$, Chechik e Wulff-Nilson [CWN16] provaram que o algoritmo guloso gera um spanner cujo peso é $O(w(MST(G)) \cdot n^{1/k}(1/\epsilon)^{3+2/k})$.

Para outros resultados de aproximabilidade e inaproximabilidade para algumas variantes deste problema, veja [EP05, EP07].

Resultados sobre (in)aproximabilidade e complexidade do MWSP estão listados na Tabela 3.1.

Na seção sobre o histórico dos problemas de spanners, apresentaremos mais alguns resultados sobre o problema de t-spanner mínimo para classes específicas de grafos.

3.2.2 Problemas sobre árvore t-spanner

Consideramos aqui três variantes de problemas sobre árvores spanner.

A primeira delas, a mais estudada, é o tópico da primeira subseção.

Problema da árvore t-spanner

Dado um par (G, t), onde G é um grafo e t é um número inteiro maior que 1, o *problema da árvore t-spanner* tem como objetivo saber se G admite uma árvore t-spanner. Neste problema, estamos supondo que o peso das arestas é unitário. O caso com pesos arbitrários veremos na seção a seguir.

Tipos de grafos	$t = 3$	t
cografo, *split*, complemento de bipartido	P [Cai92]	
intervalo, permutação, bipartido regular	P [MVR96]	
planar	P [FK01]	
exoplanar		P ($t \geq 1$) [NR15]
bipartido convexo	P [VRM+97]	
bipartido	? [MVR96]	NP-c ($t \geq 5$) [BDL+07]
grafo de cocomparabilidade	? [MVR96]	
grau máximo $\leq b \cdot \log n$, $b > 0$	P [Pap18]	
fortemente cordal	P [BDLL04]	P ($t = 4$) [BCD99]
1-split	P [BDLL04]	
cordal com diâmetro ≤ 2	P [BDLL04]	
cordal com diâmetro $\leq t + 1$ (para t par)		NP-c ($t \geq 4$) [BDLL04]
cordal com diâmetro $\leq t + 2$ (para t ímpar)		NP-c ($t \geq 4$) [BDLL04]
grafo com diâmetro ≤ 5	P [Pap14]	
cordal	? [MVR96]	NP-c ($t \geq 4$) [BDLL04]
grafos que não possuem K_6 como menor		NP-c ($t \geq 4$) [DFG11b]
P (Polinomial); NP-c (NP-completo)		

Tabela 3.2: *Problema da árvore t-spanner*

Encontrar uma árvore t-spanner pode ser resolvido em tempo linear quando $t = 2$ (veja [Bon89, Cai92, CC95]). Cai e Corneil [Cai92, CC95] mostraram que determinar a existência de uma árvore t-spanner é NP-completo para todo $t \geq 4$ fixo, fazendo uma redução do problema 3-SAT. No caso em que $t = 3$ a sua complexidade computacional é desconhecida. Para o caso $t = 3$, Papoutsakis [Pap13] apresentou um programa inteiro com formulação robusta.

Na Tabela 3.2, apresentamos vários resultados conhecidos para o problema da árvore t-spanner. Os casos em aberto estão indicados com o símbolo '?'. Estes resultados, para classes específicas, serão abordados na seção de histórico dos problemas sobre spanners.

A segunda variante é uma versão de otimização do problema da árvore t-spanner, quando são dados pesos nas arestas.

Árvore t-spanner de peso mínimo

Dada uma tripla (G, w, t), o *problema da árvore t-spanner de peso mínimo* (*Minimum Weight Tree t-spanner Problem* - MWTSP) tem por objetivo encontrar em G uma árvore t-spanner que tenha peso mínimo.

Observe que no caso em que os pesos são unitários, este problema corresponde ao problema clássico da árvore t-spanner. Estamos supondo que o grafo G admite uma árvore t-spanner. Em decorrência dos resultados de NP-completude para a versão de decisão do MWSP (vistos na subseção anterior), a versão de decisão do MWTSP também é um pro-

Trabalhos	Resultados
Arestas com peso unitário	
Cai e Corneil '95 [CC95] Emek e Peleg '08 [EP08]	NP-c ($t \geq 4$) APROX - $O(\log n)$
Arestas com peso arbitrário	
Cai e Corneil '95 [CC95] Peleg e Reshef '99 [PR99] Galbiati '01 [Gal01]	NP-c ($t > 1$) INAPROX - $c(1+\sqrt{5})/2$, ($c < 1$) INAPROX - $(2-\epsilon)$, $\epsilon > 0$
APROX (Alg. Aproximação); INAPROX (Fator de Inaproximabilidade); NP-c (NP-completo)	

Tabela 3.3: *Problema da árvore spanner de menor fator de dilatação*

blema NP-completo para os casos anteriormente mencionados.

Cai e Corneil [CC95] provaram que o problema de decisão correspondente é NP-completo para todo $t > 1$ fixo.

Árvore spanner de menor fator de dilatação

Uma outra variante de problema sobre árvores spanners, que chamaremos de problema da *árvore spanner de menor fator de dilatação* (*Minimum Max-Stretch spanning Tree* - MMST), consiste em minimizar o fator de dilatação. Neste caso, é dado apenas um par (G, w), e o objetivo é encontrar o menor t (real) tal que G admite uma árvore t-spanner.

No caso unitário, o problema MMST foi investigado por Emek e Peleg [EP08]. Esses autores propuseram um algoritmo com fator de aproximação $O(\log n)$. Fekete e Kremer [FK01] provaram que o problema MMST é NP-difícil mesmo quando restrito aos grafos planares. Galbiati [Gal01] provou que é NP-difícil obter uma $(2-\epsilon)$-aproximação, para todo $\epsilon > 0$. Na verdade, a autora provou um resultado de inaproximabilidade para um outro problema, e mostrou que o resultado é válido para o problema da MMST com peso unitário. Para o caso geral, Peleg e Reshef [PR99] provaram que o problema MMST não admite uma α-aproximação, onde $\alpha \leq (1+\sqrt{5})/2$.

Apresentamos na Tabela 3.3 alguns resultados de inaproximabilidade e complexidade computacional do problema MMST.

3.3 Histórico

A seguir apresentamos um histórico sobre a evolução do estudo de problemas sobre spanners. Para cada período, abordamos o foco da pesquisa assim como descrevemos alguns resultados específicos. Além disso, alguns resultados de complexidade para classes específicas de grafos, que não foram apresentados na seção anterior, serão abordados nesta seção.

3.3.1 1986 – 1989: início e spanners esparsos

O conceito de spanner foi introduzido em 1987 (numa conferência) por Peleg e Ullman [PU87], que mostraram que spanners podem ser usados para construir sincronizadores que permitem que algoritmos síncronos executem em ambientes assíncronos (a versão completa do trabalho foi apresentada em [PU89]. Em 1986, numa conferência em geometria computacional, Chew [Che86] apresentou o problema de aproximar grafos completos euclidianos por subgrafos planares (a versão completa do trabalho foi apresentada em [Che89]). O termo spanner foi utilizado pela primeira vez em [PS89].

Spanners esparsos começaram a ser estudados em 1989 por Peleg e Schäffer [PS89]. Estes autores mostraram que, dado um grafo G e um inteiro $M \geq 1$, determinar se G possui um 2-spanner com no máximo M arestas é NP-completo. Os autores ainda mostraram que, dado um grafo G com n vértices, para $1 < k < n$, é possível construir polinomialmente um $(4 \log_k n + 1)$-spanner com no máximo kn arestas.

3.3.2 1992 – 1995: complexidade de problemas sobre árvores t-spanner e spanners esparsos

Os autores que publicaram trabalhos neste período provaram resultados de complexidade para o problema clássico de árvore t-spanner, como Cai [Cai92] mostrou em 1992, em sua tese de doutorado, que o problema da árvore t-spanner é NP-Completo para $t > 1$ (este resultado apareceu em [CC95]). Outros resultados para o caso unitário também foram provados, como abordamos na Seção 3.2.2.

A esparsidade de um spanner pode ser medida em termos da quantidade de arestas ou do peso do spanner. Motivados pelo estudo de spanners mais esparsos, Cai [Cai94] provou resultados de complexidade para o problema de t-spanner esparso, como abordado na Seção 3.2.1. Considerando o peso do spanner, o problema de t-spanner de peso mínimo também foi estudado, resultando no algoritmo guloso clássico de Althöfer et al [ADD+93].O interessante é que este algoritmo além de ser simples e limitar o número de arestas do spanner, ele também limita o peso do spanner.

3.3.3 1993 – 2001: classes específicas de grafos

Para o problema de t-spanner mínimo, em decorrência dos resultados de NP-completude para grafos arbitrários, vários autores estudaram o problema para classes específicas de grafos.

Considerando grafos bipartidos, para $t \geq 3$, Venkatesan, Rotics, Madanlal, Makowsky e Rangan [VRM+97] simplificaram a prova apresentada por Cai [Cai94]. Para $t \geq 2$, os mesmos autores simplificaram a prova de NP-completude apresentada por Cai e Keil [CK94] para grafos com grau máximo nove. Os autores ainda provaram resultados de complexidade para grafos do tipo cordal, intervalo e *split*. Para a classe dos grafos cordais, Peleg e Schäffer [PS89]

propuseram algoritmos polinomiais com um limitante no número de arestas na solução, para os casos em que $t = 2, 3, 5$. Para $t = 2$, os autores mostraram um limite inferior no número de arestas de um conjunto (de tamanho infinito) de soluções e, nesse sentido, o algoritmo proposto pelos mesmos é ótimo. Para um hipercubo, quando $t = 2$, o problema é fácil em decorrência do fato de um hipercubo ser um grafo bipartido. Não existe nenhum subgrafo próprio de um grafo bipartido que seja um 2-spanner. Para $t = 3$, Duckworth, Wormald, Nicholas e Zito [DZ00] conseguiram provar um limite inferior de $\frac{3n}{2} - o(1)$, onde $n = 2^d$ corresponde ao número de vértices de um hipercubo de dimensão d, e um limite superior de $4n$. Com relação aos grafos planares, Brandes e Handke [BH97] mostraram que, para $t \geq 3$, o problema de t-spanner mínimo é NP-completo. Estes resultados estão listados na Tabela 3.4.

Alguns autores também consideraram a variante do problema de t-spanner mínimo em que existe o requisito adicional de que a solução resultante tenha grau máximo limitado. O grau máximo será representado pela letra Δ. Para $t \geq 2$ e um $\Delta \geq 4$, Cai e Keil [CK93] mostraram que o problema de t-spanner mínimo com grau máximo Δ é NP-completo. Os autores mostraram que o mesmo problema para $\Delta = 2$ possui solução linear. Para a versão de otimização do problema de t-spanner mínimo com grau máximo limitado, os autores em [KP98] propuseram um algoritmo probabilístico que gera um 2-spanner com grau máximo, com alta probabilidade, não maior do que $O(\Delta^{1/4})$ vezes o grau máximo ótimo.

Para o problema da árvore t-spanner, vários autores examinaram classes específicas de grafos. Madanlal, Venkatesan e Rangan [MVR96] mostraram que as classes de grafo intervalo, permutação e bipartido regular admitem uma 3-spanner, podendo ser encontrada em tempo polinomial para as três classes. Eles conjecturam que grafos de cocomparabilidade admitem uma 3-spanner. Considerando a classe de grafos fortemente cordais, os autores em [BCD99] mostraram um algoritmo de tempo polinomial para construir uma árvore 4-spanner, dentre outros resultados. Para os grafos planares, os autores em [FK01] mostram que encontrar uma árvore 3-spanner pode ser feita em tempo polinomial. Estes resultados estão resumidos na Tabela 3.2.

3.3.4 2001 – 2008: resultados de (in)aproximabilidade para classes gerais

No período de 2001–2008 foram obtidos resultados de (in)aproximabilidade para o problema de t-spanner mínimo, como foi descrito na Seção 3.2.1.

Para o problema de t-spanner de peso mínimo, Sigurd e Zachariasen [SZ04] propuseram um algoritmo exato usando o método *branch-and-price*.

Como foi mencionado na Seção 3.2.2, para o caso unitário do problema da árvore spanner de menor fator de dilatação, Emek e Peleg [EP04] propuseram um algoritmo de aproximação cujo fator de aproximação é de $O(\log n)$ (a versão completa do trabalho foi apresentada em [EP08]). O algoritmo é baseado numa decomposição em que, dado um grafo G, o grafo

Tipos de grafos	$t = 2$	$t \geq 3$
bipartido	P [Cai94]	NP-c [Cai94, VRM+97]
bip. reg. completo		P ($t = 3$) [MVR96]
hipercubo		t-spanner H de G, onde $3\frac{n}{2} \leq \|E(H)\| \leq 4n$ [DZ00]
$\Delta \leq 4$	P [CK94]	?
$\Delta \leq 9$	NP-c [CK94, VRM+97]	NP-c [CK94, VRM+97]
cordal	NP-d [VRM+97]; APROX [PS89]	NP-d [VRM+97]; APROX ($t = 3, 5$) [PS89]
split	NP-d [VRM+97]	P [VRM+97]
intervalo	? ; 2-aproximação [VRM+97]	P ($t = 3$) [MVR96]
permutação	?	P ($t = 3$) [MVR96]
planar	?	? ($t = 3, 4$); NP-c ($t \geq 5$) [BH97]; EPTAS ($t \geq 1$) [DFG11a]

APROX (Alg. Aproximação); NP-d (NP-difícil); NP-c (NP-completo); P (Polinomial)

Tabela 3.4: *Problema do t-spanner mínimo*

é particionado em componentes, de tamanho no máximo de $n/2$ vértices, por meio da eliminação de arestas que são internas a uma *bola* B de raio proporcional ao fator de dilatação ótimo γ. Para cada componente, uma árvore geradora é construída (de maneira recursiva), e estas são unidas por meio de uma árvore de caminhos mínimos construída a partir de B. Seja uv uma aresta pertencente ao grafo de entrada. A razão de aproximação é garantida provando que, para cada nível de recursão, o número de arestas adicionadas entre u e v na solução final é $O(\gamma)$.

Alguns resultados para classes específicas de grafos, tanto para o problema da árvore t-spanner como para o problema de t-spanner mínimo, foram desenvolvidos durante este período, e são listados nas Tabelas 3.2 e 3.4 respectivamente. Dentre estes resultados, os autores em [BDL+07] estudaram o problema da árvore t-spanner com peso unitário para as classes de grafo bipartido e intervalo (mais especificamente, subclasses destas classes). Dentre os resultados apresentados, eles mostram que, para $t \geq 5$, decidir se um grafos bipartido cordal admite uma árvore t-spanner é NP-completo. Até então, o resultado de NP-completude valia para a classe mais geral dos grafos bipartidos (a partir dos resultados apresentados em [CC95]). Outros resultados para o problema da árvore t-spanner e suas variantes são apresentados em [LW08].

3.3.5 2010 – 2014: algoritmos de aproximação baseados em PL

Os trabalhos [DK11a, DK11b] consideram digrafos. Os autores em ambos os trabalhos utilizaram *Programação Linear* (PL) para gerar algoritmos de aproximação. Dinitz e Krauthgamer [DK11a] propuseram o primeiro algoritmo de aproximação para o problema de t-spanner de peso mínimo para o caso arbitrário. O interessante é que eles mostraram que

o fator de aproximação não precisa crescer com o parâmetro t. Para $t = 3$ e considerando o caso unitário, os autores propuseram um algoritmo diferente (do algoritmo para o caso arbitrário) que gera uma aproximação ainda melhor e cujo fator coincide com o caso não direcionado. Eles estenderam o trabalho para o caso de tolerância a falhas, sendo o primeiro trabalho proposto de tolerância a falhas para digrafos. Eles resolvem o PL por meio da resolução do problema dual, que se reduz ao problema de caminho mínimo restrito (*Constrained Shortest Path Problem*). Para este problema, é sabido que existe uma PTAS. Por meio disso, os autores conseguem gerar um $(1+\epsilon)$-aproximação para o PL original. A partir da resolução do PL, o algoritmo para gerar uma spanner é baseado na ideia de separar os pares (u, v) de vértices entre aqueles que possuem poucos caminhos viáveis (caminhos que respeitam a restrição de spanner) e os que possuem muitos caminhos viáveis. O números de caminhos é baseado no número de vértices que fazem parte dos caminhos. Para os pares (u, v) que possuem poucos caminhos, aplica-se um arredondamento nas variáveis fracionárias para garantir que pelo menos um caminho fará parte da solução final. Para os pares (u, v) que possuem muitos caminhos viáveis, são criadas e unidas duas árvores de caminhos mínimos.

Outros trabalhos que utilizam PL para gerar algoritmos de aproximação são [KP98, BRR10, CDK12, BBM+13, BBG+14].

Para o problema da árvore t-spanner, Narayanaswamy e Ramakrishna [NR15] propuseram um algoritmo de tempo linear para a classe dos grafos exoplanares. Os autores propuseram uma redução (tempo linear) do problema da árvore t-spanner em grafos exoplanares para o problema S-partição em árvores, definido da seguinte forma: dados uma árvore T, uma funcão peso $w : V(T) \to \mathbb{N}$, um conjunto $S \subseteq V(T)$ e um $t \in \mathbb{N}$, o objetivo é saber se existe uma partição de $V(T)$ nos conjuntos $V_1, \ldots, V_{|S|}$ tal que para $1 \leq i \leq |S|$, $T[V_i]$ é conexo, $|V_i \cap S| = 1$ e $\sum_{v \in V_i} w(v) \leq t$. Os autores então mostraram uma redução de tempo linear do problema de S-partição em árvores para o problema de partição da oferta e demanda em árvore, definido da seguinte forma: dados uma árvore T tal que $V(T) = V_s \cup V_d$ e $V_s \cap V_d = \emptyset$, uma função oferta $s : V_s \to \mathbb{R}_{\geq 0}$ e uma função demanda $d : V_d \to \mathbb{R}_{\geq 0}$, o objetivo é saber se existe uma partição de $V(T)$ nos conjuntos $V_1, ..., V_k$, onde $k = |V_s|$, tal que para $1 \leq i \leq k$, $T[V_i]$ é conexo, V_i contém exatamente um vértice u de V_s e $\sum_{v \in V_i \setminus V_s} d(v) \leq s(u)$. Em decorrência destas duas reduções serem feitas em tempo linear e do fato de existir um algoritmo linear para decidir o problema de partição da oferta e demanda em árvore [IZN02], segue que o algoritmo apresentado é de tempo linear.

Dado um grafo G que admite uma árvore t-spanner, Dragan e Kohler [DK14] propuseram um algoritmo de tempo polinomial que produz uma árvore $(2\lceil t/2 \rceil \lfloor \log n \rfloor)$-spanner. Mais do que isto, os autores provaram uma condição necessária para que o grafo admita uma árvore t-spanner baseada em existência de decomposição arbórea RS (*Robertson-Seymour*) com certas propriedades. Inicialmente, os autores propuseram um algoritmo para grafos cordais. Provaram que se o grafo admite uma decomposição arbórea RS, então o grafo possui um $(2\rho\lfloor \log n \rfloor)$-spanner, onde ρ é o tamanho máximo da largura da decomposição

arbórea. Comparado com a solução proposta por Emek e Peleg [EP08], uma variante do algoritmo proposto por Dragan e Kohler garante um fator de aproximação menor para uma mesma complexidade de tempo, e uma outra variante do algoritmo possui o mesmo fator de aproximação só que com uma complexidade de tempo menor.

Papoutsakis [Pap13] investigou para cada t (inteiro) propriedades de grafos que admitem uma única árvore t-spanner. Para $t \leq 2$, é possível decidir em tempo polinomial se um grafo admite uma única árvore t-spanner; mas nada se sabe para $t \geq 3$. Outro estudo feito por este autor foi verificar como as árvores t-spanner, para o mesmo t, estão relacionadas. Por meio deste estudo, o autor concluiu que o caso $t = 3$ corresponde a um caso excepcional (em termos de complexidade computacional é o único caso do problema clássico que está em aberto até hoje). Para este caso, existem indícios que o problema seja NP-completo bem como há indícios de que ele seja fácil. A suspeita de que ele possa ser fácil decorre do fato de que, para $t = 3$, verificar se um grafo admite uma árvore t-spanner de diâmetro no máximo $t + 1$ é fácil e, para $t \neq 3$, a complexidade computacional deste problema é igual à do problema da árvore t-spanner. Por outro lado, o problema de 3-spanner se assemelha ao problema de 4-spanner, sendo que este último é NP-completo. Papoutsakis [Pap13] também caracterizou uma família de grafos que não admitem árvores 3-spanner, utilizando formulações lineares inteiras, e também apresentou uma série de problemas cujas complexidades computacionais estão em aberto.

3.3.6 2015 – 2018: spanners distribuídos, oráculos de distância e spanners esparsos

A partir de 2015, muitos trabalhos focaram a construção de spanners distribuídos [KNPR15, CHKPY16, EN17, GP17]. A motivação surge quando a quantidade de dados das novas aplicações é alta a ponto de inviabilizar o armazenamento centralizado de todos os dados. Nestes spanners distribuídos, assume-se que o grafo de entrada não é conhecido de maneira completa por nenhum dos vértices.

Como já foi dito anteriormente, a esparsidade de um spanner pode ser medida em termos da quantidade de arestas ou do peso do spanner. Os trabalhos em [ENS15, CWN16] melhoram a análise do algoritmo clássico proposto em [ADD+93], garantindo um fator de aproximação menor para o peso do spanner. Os autores em [BW15, EN17] focaram no desenvolvimento de spanners com baixa quantidade de arestas. Outros trabalhos relacionados à esparsidade são [BW15, BLWN17].

Outro cenário em que o problema foi tratado consiste no desenvolvimento de oráculos de distância [EP16, Som16, Knu17].

Capítulo 4

Formulações lineares inteiras para o problema da árvore t-spanner de peso mínimo

Neste capítulo abordamos o problema da árvore t-spanner de peso mínimo (MWTSP). Mais especificamente, apresentamos duas formulações lineares inteiras que propusemos para o MWTSP. Estas são as primeiras formulações específicas para este problema (no sentido de não serem formulações para o problema de t-spanners adaptadas para árvores). Inicialmente mencionamos outras definições equivalentes à definição de t-spanner apresentada no Capítulo 1, uma das quais será utilizada nas formulações apresentadas neste e no próximo capítulo. Depois apresentamos uma heurística que pode ser utilizada no caso de peso unitário. Posteriormente, apresentamos uma formulação linear baseada no uso de arborescências. Na Seção 4.4, apresentamos outra formulação linear para o MWTSP também baseada no uso de arborescências, mas com variáveis específicas que capturam a noção de distância.

4.1 Outras definições equivalentes de t-spanner

Cai e Corneil [CC95] mostraram algumas definições de t-spanner equivalentes à definição apresentada no Capítulo 1. Para a formulação que apresentamos na Seção 5.2, os seguintes resultados são úteis.

Lema 4.1. *Seja $G = (V, E)$ um grafo conexo com peso $w : E \to \mathbb{R}_{\geq 0}$, e $t \geq 1$ um número real. Seja H um subgrafo gerador de G. As seguintes afirmações são equivalentes:*

(a) *H é um t-spanner de G, isto é, H satisfaz (3.1);*

(b) $\text{dist}_H(u, v) \leq t \cdot \text{dist}_G(u, v)\ \forall\, uv \in E$;

(c) $\text{dist}_H(u, v) \leq t \cdot w_{uv}\ \forall\, uv \in E$.

Demonstração. A prova de que (a) implica (b) é direta. Vamos provar que (b) implica (a). Isto é,

$$\begin{aligned}\text{dist}_H(u,v) \leq t \cdot \text{dist}_G(u,v)\ \forall\, uv \in E \implies \\ \text{dist}_H(u,v) \leq t \cdot \text{dist}_G(u,v)\ \forall\, u,v \in V.\end{aligned}$$

Sejam $u, v \in V$. É suficiente provarmos a desigualdade para o caso em que $uv \notin E$. Seja $P = (u = u_0, u_1, \ldots, u_{\ell-1}, u_\ell = v)$ um caminho de peso mínimo entre u e v em G. Por hipótese, para cada $u_{i-1}u_i \in E(P)$, temos que $\text{dist}_H(u_{i-1}, u_i) \leq t \cdot \text{dist}_G(u_{i-1}, u_i)$. Então

$$\begin{aligned}\text{dist}_H(u,v) &\leq \sum_{i=1}^{\ell} dist_H(u_{i-1}, u_i) \\ &\leq \sum_{i=1}^{\ell} t \cdot \text{dist}_G(u_{i-1}, u_i) \\ &= t \cdot \sum_{i=1}^{\ell} \text{dist}_G(u_{i-1}, u_i) \\ &= t \cdot \text{dist}_G(u,v),\end{aligned}$$

onde a última igualdade segue do fato de que a distância entre u e v em G é igual à soma das distâncias entre os vértices consecutivos que compõem um caminho de peso mínimo entre u e v em G.

A prova de que (a) implica (c) é imediata. Vamos provar que (c) implica (a). Considere $u, v \in V$, e tome um caminho de peso mínimo de u a v, digamos P, em G. Seja xy uma aresta de P:

- Se $xy \in E(H)$, então $\text{dist}_H(x,y) \leq w_{xy} \leq t \cdot w_{xy}$;
- Se $xy \notin E(H)$, então $\text{dist}_H(x,y) \leq t \cdot w_{xy}$ (por hipótese).

Portanto, $\text{dist}_H(u,v) \leq \sum_{xy \in E(P)} dist_H(x,y) \leq t \cdot \sum_{xy \in E(P)} w_{xy} = t \cdot \text{dist}_G(u,v)$. □

4.2 Heurística para o caso unitário

Dado um grafo $G = (V, E)$, onde $|V| = n$, uma função peso $w : E \to \mathbb{Z}_{\geq 0}$, e um inteiro $2 \leq D \leq n-2$, o problema da *árvore geradora mínima com diâmetro limitado* (do inglês *Bounded Diameter Minimum Spanning Tree Problem* - BDMSTP) tem por objetivo encontrar uma árvore geradora de peso mínimo cujo número de arestas no caminho entre quaisquer dois vértices seja no máximo D. É sabido que a versão de decisão deste problema é NP-completo [GJ90].

Se o peso das arestas é unitário, o BDMSTP pode ser resolvido em tempo polinomial (no tamanho de G) [GJ90]. Neste caso, temos uma heurística que, caso seja bem-sucedida

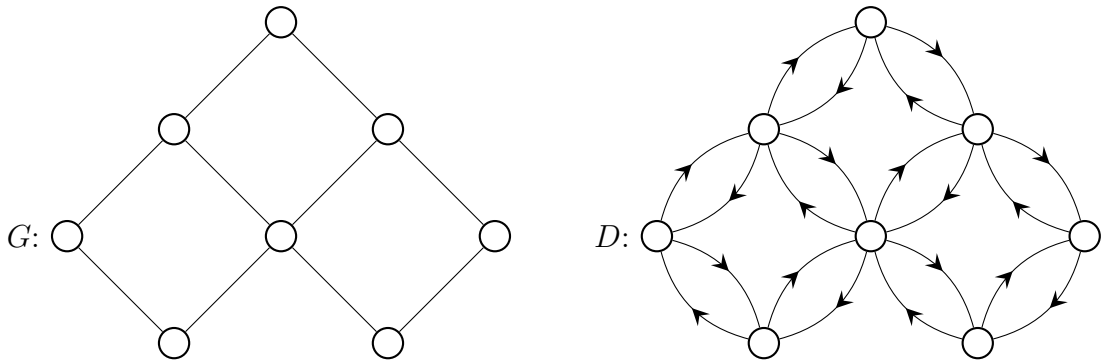

Figura 4.1: *Grafo G e o correspondente digrafo D*

na busca de uma solução viável, tal solução é ótima para o MWTSP (considerando $t = D$). Para resolver o BDMSTP no caso unitário, podemos executar o algoritmo de busca em largura [CLRS09] a partir de cada um dos vértices de G até que, para um determinado vértice, a altura da árvore de busca (de G) não seja maior do que $h = \lfloor D/2 \rfloor$. Caso h seja ímpar, consideramos cada dois vértices (adjacentes) ao invés de um como centro, realizamos a contração da aresta representada pelos dois vértices e executamos o algoritmo de busca em largura a partir deste vértice criado. Havendo solução, a árvore final deverá conter a aresta contraída.

4.3 Formulação SR: encontrando distâncias entre os vértices

Nesta seção, vamos apresentar uma formulação linear inteira para o problema de árvores t-spanner baseada no uso de arborescências. Para trabalhar com arborescências, a partir do grafo dado G construímos inicialmente um digrafo D. Fazendo uso de arborescências enraizadas em cada um dos vértices do digrafo D, podemos impor condições para que essas arborescências, quando "sobrepostas", definam uma árvore geradora em G. Isto nos permite distinguir (facilmente), para cada par u, v de vértices, quais são as arestas que pertencem ao caminho entre u e v em tal árvore. Isto ficará mais claro a seguir.

Dado um grafo conexo $G = (V, E)$, seja $D = (V, A)$ o digrafo obtido a partir de G, onde $A = \{(u, v), (v, u) \mid uv \in E\}$ (veja Figura 4.1). Fixe um vértice $v \in V$ (raiz), considere $z^v = (z^v_{ij})_{ij \in A}$, e as seguintes restrições:

$$\sum_{i \in \delta^-(j)} z^v_{ij} = 1 \qquad \forall j \in V \setminus \{v\} \tag{4.1}$$

$$\sum_{i \in \delta^-(v)} z^v_{iv} = 0 \tag{4.2}$$

$$z^v_{ij} \in \{0, 1\} \qquad \forall ij \in A \tag{4.3}$$

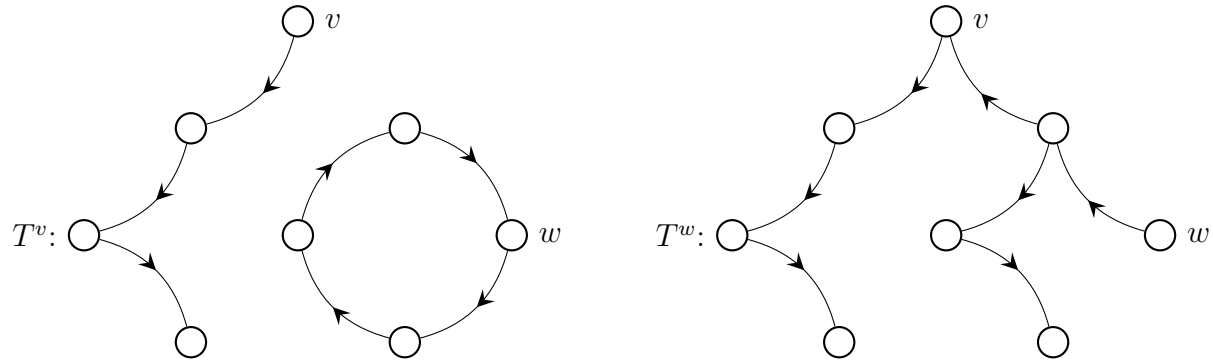

Figura 4.2: *Subgrafos T^v e T^w*

A restrição (4.1) impõe que para cada vértice j distinto de v seja selecionado exatamente um arco que chega em j. A restrição (4.2) impõe que não seja selecionado nenhum arco que chega em v.

Seja $\tilde{z}^v$ um ponto que satisfaz o sistema acima. Seja $T^v \subseteq D$ tal que $A(T^v) = \{ij \in A \mid \tilde{z}^v_{ij} = 1\}$, ou seja, T^v é o digrafo de D formado pelos arcos selecionados, conforme as restrições acima. É imediato que vale o seguinte resultado.

Fato 4.1. *O subgrafo T^v tem exatamente $|V| - 1$ arcos.*

Figura 4.2 ilustra possibilidades para T^v e T^w, onde v e u são vértices de V.

Estamos interessados nos subconjuntos $F \subseteq E$ tais que $G[F]$ é uma árvore t-spanner de G. Em outras palavras, o poliedro das árvores t-spanners de um grafo $G = (V, E)$, para um número real $t \geq 1$, é definido como

$$P_{tree}(G, t) := \text{conv}\{\mathcal{X}^F \in \mathbb{R}^{|E|}, F \subseteq E \mid G[F] \text{ é uma árvore } t\text{-spanner}\}.$$

Vamos agora apresentar uma formulação linear inteira que descreve o poliedro $P_{tree}(G, t)$. Para isso, associamos a cada aresta e em E uma variável x_e com o seguinte significado: $x_e = 1$ se e só se e faz parte da solução.

As restrições sobre $x = (x_e)_{e \in E}$ são impostas fazendo uso das restrições vistas anteriormente — para os subgrafos T^v —, definidas para cada vértice v do grafo G.

$$\sum_{i \in \delta^-(j)} z^v_{ij} = 1 \qquad \forall v \in V, \forall j \in V \setminus \{v\} \tag{4.4}$$

$$\sum_{i \in \delta^-(v)} z^v_{iv} = 0 \qquad \forall v \in V \tag{4.5}$$

$$x_e = z^v_{ij} + z^v_{ji} \qquad \forall v \in V, \forall e = ij \in E \tag{4.6}$$

$$z^v_{ij} \in \{0, 1\} \qquad \forall v \in V, ij \in A \tag{4.7}$$

$$x_e \in \{0, 1\} \qquad \forall e \in E \tag{4.8}$$

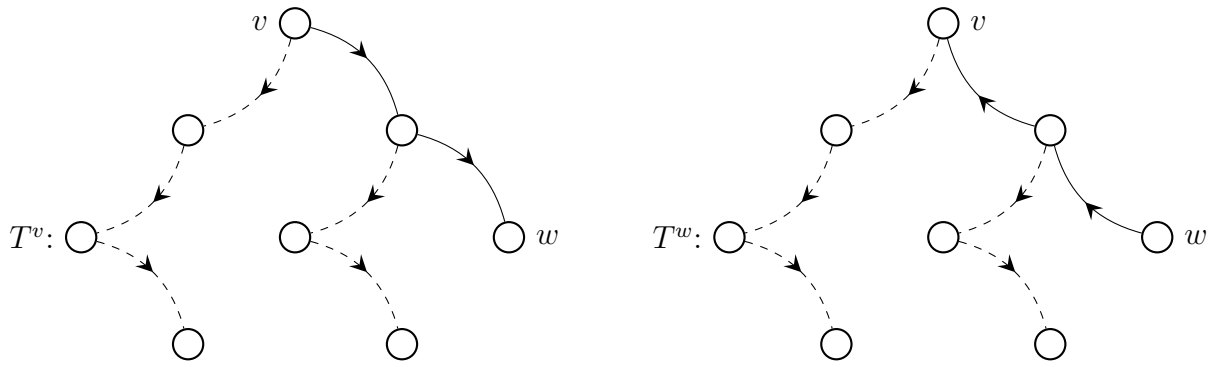

Figura 4.3: *Arborescências T^u e T^v se sobrepondo*

Este sistema linear foi inspirado no trabalho de Conforti, Cornuéjols e Zambelli [CCZ13] que trata do problema de representar o fecho convexo dos vetores de inciência das árvores geradoras de um grafo.

Seja $(\tilde{x}, \tilde{z})$ uma solução do sistema acima. Como definimos anteriormente, para cada $v \in V$, seja T^v o digrafo tal que $A(T^v) = \{ij \in A \mid \tilde{z}^v_{ij} = 1\}$, e seja $\widetilde{T}^v \subseteq G$ o grafo subjacente a T^v tal que $E(\widetilde{T}^v) = \{ij \in E \mid ij \in A(T^v) \text{ ou } ji \in A(T^v)\}$.

Não é difícil verificar as seguintes afirmações:

$$\widetilde{T}^v = \widetilde{T}^w, \ \forall v, w \in V; \tag{4.9}$$

$$T^v \text{ é uma arborescência de } D, \text{ com raiz } v, \ \forall v \in V. \tag{4.10}$$

A afirmação (4.9) é garantida pelas restrições (4.6).

Suponha que a afirmação (4.10) não seja correta. Suponha que exista um vértice $u \in V$ para o qual exista um circuito em $\widetilde{T}^u$. Note que u não está contido em nenhum circuito de $\widetilde{T}^u$, caso contrário violaria a restrição (4.5). Seja $v \in V$ tal que v pertence a um circuito C_v de $\widetilde{T}^u$. Em T^v, não existem dois arcos entrando em v, caso contrário a restrição (4.5) seria violada. Se considerarmos os dois caminhos direcionados a partir de v em $\{ij \in A(T^v) \mid \{i, j\} \in E(C_v)\}$, existirá um vértice $w \in V(C_v)$ tal que ambos os caminhos se encontram em w. Então a restrição (4.4) é violada para o vértice w, uma contradição.

A partir das observações feitas acima, podemos concluir o seguinte fato relacionado ao sistema linear apresentado anteriormente.

Fato 4.2. *Seja $T \subseteq G$ tal que $E(T) = \{e \in E \mid x_e = 1\}$. Então T é uma árvore geradora de G, e $T = \widetilde{T}^v$ para todo $v \in V$.*

Figura 4.3 mostra duas arborescências que se "sobrepõem".

Como dito anteriormente, a formulação acima foi feita com o propósito de facilitar distinguir, para cada par de vértices u, v em G, quais são as arestas que pertencem ao caminho de u a v na árvore T (acima definida). Precisamos identificar tais caminhos para impor a condição sobre o fator de dilatação. Para isso, vamos definir novas variáveis de decisão.

É importante observar que, como todas as arborescências têm como grafo subjacente a árvore T, quando tomamos dois vértices distintos em T, digamos u e v, as arborescências T^u e T^v diferem apenas na orientação dos arcos no caminho de u a v. Mais precisamente, na arborescência T^u há um caminho orientado de u a v; e na arborescência T^v há um caminho orientado de v a u. Os arcos que não pertencem ao caminho de u a v têm a mesma orientação tanto em T^u quanto em T^v. Este fato (ilustrado na Figura 4.3) é central para as as restrições que apresentaremos a seguir.

Para cada aresta $f \in E$, considere $y^f = (y^f_e)_{e \in E}$. A variável $y \in \mathbb{R}^{|E|\times|E|}_{\geq 0}$ tem o seguinte significado: para cada aresta $e \in E$, e cada aresta $f = uv \in E$, queremos que $y^f_e = 1$ se e somente se a aresta e pertence ao caminho entre u e v em T.

Note que na definição da variável y, só consideramos os pares de vértices que definem aresta, em vez de considerarmos todos os pares de vértices. Estamos usando aqui a equivalência garantida no Lema 4.1.

Para todo $uv, ij \in E$, considere as seguintes definições, feitas para poder distinguir a pertinência de ij nas arborescências T^u e T^v:

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

Tabela 4.1: *Diferenças e somas relativas à variável z*

d^{uv}_{ij}	s^{uv}_{ij}	z^u_{ij}	z^v_{ij}	z^u_{ji}	z^v_{ji}	d^{uv}_{ji}	s^{uv}_{ji}
1	1	1	0	0	1	−1	1
−1	1	0	1	1	0	1	1
0	2	1	1	0	0	0	0
0	0	0	0	1	1	0	2
0	0	0	0	0	0	0	0

As quatro primeiras linhas da Tabela 4.1 representam os casos nos quais $ij \in E(T)$, enquanto que na quinta linha, $ij \notin E(T)$. Para $u, v \in V(T)$, seja $T_{u,v}$ o caminho entre u e v em T. Nas duas primeiras linhas, $ij \in E(T_{u,v})$, enquanto que na terceira e na quarta, $ij \notin E(T_{u,v})$ Considerando o caminho $T_{u,v}$, os valores apresentados nas quatro primeiras linhas se devem a dois fatos: cada arco em T^u (referente a este caminho) está no sentido contrário do arco análogo em T^v; os demais arcos de T^u estão no mesmo sentido que os análogos em T^v.

Considere as seguintes inequações:

$$d_{ij}^{uv} \leq y_e^{uv} \leq s_{ij}^{uv} \qquad \forall uv \in E, \forall e = \{i,j\} \in E \tag{4.11}$$

$$d_{ji}^{uv} \leq y_e^{uv} \leq s_{ji}^{uv} \qquad \forall uv \in E, \forall e = \{i,j\} \in E \tag{4.12}$$

Pela Tabela 4.1 e pelas inequações (4.11) e (4.12), podemos concluir que para todo $uv \in E$ e todo $e \in E$, temos que

$$y_e^{uv} = 1 \Leftrightarrow e \in E(T_{u,v}). \tag{4.13}$$

Juntando todas a restrições mencionadas, obtemos a seguinte formulação, denominada *sem rótulos* (SR), para o MWTSP.

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

s.a

$$\sum_{e \in E} x_e = |V| - 1 \tag{4.14}$$

$$\sum_{i \in \delta^-(j)} z_{ij}^v = 1 \qquad \forall v \in V, \forall j \in V \setminus \{v\} \tag{4.15}$$

$$\sum_{i \in \delta^-(v)} z_{iv}^v = 0 \qquad \forall v \in V \tag{4.16}$$

$$x_e = z_{ij}^v + z_{ji}^v \qquad \forall v \in V, \forall e = \{i,j\} \in E \tag{4.17}$$

$$z_{ij}^u - z_{ij}^v \leq y_e^{uv} \leq z_{ij}^u + z_{ij}^v \qquad \forall uv \in E, \forall e = \{i,j\} \in E \tag{4.18}$$

$$z_{ji}^u - z_{ji}^v \leq y_e^{uv} \leq z_{ji}^u + z_{ji}^v \qquad \forall uv \in E, \forall e = \{i,j\} \in E \tag{4.19}$$

$$\sum_{e \in E} w_e \, y_e^{uv} \leq t \cdot w_{uv} \qquad \forall uv \in E \tag{4.20}$$

$$x_e \in \{0,1\} \qquad \forall e \in E \tag{4.21}$$

$$y_e^f \in \{0,1\} \qquad \forall f \in E, \forall e \in E \tag{4.22}$$

$$z_{ij}^v \in \{0,1\} \qquad \forall v \in V, \forall ij \in A \tag{4.23}$$

Em decorrência do Fato 4.2, a restrição (4.14) é redundante.

4.4 Formulação CR: com variáveis que representam distâncias entre pares de vértices

Assim como na formulação anterior, vamos apresentar uma formulação baseada na ideia de arborescências que se sobrepõem. A diferença é que esta formulação conterá rótulos específicos com informações que representam distâncias entre pares de vértices. Dado um grafo conexo $G = (V, E)$, seja $D = (V, A)$ o digrafo obtido a partir de G, onde $A = \{(u,v), (v,u) \mid uv \in E\}$. Fixe um vértice $v \in V$ (raiz) e considere $z^v = (z_{ij}^v)_{ij \in A}$. Nesta formulação linear inteira, as soluções também serão

representadas pelos vetores de incidência das árvores t-spanner. Para isto, vamos também associar à cada aresta $e \in E$ uma variável x_e com o seguinte significado: $x_e = 1$ se e só se a aresta e faz parte da solução.

$$\sum_{i \in \delta^-(j)} z^v_{ij} = 1 \qquad \forall v \in V, \forall j \in V \setminus \{v\} \tag{4.24}$$

$$\sum_{i \in \delta^-(v)} z^v_{iv} = 0 \qquad \forall v \in V \tag{4.25}$$

$$x_e = z^v_{ij} + z^v_{ji} \qquad \forall v \in V, \forall e = ij \in E \tag{4.26}$$

$$z^v_{ij} \in \{0, 1\} \qquad \forall v \in V, ij \in A \tag{4.27}$$

$$x_e \in \{0, 1\} \qquad \forall e \in E \tag{4.28}$$

Seja (x, z) uma solução do sistema acima. Como definimos na formulação anterior, para cada $v \in V$, seja T^v o digrafo tal que $A(T^v) = \{ij \in A \mid \tilde{z}^v_{ij} = 1\}$, e seja $\widetilde{T}^v \subseteq G$ o grafo subjacente a T^v tal que $E(\widetilde{T}^v) = \{ij \in E \mid ij \in A(T^v) \text{ ou } ji \in A(T^v)\}$. Pelo Fato (4.2), seja $T \subseteq G$ tal que $E(T) = \{e \in E \mid x_e = 1\}$. Então T é uma árvore geradora de G, e $T = \widetilde{T}^v$ para todo $v \in V$.

As observações sobre os caminhos entre dois vértices u e v nas respectivas arborescências T^u e T^v são as mesmas apresentadas na formulação anterior.

Vamos agora definir variáveis que representarão distâncias entre pares de vértices de V. Seja $u \in \mathbb{R}^{|V|\times|V|}_{\geq 0}$. Para $r, j \in V$, o papel da variável u^r_j é expressar a distância entre os vértices r e j na arborescência T^r. Para capturar essas distâncias na arborescência, fazemos uso de uma constante M^r_{ij} que corresponde a um limite superior para a diferença $u^r_i - u^r_j$. Na inequação (4.29), para o caso unitário, como $j \neq r$, um limite superior pode ser $n - 2$. Para o caso com pesos, um limite seria $t \cdot dist_G(i, j)$. Na inequação (4.30), a constante M^r_{ir} corresponde a um limite superior para u^r_i. Para o caso de peso unitário, um limite superior pode ser $n - 1$. Para o caso de pesos arbitrários, um limite seria $t \cdot dist_G(r, i)$. A definição dos limites ficará clara mais adiante.

$$u^r_i - u^r_j + (M^r_{ij} + w_{ij})z^r_{ij} + (M^r_{ij} - w_{ij})z^r_{ji} \leq M^r_{ij} \qquad \forall r \in V, \forall ij \in A, j \neq r \tag{4.29}$$

$$u^r_i + (M^r_{ir} - w_{ir})z^r_{ri} \leq M^r_{ir} \qquad \forall r \in V, \forall ri \in A \tag{4.30}$$

$$u^r_r = 0 \qquad \forall r \in V \tag{4.31}$$

As restrições (4.29) e (4.30) foram inspiradas no trabalho de Miller, Tucker e Zemlin [MTZ60] que trata do problema do caixeiro viajante (TSP).

Seja $r \in V$. Seja $ij \in A$ tal que $z^r_{ij} = 1$ e considere tal arco na restrição (4.29) (observe que $j \neq r$). Pela restrição (4.29) concluímos que $u^r_j \geq u^r_i + w_{ij}$. Considere agora, para o caso de $i \neq r$, o arco ji na restrição (4.29). Se $i = r$, considere o arco rj na restrição (4.30) (observe que, neste caso, a variável z^r_{ri} na inequação (4.30) tem valor um). Então, restrições (4.29) e (4.30) mostram que $u^r_j = u^r_i + w_{ij}$ quando $z^r_{ij} = 1$. Figura 4.4 ilustra a definição da variável u^r. Considere a arborescência T^r e a aresta $ij \in E$. Observe a inequação (4.29) com relação ao arco ij (lembre-se que $z^r_{ij} = 1$, ou seja, $z^r_{ji} = 0$):

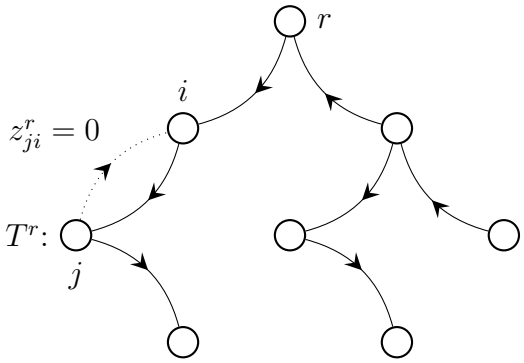

Figura 4.4: *Arborescência T^r para ilustrar o significado da variável u^r*

$$u^r_i - u^r_j + (M^r_{ij} + w_{ij}) \cdot 1 + (M^r_{ij} - wij) \cdot 0 \leq M^r_{ij} \Rightarrow u^r_j \geq u^r_i + w_{ij}.$$

Agora vamos ver a inequação (4.29) com relação ao arco ji (lembre-se que $z^r_{ij} = 1$, ou seja, $z^r_{ji} = 0$):

$$u^r_j - u^r_i + (M^r_{ji} + w_{ij}) \cdot 0 + (M^r_{ji} - w_{ij}) \cdot 1 \leq M^r_{ji} \Rightarrow u^r_j \leq u^r_i + w_{ij}.$$

As duas desigualdades resultam na igualdade $u^r_j = u^r_i + w_{ij}$.

Logo, podemos concluir que para $j \in V$, o valor de u^r_j corresponde exatamente à distância entre r e j em T^r.

A validade de (4.29) pode ser vista das seguintes implicações:

- $z^r_{ij} = 1, z^r_{ji} = 0 \Rightarrow u^r_i + w_{ij} \leq u^r_j$;
- $z^r_{ij} = 0, z^r_{ji} = 1 \Rightarrow u^r_i \leq u^r_j + w_{ij}$;
- $z^r_{ij} = 0, z^r_{ji} = 0 \Rightarrow u^r_i - u^r_j \leq M^r_{ij}$.

A validade de (4.30) segue das seguintes implicações:

- $z^r_{ij} = 1 \Rightarrow u^r_i \leq w_{ij}$;
- $z^r_{ij} = 0 \Rightarrow u^r_i \leq M^r_{ir}$.

Juntando todas as restrições mencionadas, obtemos a seguinte formulação, denominada *com rótulos* (CR), para o MWTSP:

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

s.a

$$\sum_{e \in E} x_e = |V| - 1 \tag{4.32}$$

$$\sum_{i \in \delta^-(j)} z^r_{ij} = 1 \qquad \forall r \in V, \forall j \in V \setminus \{r\} \tag{4.33}$$

$$\sum_{i \in \delta^-(r)} z^r_{ir} = 0 \qquad \forall r \in V \tag{4.34}$$

$$x_e = z^r_{ij} + z^r_{ji} \qquad \forall r \in V, \forall e = \{i, j\} \in E \tag{4.35}$$

$$u^r_i - u^r_j + (M^r_{ij} + w_{ij}) z^r_{ij} + (M^r_{ij} - w_{ij}) z^r_{ji} \leq M^r_{ij} \qquad \forall r \in V, \forall ij \in A, j \neq r \tag{4.36}$$

$$u^r_i + (M^r_{ir} - w_{ir})) z^r_{ri} \leq M^r_{ir} \qquad \forall r \in V, \forall ri \in A \tag{4.37}$$

$$u^r_r = 0 \qquad \forall r \in V \tag{4.38}$$

$$u^j_i = u^i_j \leq t \cdot w_{ij} \qquad \forall ij \in E \tag{4.39}$$

$$x \in \mathbb{B}^{|E|}, z^r \in \mathbb{B}^{|E| \times |E|}, u^r \in \mathbb{R}^{|V|} \qquad \forall r \in V \tag{4.40}$$

Na restrição (4.39), para $ij \in E$, como sabemos que $E(\widetilde{T}^i) = E(\widetilde{T}^j)$ pelo Fato 4.2, podemos impor a igualdade $u^j_i = u^i_j$; a restrição de desigualdade é óbvia.

4.4.1 Inequações válidas

Quando as arestas têm peso unitário, a seguinte inequação é válida:

$$u^r_v + z^r_{rv} \geq 2, \ \forall r \in V, \forall v \in V \setminus \{r\}. \tag{4.41}$$

A inequação (4.41) diz que para um par de vértices r, v que define uma aresta, ou a aresta uv pertencerá à solução ou o caminho entre r e v na solução será maior ou igual a dois.

Finalizamos este capítulo mencionando que as duas formulações lineares inteiras para o MWSTP, acima apresentadas, constam do trabalho [Bra17], que apresentamos no *II Encontro de Teoria da Computação (ETC 2017)*.

Capítulo 5

Problema de t-spanner de peso mínimo

Neste capítulo apresentamos uma formulação linear inteira que propusemos para o problema de t-spanner de peso mínimo (MWSP).

5.1 Poliedro dos t-spanners de um grafo

No problema do t-spanner de peso mínimo (MWSP), dado um grafo $G = (V, E)$ com peso $w : E \to \mathbb{R}_{\geq 0}$, e um número real $t \geq 1$, queremos encontrar um conjunto $F \subseteq E$ de peso mínimo tal que $G[F]$ é um t-spanner de G. Para isso, definimos o poliedro dos t-spanners de G, objetivando usar técnicas de programação linear para encontrar uma solução ótima. Esse poliedro é assim definido.

$$P_{span}(G,t) := \text{conv}\{\mathcal{X}^F \in \mathbb{R}^{|E|}, F \subseteq E \mid G[F] \text{ é um } t\text{-spanner de } G\}.$$

Observe que se $t'' \geq t' \geq 1$, então $P_{span}(G,t') \subseteq P_{span}(G,t'')$.

5.1.1 Dimensão do poliedro

Seja $\mathcal{B}^t(G) = \{e \in E \mid e \text{ é uma aresta } t\text{-essencial de } G\}$. Lembramos que uma aresta e de G é t-essencial se $G - e$ não é um t-spanner de G.

No que segue, supomos que $|E| = m$, $E = \{e_1, \ldots, e_m\}$ e que os vetores $x \in \mathbb{R}^{|E|}$ estão indexados por E, ou seja, $x = (x_{e_1}, \ldots, x_{e_m})$.

Lema 5.1. *$dim(P_{span}(G,t)) = |E| - |\mathcal{B}^t(G)|$.*

Demonstração. Seja $f \in \mathcal{B}^t(G)$. Então, para cada $\mathcal{X}^F \in P_{span}(G,t)$, segue que $f \in F$. Sendo assim, $P_{span}(G,t) \subseteq \{x \in \mathbb{R}^{|E|} | \, x_f = 1\}$. Logo, $dim(P_{span}(G,t)) \leq |E| - |\mathcal{B}^t(G)|$.

Como G é t-spanner, $\mathcal{X}^E \in P_{span}(G,t)$ e $\mathcal{X}^{E-f} \in P_{span}(G,t), \forall f \in E \setminus \mathcal{B}^t$. Claramente tais vetores são afim-independentes, e portanto, $dim(P_{span}(G,t)) \geq |E| - |\mathcal{B}^t(G)|$. □

5.2 Outro poliedro associado ao poliedro dos t-spanners

Para $G' = (V, E') \subseteq G = (V, E)$, denotamos por $\mathcal{P}_{u,v}(G')$ a coleção de caminhos t-spanner entre u e v que estão contidos em G'.

Para obter uma descrição do poliedro $P_{span}(G,t)$ definido na seção anterior, introduziremos um outro poliedro, e mostraremos como esses dois poliedros esstão relacionados.

Para definir esse novo poliedro, faremos uso de dois tipos de variáveis de decisão 0/1: (a) variável $x \in \mathbb{R}^{|E|}$ com o seguinte significado: para cada $e \in E$, $x_e = 1$ se e só se e faz parte da solução F; (b) variável $Y \in \mathbb{R}_{\geq 0}^{|E|\times|E|}$, para impor a condição de existência de caminhos t-spanner para cada aresta $e = uv$ de G. Para conseguir isso, impomos a existência de um caminho $P \in \mathcal{P}_{u,v}(G[F])$, onde $F \subseteq E$ é o conjunto das arestas selecionadas.

As três primeiras restrições do sistema abaixo, juntamente com a restrição de que x e Y são variáveis $0/1$, capturam exatamente isso que desejamos. (Para cada aresta $e = uv$ em G, garantimos a existência de um caminho P em $\mathcal{P}_{u,v}(G[F])$ que é t-spanner, e para o qual temos que $Y_{e,f} = 1$ para toda aresta $f \in E(P)$.) Nessa descrição, para $W \subset V$, denotamos por $\delta(W)$ o conjunto das arestas de G que têm uma extremidade em W e outra em $\overline{W}$ (ou seja, $\delta(W)$ define um corte em G). Além disso, para $F \subseteq E$, $Y_{e,F}$ denota a soma $\sum_{f\in F} Y_{e,f}$.

Considere o seguinte sistema linear, no qual está relaxada a condição de integralidade das variáveis.

$$
\begin{aligned}
\text{(PL)} \quad & Y_{e,\delta(W)} \geq 1 && \forall e \in E, \forall\, W \subset V \text{ com } e \in \delta(W) && (5.1)\\
& Y_{e,f} \leq x_f && \forall e, f \in E && (5.2)\\
& \sum_{f\in E} Y_{e,f} w_f \leq t \cdot w_e && \forall e \in E && (5.3)\\
& 0 \leq x_e \leq 1 && \forall e \in E && (5.4)\\
& 0 \leq Y_{e,f} \leq 1 && \forall e, f \in E && (5.5)
\end{aligned}
$$

$$P(G,t) = \{(x,Y) \mid x \in \mathbb{R}^{|E|}, Y \in \mathbb{R}_{\geq 0}^{|E|\times|E|} \text{ e } (x,Y) \text{ satisfaz (PL)}\}.$$

Seja $(P(G,t))_I$ o fecho inteiro do poliedro $P(G,t)$. Veremos a seguir como $P_{span}(G,t)$ e $(P(G,t))_I$ estão relacionados.

Note que no programa linear (PL) não há variáveis diretamente associadas aos caminhos que respeitam a restrição de t-spanner, como é comum ocorrer nas modelagens tradicionais [SZ04, DK11a]. No lugar de variáveis associadas a caminhos, usamos a variável Y (indexada por pares de arestas).

Seja (x,Y) um ponto inteiro de $P(G,t)$ e H a solução associada a (x,Y), definida como $H = G[F]$, onde $x = \mathcal{X}^F$. A restrição (5.2) diz que para $f \in E$, se $Y_{e,f} = 1$ para algum $e \in E$, então f deve fazer parte da solução H. Além disso, se $f \in E$ não faz parte da solução H, então não existe $e \in E$ tal que $Y_{e,f} = 1$.

Pelo Lema 4.1 é suficiente garantirmos que a condição de caminho t-spanner está satisfeita pelos pares de vértices $u, v \in V$ tais que $uv \in E$. Por essa razão, apenas para cada aresta definimos variáveis da forma $Y_{e,f}$, para cada $e, f \in E$.

Note que, nesta formulação, a menos das restrições (5.1), há apenas um número polinomial de restrições no tamanho de E. Veremos mais adiante que as restrições (5.1) podem ser separadas em

tempo polinomial (no tamanho de V).

A proposição a seguir mostra que a todo vértice de $P_{span}(G,t)$ corresponde um vértice de $(P(G,t))_I$.

Proposição 5.1. *Seja x' um vértice de $P_{span}(G,t)$. Então existe Y' tal que (x',Y') é um vértice de $(P(G,t))_I$.*

Demonstração. Seja $x' = \mathcal{X}^F$ um vértice de $P_{span}(G,t)$ associado ao t-spanner $G[F]$. Para cada $e = uv \in E$, considere um caminho $P_{u,v} \in \mathcal{P}_{u,v}(G[F])$. Defina $Y' \in \mathbb{R}_{\geq 0}^{|E|\times|E|}$ da seguinte forma:

$$\forall e = uv \in E, f \in E,\ \ Y'_{e,f} = \begin{cases} 1 & \text{se } f \in E(P_{u,v}), \\ 0 & \text{caso contrário.} \end{cases}$$

Claramente, (x,Y') satisfaz todas as restrições do sistema (PL), e como este é um vetor 0/1, segue que é um vértice de $(P(G,t))_I$.

Note que, pode haver vetores Y'' distintos de Y' para os quais (x',Y'') são vértices de $(P(G,t))_I$. □

A proposição a seguir mostra que a todo vértice de $(P(G,t))_I$ corresponde um vértice de $P_{span}(G,t)$.

Proposição 5.2. *Seja (x',Y') um vértice de $(P(G,t))_I$. Para $e \in E$, seja $F(e) = \{f \in E \mid Y'_{e,f} = 1\}$. Seja $E_{x'} = \{e \in E \mid x'_e = 1\}$ e $H = G[E_{x'}]$. Então*

(a) *Para cada aresta $e = uv \in E$, existe um caminho P entre u e v em $G[F(e)]$ para o qual $Y'(e,f) = 1$ para todo $f \in E(P)$.*

(b) *O subgrafo $G' = G[\bigcup_{e\in E} F(e)]$ é gerador e conexo.*

(c) *H é um t-spanner de G, e portanto x' é um vértice de $P_{span}(G,t)$.*

Demonstração. Prova do item (a). Seja $e = uv \in E$. Suponha, por absurdo, que não exista um caminho entre u e v em $G[F(e)]$. Então u e v estão em componentes distintas em $G[F(e)]$. Seja G_u o grafo induzido pelos vértices da componente de $G[F(e)]$ que contém u. Seja $W = V(G_u)$. Mas então a restrição (5.1) é violada pelo par e, W, contradizendo a hipótese de que $(x',Y') \in P(G,t)$. Concluímos então que deve existir um caminho P entre u e v em $G[F(e)]$ para o qual $Y'(e,f) = 1$ para todo $f \in E(P)$.

Prova do item (b). Sejam $u, v \in V$. Como G é conexo, existe um caminho P entre u e v em G, digamos $P = (u = u_0, u_1, \ldots, u_{l-1}, u_l = v)$. Pelo item (a) e sabendo que $G[F(e)] \subseteq G'$, para cada $e = u_{i-i}u_i \in E(P)$, existe um caminho entre u_{i-1} e u_i em $G[F(e)]$, e portanto em G'. Sendo assim, u e v estão conectados em G', donde segue que G' é um subgrafo gerador conexo de G.

Prova do item (c). Seja $e = uv \in E$. Pelo item (a), temos que $\text{dist}_{G[F(e)]}(u,v) \leq \sum_{f\in E} Y'_{e,f}\, w_f$. Como valem os itens (a) e (b) e $G[F(e)] \subseteq G'$, temos que $\text{dist}_{G'}(u,v) \leq \text{dist}_{G[F(e)]}(u,v)$. Pela

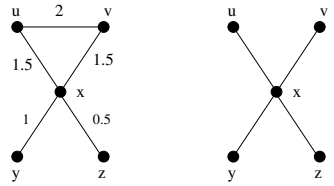

Figura 5.1: *Um grafo com seu respectivo 2-spanner de peso mínimo*

restrição (5.2) temos que $G' \subseteq H$; logo H é subgrafo conexo e gerador de G. Como (x', Y') respeita a restrição (5.3), segue que

$$\text{dist}_H(u,v) \leq \text{dist}_{G'}(u,v) \leq \text{dist}_{G[F(e)]}(u,v) \leq \sum_{f \in E} Y'_{e,f}\ w_f \leq t \cdot w_e.$$

A conclusão de que H é um t-spanner de G segue do Lema 4.1. Como x' é o vetor de incidência de $E(H)$, segue que x' é um vértice de $P_{span}(G,t)$. □

Juntamos a seguir os resultados das duas proposições anteriores.

Proposição 5.3. *As seguintes afirmações resumem como $P_{span}(G,t)$ e $((P(G,t))_I$ estão relacionados:*

(a) *Se x' é um vértice de $P_{span}(G,t)$, então existe Y' (inteiro) tal que (x', Y') é um vértice de $((P(G,t))_I$.*

(b) *Se (x', Y') é um ponto inteiro que pertence a $P(G,t)$, então x' é um vértice de $P_{span}(G,t)$.*

Em vista do que foi provado anteriormente, o seguinte programa linear inteiro encontra uma solução ótima para o MWSP, com entrada (G, w, t).

$$\begin{aligned}
(P) \qquad & \min \sum_{e \in E} w_e x_e \\
& \text{s.a} \\
& (x, Y) \in P(G,t) \\
& x \in \{0,1\}^{|E|} \\
& Y \in \{0,1\}^{|E| \times |E|}
\end{aligned}$$

5.2.1 Separação das inequações de corte

Na formulação (PL), o número de inequações correspondentes à restrição (5.1) — conhecidas como *inequações de corte* — é exponencial no tamanho de V.

A separação das inequações de corte pode ser feita em tempo polinomial no tamanho de V. Dado um ponto (x^*, Y^*), para cada $e = uv \in E$, seja $G_e = (V, E')$, onde $E' = E$ e $w_f := Y^*_{e,f}, \forall f \in E(G_e)$. A capacidade de um (u,v)-corte mínimo em G_e deve ser maior ou igual a 1. Caso exista um $e = uv \in E$ tal que um (u,v)-corte mínimo $\delta(W)$ em G_e tenha capacidade menor do que 1, então

o par (e, W) é um certificado de que a restrição (5.1) é violada. Como o problema de encontrar um (u, v)-corte mínimo pode ser resolvido em tempo polinomial (no tamanho do grafo), a separação para as inequações de corte pode ser feita em tempo polinomial (veja [KV12]).

Capítulo 6

Um algoritmo de *branch-and-price* para o problema de t-spanner de peso mínimo

Neste capítulo abordamos o problema de t-spanner de peso mínimo (MWSP), focando na implementação que fizemos de um algoritmo de *branch-and-price* para o MWSP. Apresentamos na Seção 6.1 o algoritmo proposto por Sigurd e Zachariasen [SZ04], no qual baseamos a nossa implementação. Na seção seguinte apresentamos os detalhes da implementação, mencionando primeiramente os pré-processamentos realizados. Em seguida, abordamos as heurísticas implementadas para calcular limitantes primais e duais. Por fim, discutimos estratégias de ramificação e de busca do nó na árvore de B&B, finalizando com detalhes sobre o processamento dos nós desta árvore. No artigo de Sigurd e Zachariasen, os autores não comentam aspectos sobre adoção de pré-processamentos e heurísticas para o cálculo de limitantes.

6.1 Geração de colunas para resolver formulação relaxada

Em 2004, Sigurd e Zachariasen [SZ04] apresentaram um algoritmo exato para uma generalização do MWSP em que a restrição de spanner deve ser satisfeita por um subconjunto de pares de vértices. Nesta tese, consideramos a versão clássica (em vez da versão generalizada), na qual todos os pares de vértices devem satisfazer a restrição de spanner. Vamos apresentar nesta seção o algoritmo exato proposto por Sigurd e Zachariasen.

Como apresentado no Capítulo 5, o poliedro dos t-spanners de um grafo $G = (V, E)$, onde $t \geq 1$ é um número real, é definido como

$$P_{span}(G, t) := \text{conv}\{\mathcal{X}^F \in \mathbb{R}^{|E|} \mid G[F] \text{ é um } t\text{-spanner de } G\}.$$

A formulação proposta por Sigurd e Zachariasen [SZ04] para o MWSP(G, w, t) é a seguinte. Estamos interessados em conjuntos $F \subseteq E$ tais que $\mathcal{X}^F \in P_{span}(G, t)$. Conforme vimos no início do Capítulo 3, para $e = uv \in E$, denotamos por $\mathcal{P}_{u,v}$ a coleção de caminhos t-spanner de u a v em G. Seja $\mathcal{P} = \bigcup_{uv \in E} \mathcal{P}_{u,v}$.

A formulação linear faz uso de dois tipos de variáveis de decisão 0/1, como segue. (a) Variável $x \in \mathbb{R}^{|E|}$ com o seguinte significado: para cada aresta $e \in E$, $x_e = 1$ se e só se e faz parte da solução F. (b) Variável $y \in \mathbb{R}^{|\mathcal{P}|}_{\geq 0}$ com o seguinte significado: para cada aresta $e = uv \in E$ e para cada caminho $p \in \mathcal{P}_{u,v}$, $y_p = 1$ se e só se p conecta u a v na solução.

Considere a seguinte formulação linear 0/1 (proposta por Sigurd e Zachariasen). Nessa formulação, para $e \in E$, $p \in \mathcal{P}$, considere que $\delta^e_p = 1$ se a aresta e pertence ao caminho p, e $\delta^e_p = 0$, em caso contrário.

Primal $\min \sum_{e \in E} w_e x_e$

s.a

$$\sum_{p \in \mathcal{P}_{u,v}} \delta^e_p y_p \leq x_e \qquad \forall e \in E, \forall uv \in E \tag{6.1}$$

$$\sum_{p \in \mathcal{P}_{u,v}} y_p \geq 1 \qquad \forall uv \in E \tag{6.2}$$

$$x_e \in \{0,1\} \qquad \forall e \in E \tag{6.3}$$

$$y_p \in \{0,1\} \qquad \forall p \in \mathcal{P} \tag{6.4}$$

A inequação (6.2) diz que para cada par de vértices u, v que define uma aresta, deve existir pelo menos um caminho t-spanner entre u e v na solução. Para cada aresta e que faça parte de um caminho t-spanner selecionado, a inequação (6.1) força que e pertença à solução.

Sigurd e Zachariasen usam o método de geração de colunas para resolver a relaxação do programa Primal (que resulta trocando-se as restrições de integralidade (6.3) e (6.4) por restrições de não negatividade). Em decorrência do número exponencial de variáveis y_p, considera-se uma versão reduzida da relaxação do programa Primal, denotada por RMP (*Reduced Master Problem*).

Para cada $uv \in E$, considere $\mathcal{P}'_{u,v} \subseteq \mathcal{P}_{u,v}$ e $\mathcal{P}' = \bigcup_{uv \in E} \mathcal{P}'_{u,v}$. O RMP do programa Primal é definido da seguinte forma:

RMP $\min \sum_{e \in E} w_e x_e$

s.a

$$\sum_{p \in \mathcal{P}'_{u,v}} \delta^e_p y_p \leq x_e \qquad \forall e \in E, \forall uv \in E \tag{6.5}$$

$$\sum_{p \in \mathcal{P}'_{u,v}} y_p \geq 1 \qquad \forall uv \in E \tag{6.6}$$

$$x_e \geq 0 \qquad \forall e \in E \tag{6.7}$$

$$y_p \geq 0 \qquad \forall p \in \mathcal{P}' \tag{6.8}$$

Para resolver o programa linear RMP, basta que para cada $e = uv \in E$, $\mathcal{P}'_{u,v}$ contenha inicialmente apenas um caminho. Para cada iteração, novas colunas (que representam novos caminhos em $\mathcal{P}'$) vão sendo adicionadas ao programa linear caracterizando a abordagem conhecida como geração de colunas. Para cada $uv, e \in E$, sejam $\pi^{u,v}_e$ e $\sigma_{u,v}$ as variáveis duais associadas às inequações (6.1)

e (6.2) respectivamente. O dual da relaxação do programa Primal é descrito pelo seguinte PL:

$$\text{Dual} \quad \max \sum_{uv \in E} \sigma_{u,v}$$

s.a

$$\sum_{uv \in E} \pi_e^{u,v} \leq w_e \qquad \forall e \in E \tag{6.9}$$

$$\sum_{e \in E} \delta_p^e \pi_e^{u,v} - \sigma_{u,v} \geq 0 \qquad \forall uv \in E,\ \forall p \in \mathcal{P}_{u,v} \tag{6.10}$$

$$\pi_e^{u,v} \geq 0 \qquad \forall e \in E,\ \forall uv \in E \tag{6.11}$$

$$\sigma_{u,v} \geq 0 \qquad \forall uv \in E \tag{6.12}$$

A resolução do RMP serve para fornecer soluções (duais) para o programa Dual. Seja (x', y') uma solução ótima do RMP e seja (π', σ') a solução dual associada. Se (π', σ') for uma solução viável para o programa Dual, então (x', y') é uma solução ótima para Primal (a ser explicado a seguir). Considere a seguinte expressão:

$$c_p^{\pi,\sigma} = \sum_{e \in E} \delta_p^e \pi_e^{u,v} - \sigma_{u,v}, \ \ \forall uv \in E,\ \forall p \in \mathcal{P}_{u,v}. \tag{6.13}$$

O valor $c_p^{\pi,\sigma}$ é denominado *custo reduzido* (*reduced cost*) do caminho p com relação às variáveis π e σ. Caso a solução (π', σ') seja uma solução viável para o programa Dual, significa que para cada $uv \in E$ e para cada $p \in \mathcal{P}_{u,v}$, o custo reduzido associado é maior ou igual a zero. Neste caso, temos que a solução (x', y') é ótima (veja o Teorema 3.1 em Bertsimas e Tsitsiklis [BT97]). Caso contrário, para algum par $uv \in E$ e $p \in \mathcal{P}$, a inequação (6.10) é violada, ou seja, $c_p^{\pi',\sigma'} < 0$. A inequação (6.9) não é violada pois, para cada $e \in E$, a inequação (6.9) está relacionada à coluna da variável x_e no programa Primal, e todas estas colunas já estão presentes no RMP.

Para um par $uv \in E$ e $p \in \mathcal{P}$ tal que a inequação (6.10) é violada, devemos adicionar o caminho p (entre os vértices u e v) ao RMP. Em cada iteração da geração de colunas, para cada $uv \in E$, queremos encontrar um caminho t-spanner entre u e v com o menor valor de custo reduzido. Para isso, devemos resolver

$$\min_{p \in \mathcal{P}_{u,v}} \sum_{e \in E} \delta_p^e \pi_e^{u,v} - \sigma_{u,v}. \tag{6.14}$$

Se o valor da solução do problema (6.14) for menor do que zero, então encontramos o caminho a ser inserido no RMP. Caso contrário, a geração de colunas é finalizada. O problema representado pela expressão (6.14) é conhecido como o problema do *pricing*. Para cada $uv \in E$, $\sigma_{u,v}$ é constante e o *pricing* consiste em encontrar um caminho t-spanner entre u e v de menor peso, sendo que o peso das arestas é dado pelos valores da variável dual π. Este problema é conhecido como *problema do caminho mínimo restrito* (*Constrained Shortest Path Problem* - CSPP).

6.1.1 Problema do caminho mínimo restrito

O problema do caminho mínimo restrito (CSPP) é definido da seguinte forma: dado um grafo $G = (V, E)$, uma função peso $w : E \to \mathbb{Z}_{\geq 0}$, uma função custo $c : E \to \mathbb{Z}_{\geq 0}$, vértices $s, d \in V$, e uma constante B denominada *limite de recurso*, o objetivo é encontrar um caminho de custo mínimo entre s e d com relação à função custo c, cuja soma dos pesos das arestas seja menor ou igual a B. A versão de decisão deste problema, onde se considera não apenas um limitante para o peso mas também um limitante para o custo, é NP-Completo (veja problema ND30 em [GJ90]).

Para resolver o CSPP, Sigurd e Zachariasen [SZ04] adotaram o algoritmo baseado em rótulos proposto por Desrochers e Soumis [DS88]. Neste algoritmo, caminhos a partir do vértice s vão sendo construídos e incrementados até atingir o vértice d. Um caminho p_i é representado pelo rótulo (p_i, v_i, w_i, c_i), onde n_i representa o vértice final do caminho, w_i e c_i representam o peso e o custo total respectivamente de p_i. Dizemos que um caminho p_i *domina* um caminho p_j se $v_i = v_j$ e $w_i \leq w_j$ e $c_i \leq c_j$.

O algoritmo funciona em rodadas, e em cada rodada é escolhido o rótulo p_i com menor custo. O caminho p_i é estendido para cada vizinho de v_i, dando origem a novos rótulos. Cada rótulo dominado, novo ou antigo, deve ser eliminado. Seja $\text{dist}_G(u, v, w)$ a distância entre u e v em G medida pela função w. De maneira análoga, defina $\text{dist}_G(u, v, c)$. Para cada novo caminho criado p_j, se $w_j + \text{dist}_G(v_j, d, w) > B$, descartamos p_j. Seja M uma constante que é um limitante para o custo máximo dos caminhos. Além da verificação anterior, se $c_j + \text{dist}_G(v_j, d, c) > M$, o caminho p_j é descartado. Para calcular as distâncias entre os vértices e o vértice destino d, árvores de caminhos mínimos são construídas. Vamos chamar de $\mathcal{T}^w$ a árvore de caminhos de peso mínimo de G a partir de d e de $\mathcal{T}^c$ a árvore de caminhos de custo mínimo de G a partir de d.

Para cada iteração da geração de colunas, o Algoritmo 6.1.1 é executado para cada par $e = uv \in E$. O peso (resp. custo) das arestas é dado pela função w associada a G (resp. $\pi^{u,v}$). Os limitantes M e B possuem valores $\sigma^{u,v}$ e $t \cdot w_e$ respectivamente. Os vértices de origem e destino são u e v respectivamente.

O Algoritmo 6.1.1 é utilizado para resolver o CSPP.

6.2 Detalhes da implementação

Nesta seção abordamos alguns detalhes relativos à implementação do algoritmo de *branch-and-price* para o MWSP baseado na formulação apresentada na seção anterior. Mencionamos os pré-processamentos realizados, e as heurísticas utilizadas para cálculo dos limitantes primais e duais.

6.2.1 Pré-processamento

Os pré-processamentos que realizamos visam fixar, quando possível, o valor de algumas variáveis associadas às arestas (tanto as que necessariamente irão fazer parte de qualquer solução como as que necessariamente não farão parte de nenhuma solução).

- *Aresta t-essencial.* Lembramos que uma aresta e de G é t-essencial se $G - e$ não é um t-spanner de G. Na fase de pré-processamento, para cada aresta t-essencial e do grafo de entrada, impomos que $x_e = 1$.

Algoritmo 6.1.1: CSPP

Entrada: $G = (V, E)$, $w : E \to \mathbb{R}_{\geq 0}$, $c : E \to \mathbb{R}_{\geq 0}$, número real B, número real M, $s \in V$, $d \in V$;
Saída: caminho p_f entre s e d de menor custo satisfazendo a restrição de peso máximo B e de custo máximo M;

```
p_f ← null;
Calcula as árvores de caminhos de peso e custo mínimos T^w e T^c;
H ← {({}, s, 0, 0)};
while H ≠ ∅ do
    Escolha o rótulo (p_α, v_α, w_α, c_α) mais barato (em termos de custo) no conjunto H;
    if v_α = d then
        p_f ← p_α;
        return p_f.
    end
    foreach v_i ∈ N_{v_α}(G) do
        Seja e_i ← v_α v_i. Crie o novo rótulo (p_α ∪ {e_i}, v_i, w_α + w_{e_i}, c_α + c_{e_i});
        Descarte todos os novos rótulos tal que w_α + w_{e_i} + T^w(v_i) > B ou
            c_α + c_{e_i} + T^c(v_i) > M. Para os demais, armazene em H;
        Descarte (de H) rótulos dominados;
    end
    return p_f.
end
```

- *Aresta t-inútil.* Dizemos que uma aresta $e = uv \in E$ é uma *aresta t-inútil* de G se $w_e > t \cdot \text{dist}_G(u, v)$. Para cada aresta t-inútil e do grafo de entrada, fixamos em zero o valor da variável x_e. Observe que, no caso unitário, nenhuma aresta é t-inútil.

6.2.2 Heurísticas para cálculo de limitantes primais

Apresentamos nesta subseção duas heurísticas para calcular limitantes primais, que utilizamos na rotina de B&B. Uma delas, a heurística baseada numa partição em *clusters* só é utilizada no caso unitário. Caso uma destas heurísticas encontre um limitante menor do que o valor da solução incumbente, atualizamos a solução incumbente.

spanner baseado numa partição em *clusters*

O Algoritmo 6.2.2 descrito a seguir constrói um spanner de G, onde G tem peso unitário, baseado na partição dos vértices de G em *clusters*. Este algoritmo foi desenvolvido por Peleg [Pel00].

Dado um inteiro $k \geq 1$, o Algoritmo 6.2.2 constrói um $(2k-1)$-spanner com no máximo $O(n^{1+1/k})$ arestas. Para o caso unitário e com este fator de dilatação, este algoritmo constrói um spanner com o menor número de arestas (veja [BKMP10]). Ele inicialmente constrói uma partição dos vértices do grafo (usando o Algoritmo 6.2.1), tal que cada classe da partição é um subgrafo conexo, chamado *cluster*.

Cada *cluster* é construído a partir de um vértice e adicionando-se camadas, finalizando-se a adição de camadas quando determinada restrição de esparsidade é violada. Após construir a partição

$\mathcal{T}$ por meio do Algoritmo 6.2.1, para cada *cluster* de $\mathcal{T}$ é construída uma árvore de caminhos mínimos. Além disso, para cada *cluster* S de $\mathcal{T}$, e para cada vértice u vizinho de S e que não esteja em S, é garantida a existência de uma aresta unindo u a S na solução final. Tal solução será formada por estas arestas externas ao *cluster* juntamente com as árvores de caminhos mínimos.

Para $S \subseteq V$, seja $\Gamma^{ver}(S) = S \cup \{v \in V \setminus S \mid \exists u \in S \text{ e } uv \in E\}$.

Algoritmo 6.2.1: Partição básica

Entrada: $G = (V, E)$, onde $|V| = n$, e um inteiro positivo k;
Saída: partição $\mathcal{T}$ de V;
$\mathcal{T} \leftarrow \emptyset$;
Seja $G' = (V', E')$ um grafo isomorfo a G;
while $V(G') \neq \emptyset$ **do**
  Selecione um vértice arbitrário $v' \in V(G')$;
  $S' \leftarrow \{v'\}$;
  while $|\Gamma^{ver}(S')| < n^{1/k}|S'|$ **do**
    $S' \leftarrow \Gamma^{ver}(S')$;
  end
  Seja $S = \{v \in V \mid \exists\, v' \in S' \text{ e } v' \text{ é o vértice correspondente a } v \text{ no isomorfismo}\}$;
  $\mathcal{T} \leftarrow \mathcal{T} \cup S$;
  $V(G') \leftarrow V(G') \setminus S'$;
end

Algoritmo 6.2.2: spanner baseado em partição em *clusters*

Entrada: $G = (V, E)$, inteiro positivo k;
Saída: Um grafo $G' = (V, E')$ que é um $(2k-1)$-spanner de G com no máximo $O(n^{1+1/k})$ arestas;

Construir uma partição $\mathcal{T}$ de V usando o Algoritmo 6.2.1 (Partição básica);
foreach $S_i \in \mathcal{T}$ **do**
  $T_i \leftarrow$ uma árvore de caminhos mínimos de $G[S_i]$ enraizada em algum vértice de S_i;

$E(G') \leftarrow \bigcup_{S_i \in \mathcal{T}} E(T_i)$;
$\breve{E} \leftarrow \emptyset$;
foreach $S_i \in \mathcal{T}$ **do**
  foreach $v \in \Gamma^{ver}(S_i) \setminus S_i$ **do**
    Seja $u \in S_i$ t.q. $uv \in E$;
    $\breve{E} \leftarrow \breve{E} \cup uv$;

$E(G') \leftarrow E(G') \cup \breve{E}$;
return G'.

O seguinte resultado prova que o Algoritmo 6.2.2 de fato produz a saída que mencionamos (veja o exercício 3 do capítulo 16 em Peleg [Pel00]). Parte da prova foi apresentada por Peleg [Pel00], a parte omitida segue abaixo.

Teorema 6.1 (Peleg [Pel00]). *O Algoritmo 6.2.2 aplicado a um grafo G de ordem n, e um inteiro positivo k, constrói um $(2k-1)$-spanner de G com no máximo $O(n^{1+1/k})$ arestas.*

Demonstração. O limite superior relativo ao número de arestas decorre do Teorema 11.5.1 (b) em [Pel00] e do fato de que a quantidade total de arestas das árvores construídas é menor do que $n^{1+1/k}$ (mais especificamente, menor ou igual a $n-1$).

Vamos verificar o fator de dilatação. Seja $G=(V,E)$, e uv uma aresta de G. Se u e v pertencem a um mesmo *cluster*, então $\text{dist}_{G'}(u,v) \leq 2(k-1)$, visto que o raio do *cluster* é menor ou igual a $k-1$ (Teorema 11.5.1 (a) em [Pel00]). Agora considere *clusters* $S_i, S_j \in \mathcal{T}$ tais que $u \in S_i$ e $v \in S_j$. Se $uv \in \breve{E}$, segue a afirmação. Caso contrário, seja $u' \in S_i$ tal que $u'v \in E$ (a existência de u' é garantida pela linha 8 do Algoritmo 6.2.2) e seja $v_i \in S_i$ o centro de S_i. Então

$$\text{dist}_{G'}(u,v) \leq \text{dist}_{G'}(u,v_i) + \text{dist}_{G'}(v_i,u') + \text{dist}_{G'}(u',v) \leq (k-1)+(k-1)+1 = 2k-1.$$

□

Na desigualde anterior, podemos afirmar que $\text{dist}_{G'}(u',v) = 1$ em decorrência das linhas 7-9 no Algoritmo 6.2.2.

Observamos que, ao executar o Algoritmo 6.2.2 ao longo da construção da árvore de B&B, nossa implementação prioriza as arestas correspondentes às variáveis escolhidas para fazer a ramificação e que foram fixadas com valor um.

Algoritmo guloso

No trabalho de Sigurd e Zachariasen [SZ04] é mencionado que os autores estavam interessados em avaliar o desempenho da heurística gulosa proposta por Althöfer et al. [ADD+93], o que os motivou a desenvolver um algoritmo exato para o MWSP. Descrevemos a seguir a heurística gulosa, alvo da comparação, referida como Algoritmo 6.2.3.

Algoritmo 6.2.3: Algoritmo Guloso de Althöfer et al.

Entrada: $G=(V,E)$ com pesos não negativos nas arestas, número real $t \geq 1$;
Saída: t-spanner G' de G com peso pequeno;
Ordene as arestas de E em ordem não decrescente de seus pesos;
$G' \leftarrow (V',E')$, onde $V(G') \leftarrow V, E(G') \leftarrow \emptyset$;
foreach $e = uv \in E$ **do**
  if $t \cdot w_e < \text{dist}_{G'}(u,v)$ **then** $E(G') \leftarrow E(G') \cup \{e\}$;
end
return G'.

A heurística para o cálculo de um limitante primal é baseada no Algoritmo Guloso de Althöfer et al. [ADD+93], acima descrito. Como o cálculo de limitantes primais é realizado periodicamente ao longo da execução do algoritmo de B&B, variáveis diferentes fixadas em zero ou um implicam (possivelmente) em valores diferentes de limitantes primais. Em cada nó da árvore, para as arestas cujas variáveis foram fixadas em um, a heurística as força a pertencer à solução viável, enquanto que para as arestas cujas variáveis foram fixadas em zero, a heurística as exclui da solução. Como

Algoritmo 6.2.4: MST baseada nas arestas fixadas pelo B&B

Entrada: $G = (V, E)$, $E_0 \subset E$, $E_1 \subset E$, $w : E \to \mathbb{R}_{\geq 0}$;
Saída: limitante dual para a tripla (G, E_0, E_1);
/* E_0: conjunto das arestas fixadas em zero;
/* E_1: conjunto das arestas fixadas em um;
$LD \leftarrow \sum_{e \in E_1} w_e$;
Seja $G_1 = (V, E_1)$, e $\mathcal{C}$ o conjunto dos componentes de G_1;
Seja $G_1^c = (V_1^c, E_1^c)$ o grafo assim definido:
$V_1^c = \{C_i |\ C_i \in \mathcal{C}\}$;
$E_1^c = \{C_i C_j \mid C_i, C_j \in \mathcal{C}, i \neq j,$ e $\exists x, y \in V(G)$ t.q. $x \in V(C_i), y \in V(C_j), xy \in E, xy \notin E_0\}$;
$T \leftarrow \text{MST}(G_1^c)$;
$LD \leftarrow LD + \sum_{e \in E(T)} w_e$;
return LD.

a heurística é executada após a execução do *pricing*, a heurística escolhe as arestas em ordem não decrescente, de acordo com os valores das variáveis associadas às arestas.

6.2.3 Heurística para cálculo de limitante dual

Para calcular um limitante dual, implementamos uma heurística que gera uma árvore geradora mínima (*Minimum Spanning Tree* - MST) conectando os componentes induzidos pelas arestas cujas variáveis foram fixadas em um. Nesta árvore, não consideramos arestas cujas variáveis foram fixadas em zero. O Algoritmo 6.2.4 corresponde à heurística mencionada. Note que sempre é possível criar uma árvore que conecta os componentes. Como veremos na Seção 6.2.4, na criação dos nós da árvore de B&B, o nó da árvore que representa a variável fixada em zero só é criado se não inviabilizar a existência de uma solução t-spanner.

Caso esta heurística devolva um valor melhor do que o valor gerado pelo *pricing* (ou seja, o valor gerado pela heurística é maior do que o valor do *pricing*), então o limitante inferior do nó da árvore sendo analisado é atualizado com o valor da heurística.

6.2.4 Estratégia de ramificação na árvore de B&B

Adotamos o critério de ramificação em valores inteiros, seguindo a proposta de Morrison et al. [MJSS16]. Neste caso, a ramificação é dividida em duas fases: selecionar uma variável fracionária onde será feita a ramificação; posteriormente, forçar esta variável fracionária a ter valores inteiros (no nosso caso, zero ou um) e, por consequência, criar subproblemas a partir destas restrições de integralidade. No nosso caso, como as variáveis do programa Primal são binárias e como é escolhida apenas uma variável fracionária, serão criados (no máximo) dois subproblemas filhos.

Criação dos subproblemas filhos

No processo de ramificação, ao fixar uma variável fracionária em um, o subproblema resultante é criado e armazenado na lista dos subproblemas ainda não analisados (veja Seção 2.3.2). Para o subproblema resultante da fixação da variável no valor zero, é feito um teste para saber se o

grafo resultante admite um caminho t-spanner entre os extremos da aresta cuja variável foi fixada em zero. Se o grafo admitir tal caminho, então o subproblema é criado e armazenado na lista dos subproblemas ainda não analisados. Caso contrário, o subproblema não é criado.

Escolha da variável fracionária

Implementamos duas formas de escolher uma variável fracionária para realizar a ramificação:

(a) Escolher uma variável fracionária cujo valor está mais próximo de $\frac{1}{2}$. Como as variáveis são binárias, este critério prioriza escolher uma variável para a qual o "*solver* estaria mais indeciso" sobre o valor a ser atribuído;

(b) Escolher uma variável com um maior valor fracionário. Este critério prioriza as variáveis para as quais o "*solver* já está quase certo" sobre o valor a ser atribuído.

6.2.5 Estratégia de escolha do nó a ser examinado na árvore de B&B

A seleção do próximo nó a ser examinado é feita utilizando a estratégia de *procurar o melhor primeiro* (*Best-First Search* - BeFS) [MJSS16, Cla99]. Nesta estratégia, para cada um dos subproblemas (nós) ativos, devemos escolher aquele que minimiza uma determinada função. No nosso caso, esta função corresponde ao limitante inferior gerado pelo *pricing*. Ou seja, ao longo da execução do algoritmo, vamos tentando diminuir a diferença entre limitantes primal e dual.

6.2.6 Processamento do nó da árvore de B&B

Seja $v(E_o, E_1)$ o nó que está sendo analisado na árvore de B&B, onde E_0 (resp. E_1) corresponde ao conjunto de arestas cujas variáveis em x foram fixadas em zero (resp. um). Seja
$\mathcal{S}_{E_0,E_1} := \{\mathcal{X}^F \in \mathbb{R}^{|E|},\ E_0 \cap F = \emptyset,\ E_1 \subseteq F \mid G[F]$ é um grafo t-spanner$\}$.

Inclusão de inequações válidas

Para $e, f \in E,\ e \neq f$, tais que x_e e x_f não foram fixados, seja
$\mathcal{S}_{E_0,E_1}(e, f) := \{\mathcal{X}^F \in \mathcal{S}_{E_0,E_1} \mid G[F - \{e, f\}]$ é um grafo t-spanner$\}$.
Neste caso, a seguinte inequação é válida (e pode ser acrescentada):

$$x_e + x_f \geq 1 \qquad \forall e, f \in E,\ e \neq f,\ e, f \notin E_0 \cup E_1,\ \mathcal{S}_{E_0,E_1}(e, f) = \emptyset \qquad . \quad (6.15)$$

A geração de colunas não será afetada pela inclusão destas inequações, pois as inequações do programa Dual que poderiam ser violadas (inequação (6.10)) não sofrerão alterações após a inclusão dessas inequações ao RMP, como pode ser visto a seguir. Para cada $e \in E$, seja $F_{ponte}(e)$ o conjunto das arestas relacionadas com e, definida como o conjunto das arestas f tal que o par e, f respeita a restrição (6.15). Para cada $e \in E$ e $f \in F_{ponte}(e)$, seja $\beta_{e,f}$ a variável dual associada à restrição (6.15). O dual da relaxação do programa Primal juntamente com as inequações (6.15) é descrito pelo seguinte PL:

Dual 2 $\max \sum_{uv \in E} \sigma_{u,v} + \sum_{e \in E} \sum_{f \in F_{ponte}(e)} \beta_{e,f}$

s.a

$$\sum_{uv \in E} \pi_e^{u,v} + \sum_{f \in F_{ponte}(e)} \beta_{e,f} \leq w_e \qquad \forall e \in E \tag{6.16}$$

$$\sum_{e \in E} \delta_p^e \pi_e^{u,v} - \sigma_{u,v} \geq 0 \qquad \forall uv \in E,\ \forall p \in \mathcal{P}_{u,v} \tag{6.17}$$

$$\pi_e^{u,v} \geq 0 \qquad \forall e \in E,\ \forall uv \in E \tag{6.18}$$

$$\sigma_{u,v} \geq 0 \qquad \forall uv \in E \tag{6.19}$$

$$\beta_{e,f} \geq 0 \qquad \forall e \in E,\ \forall f \in F_{ponte}(e) \tag{6.20}$$

Tratamento das variáveis fixadas em zero ou um

Para cada nó da árvore de B&B, temos um conjunto de arestas que foram fixadas em zero ou um. Seja $v(E_0, E_1)$ como descrito anteriormente. Adicionamos as seguintes equações ao RMP para tratar as arestas fixadas:

$$\sum_{e \in E_1} x_e = |E_1| \tag{6.21}$$

$$\sum_{e \in E_0} x_e = 0 \tag{6.22}$$

Para $e \in E$, vamos considerar o caso em que queremos fixar a variável x_e em um. Quando adicionamos a restrição $x_e = 1$, o *pricing* não é comprometido visto que o grafo de entrada para o *pricing* não é alterado. Este grafo não necessita ser alterado pois não é necessário usar a aresta e em nenhum caminho gerado para ter uma solução válida, visto que a restrição (6.5) para ser válida, não exige que nenhum caminho contenha tal aresta. Uma alternativa (que não foi implementada) seria não adicionar a restrição $x_e = 1$. Neste caso, os valores duais gerados (dados pela variável π) seriam diferentes e, consequentemente, as arestas teriam custos diferentes no grafo de entrada do CSPP, mas a estrutura do grafo de entrada seria a mesma.

Para $e \in E$, quando queremos fixar a variável x_e em zero, adicionamos a restrição $x_e = 0$ e removemos a aresta e do grafo de entrada do CSPP. Uma alternativa (que não foi implementada) seria simplesmente remover a aresta e do grafo de entrada do CSPP. Neste caso, os valores duais gerados seriam diferentes, mas a aresta e continuaria sem poder ser escolhida pelo *pricing* para gerar um novo caminho. Em outras palavras, em ambos os casos a aresta e nunca poderá ser escolhida como parte de um caminho gerado pelo *pricing*.

6.3 Uso do algoritmo para árvore t-spanner de peso mínimo

Fazendo-se uma pequena alteração, a implementação descrito neste capítulo pode ser utilizada para resolver o problema da árvore t-spanner de peso mínimo (MWTSP). Com isso, podemos comparar essa implementação com as formulações lineares inteiras propostas por nós para o MWTSP, apresentadas no capítulo anterior.

Para evitar alterar o *pricing*, não adicionamos ao programa Primal a restrição que limita o número de arestas da solução. Em vez disso, adicionamos coeficientes com valores altos para cada uma das variáveis (associadas às arestas) que compõem a função objetivo do programa Primal. Os coeficientes escolhidos são suficientemente altos para que qualquer solução viável com um número menor de arestas seja priorizada com relação às soluções com um número maior de arestas. Ao fim, se a solução ótima encontrada tiver um número de arestas maior ou igual ao número de vértices, então sabemos que o grafo de entrada não admite uma árvore t-spanner.

Toda a implementação (incluindo as heurísticas) para o MWSP continua sendo válida para o MWTSP. Não encontramos na literatura heurísticas para cálculo de limitantes primais para o MWTSP.

A implementação de Sigurd e Zachariasen [SZ04] não está disponível. Resolvemos implementar o algoritmo de Sigurd e Zachariasen para realizar um conjunto maior de experimentos computacionais (quando comparado à quantidade de experimentos realizados pelos autores) e para comparar a implementação com as formulações lineares inteiras que propusemos no Capítulo 4 para o MWTSP. Estas comparações serão apresentadas no Capítulo 7. Como comentado no início do capítulo, Sigurd e Zachariasen não mencionam pré-processamentos e heurísticas para cálculo de limitantes primais e duais, sendo estas as principais diferenças de nossa implementação para a dos autores.

Capítulo 7

Experimentos computacionais

Neste capítulo apresentamos os resultados de experimentos computacionais realizados para comparar as duas formulações lineares inteiras que propusemos no Capítulo 4 para o *problema da árvore t-spanner de peso mínimo* (MWTSP). Também apresentamos os resultados computacionais relativos à implementação do algoritmo de *branch-and-price* para o *problema de t-spanner de peso mínimo* (MWSP).

Conforme discutimos na Seção 6.3 do capítulo anterior, a implementação do algoritmo de *branch-and-price* para o MWSP pode ser usada para o MWTSP. Realizamos testes computacionais para comparar esta implementação com o desempenho das formulações inteiras propostas para o MWTSP.

7.1 Ambiente computacional e programas

Para realizar os experimentos computacionais, utilizamos o pacote de *software* de otimização CPLEX [IBM18], versão 12.5, configurado para utilizar uma única *thread* do processador. Para a implementação do algoritmo de *branch-and-price*, desabilitamos o pré-processamento do CPLEX em decorrência da geração de colunas, como sugere o manual do resolvedor. Os algoritmos foram implementados utilizando a linguagem de programação C++. O código foi executado num computador com 65 GB de memória RAM e processador de 1.6 GHz. Para implementar alguns algoritmos tradicionais para problemas de grafos, utilizamos a biblioteca *Lemon* [oCOE16]. Para gerar as instâncias de entrada utilizamos o *software* Scilab [Ent18].

7.2 Configuração dos parâmetros

Utilizamos como base dos nossos experimentos o trabalho de Sigurd e Zachariasen [SZ04], para escolher certos parâmetros e valores. Adicionalmente, introduzimos novos parâmetros e valores aos nossos experimentos.

Nos experimentos realizados, variamos seis parâmetros:

- *Ordem do grafo*: número de vértices. Variamos entre 16 e 100 vértices.

- *Densidade*: percentagem de arestas existentes no grafo de entrada, quando comparado ao grafo completo de mesma ordem. Consideramos densidades de 20% a 100%;
- *Tipo do grafo*: tipo do grafo com relação ao peso de suas arestas. Consideramos grafos com pesos unitários, com distância euclidiana, e pesos aleatórios. No caso de distância euclidiana, consideramos que os vértices do grafo estão espalhados no $\mathbb{R}^2_{\geq 0}$, dentro de um quadrado de lado 100. No caso dos grafos com pesos aleatórios, temos dois casos:
  - valores mais espaçados: pesos pertencentes ao conjunto $\{1, 2, 4, 8, 16\}$, cada qual com a mesma probabilidade de ser escolhido;
  - valores menos espaçados: pesos pertencentes ao conjunto $\{1, 2, 3\}$, cada qual com a mesma probabilidade de ser escolhido;
- *Fator de dilatação*: consideramos os valores 1,1; 2; 3; e 4;
- *Grau médio dos vértices*: utilizamos os valores 4 e 8;
- *Tempo limite*: para cada instância consideramos os tempos limite de execução de 1800 e 3600 segundos (de CPU).

A construção dos nossos grafos é feita de tal forma que as arestas são adicionadas de maneira aleatória, onde a probabilidade de adioná-las é a mesma, respeitando as restrições que são impostas para cada instância. Por exemplo, para um grafo de densidade de $x\%$, para cada par de vértices u, v, a probabilidade de a aresta uv ser inserida no grafo é de $x\%$.

Nos experimentos feitos, para cada combinação de parâmetros em que o limite de tempo é 1800 segundos, criamos 20 instâncias (reservamos 1800s para cada instância). Em decorrência da quantidade de experimentos realizados, para cada combinação em que o limite de tempo é 3600 segundos, criamos 10 instâncias (novamente, reservamos 3600s para cada instância). Os valores indicados de tempo referem-se à média considerando-se as instâncias resolvidas (a ser explicado a seguir) dentre o grupo de 20 (resp. 10) instâncias. O tempo médio foi arrendondado para o inteiro mais próximo. Com relação à implementação do algoritmo de *branch-and-price*, alguns testes mostraram que vale a pena adotar o critério (a) (veja Seção 6.2.4) como forma de escolher a variável fracionária para fazer a ramificação. Os dados exibidos neste capítulo foram obtidos com esse critério.

Consideramos que uma instância é resolvida quando, dentro do limite de tempo estabelecido, a implementação encontra uma solução ótima ou verifica que não existe uma solução viável (para o caso de árvores t-spanner).

A heurística (veja Seção 4.2) apresentada para as formulações lineares inteiras para o MWTSP (para o caso unitário) não foi executada nestes experimentos para que pudéssemos medir e comparar o desempenho das formulações lineares inteiras.

7.3 Resultados para árvore t-spanner de peso mínimo

Nesta seção apresentamos os experimentos para o problema da árvore t-spanner, comparando os desempenhos das formulações lineares inteiras e da implementação do algoritmo de *branch-and-price* que descrevemos para este problema. Ao longo deste capítulo, a formulação apresentada na

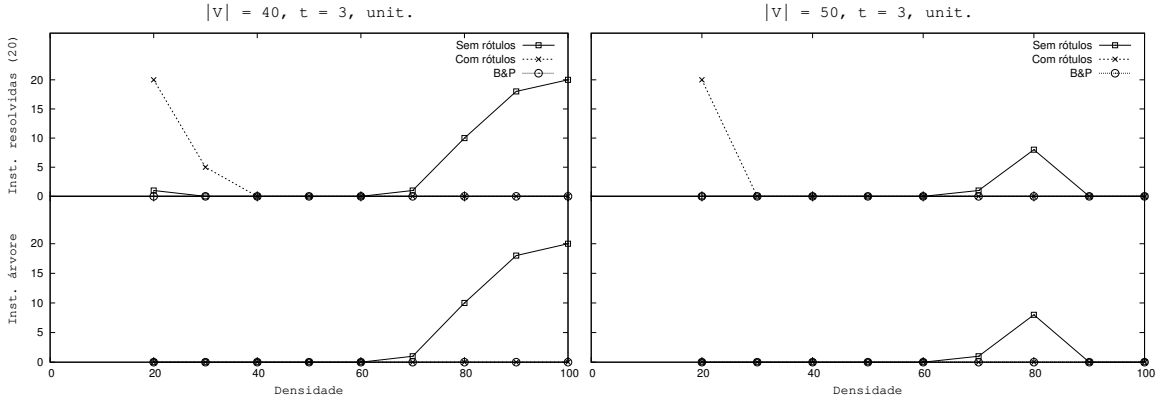

Figura 7.1: *Quantidade de instâncias resolvidas para $t = 3$, peso unitário e ordem alta*

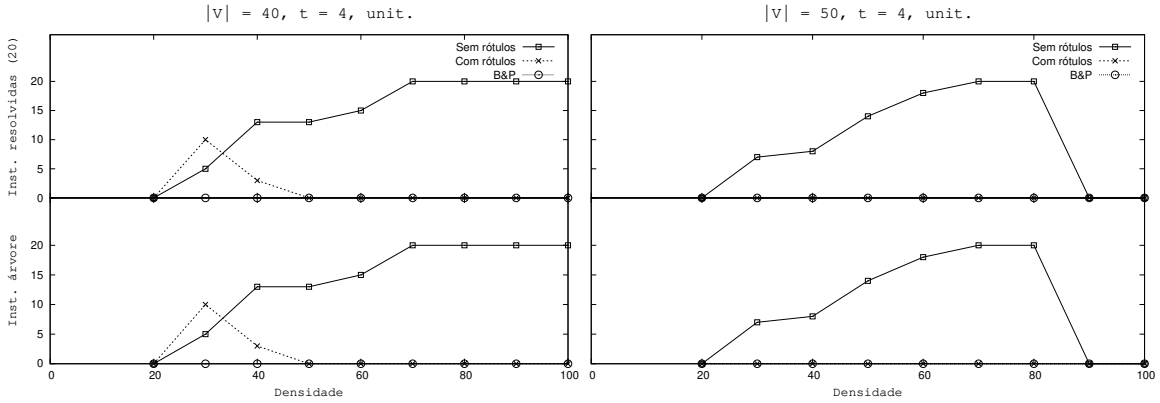

Figura 7.2: *Quantidade de instâncias resolvidas para $t = 4$, peso unitário e ordem alta*

Seção 4.3 será denominada *sem rótulos*, abreviada por *SR*; e a formulação apresentada na Seção 4.4 será denominada *com rótulos*, abreviada por *CR*. Por fim, a implementação do algoritmo de *branch-and-price* será chamada de *B&P*. Ao fim desta seção, apresentamos um quadro resumo indicando em quais situações cada algoritmo exato é mais indicado.

Ao longo deste capítulo, os gráficos que mostram somente informações relativas à quantidade de instâncias resolvidas têm o seguinte formato: a parte superior mostra a quantidade de instâncias resolvidas, e a parte inferior mostra a quantidade de soluções ótimas, que são árvores. A quantidade de instâncias totais, para as quais foram realizados os experimentos, está dentro dos parênteses ao lado do eixo das ordenadas. Para todo gráfico mostrado neste capítulo, na parte superior do gráfico são indicados os parâmetros que foram fixados assim como seus respectivos valores. Os grafos com pesos aleatórios e com valores mais espaçados serão abreviados por *aleat.* Já os grafos com pesos aleatórios mas com valores menos espaçados serão abreviados por *aleat. menor*. Abreviaremos os grafos com pesos unitários por *unit.*

Iniciamos os experimentos com grafos de ordem até 50. As Figuras 7.1 e 7.2 mostram a quantidade de instâncias resolvidas com pesos unitários e ordens 40 e 50. Pela Figura 7.1, podemos observar que, para os casos com pesos unitários e com densidades maiores, a formulação SR resolveu mais instâncias do que a formulação CR. Este comportamento fica mais claro quando o fator de dilatação é maior, como ilustrado na Figura 7.2. Para grafos pequenos (em termos de densidade e

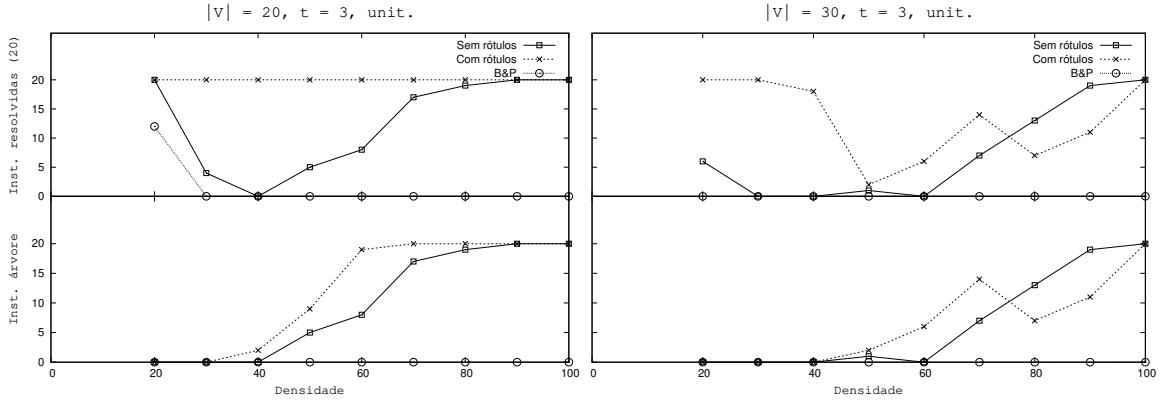

Figura 7.3: *Quantidade de instâncias resolvidas para $t = 3$, peso unitário e ordem baixa*

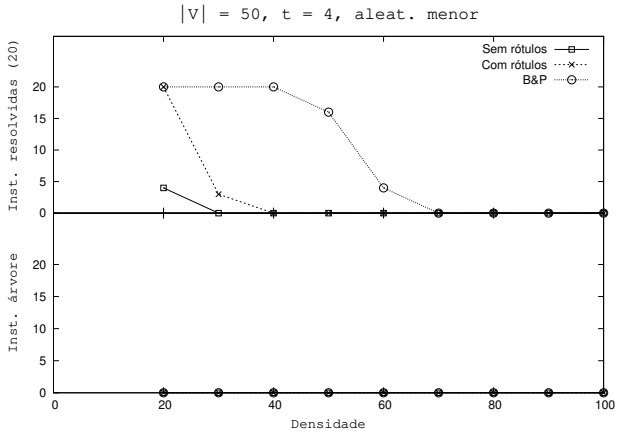

Figura 7.4: *Quantidade de instâncias resolvidas para $t = 4$, peso aleatório menos espaçado e $|V| = 50$*

ordem) com pesos unitários e com fator de dilatação baixo (ou seja, menos opções a serem analisadas pelo resolvedor), a formulação CR teve um desempenho melhor do que a formulação SR, como vemos na Figura 7.3. Este comportamento também é visto para os valores de densidade baixa na Figura 7.1.

Para instâncias com pesos aleatórios menos espaçados, a formulação CR mostrou-se melhor do que a formulação SR em todas as situações, considerando grafos com ordem até 50. O fato interessante é que à medida que a ordem do grafo ou o fator de dilatação aumenta, ambas as formulações vão resolvendo menos instâncias até que, para ordem 50 e fator de dilatação grande (quatro), B&P tem desempenho melhor (veja Figura 7.4). Já para instâncias com pesos mais espaçados, o comportamento das formulações SR e CR é semelhante, com a diferença de que a degradação acontece de maneira um pouco mais acentuada (veja Figura 7.5).

Notamos que, no caso de instâncias que admitem uma árvore t-spanner, com pesos unitários, quando o fator de dilatação é alto (quatro), as curvas dos desempenhos de uma mesma formulação nos gráficos superior e inferior são as mesmas (veja Figura 7.2). Em outras palavras, para cada instância resolvida, esta admite uma árvore t-spanner. Mesmo para o fator de dilatação três, quando as densidades são mais altas, as curvas são as mesmas (veja o lado direito das Figuras 7.1 e 7.3).

Para instâncias com pesos aleatórios menos espaçados, o único caso em que foram encontradas

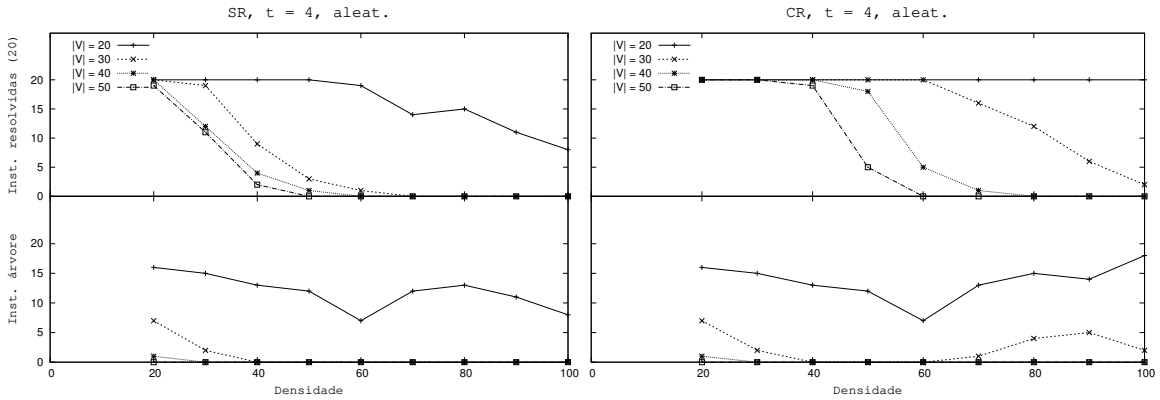

Figura 7.5: *Quantidade de instâncias resolvidas para $t = 4$ e peso aleatório*

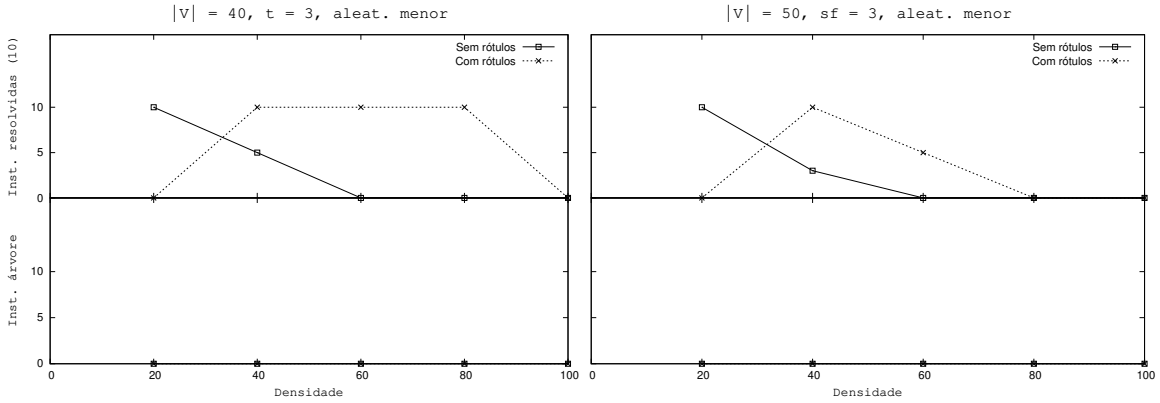

Figura 7.6: *Quantidade de instâncias resolvidas para $t = 3$, peso aleatório menos espaçado, ordem alta e com tempo limite de 3600 segundos*

árvores t-spanner como solução são aquelas com ordem 20 e com fator de dilatação quatro. No caso de pesos aleatórios mais espaçados, o comportamento é semelhante.

Vamos analisar agora o desempenho das formulações SR e CR quando o tempo limite de simulação é maior. Novamente consideramos a ordem variando entre 20 e 50 e a densidade entre 20% e 100%. Para instâncias com pesos aleatórios menos espaçados, ambas as formulações SR e CR se comportam de maneira semelhante à situação onde o tempo de simulação foi menor, descrita anteriormente. Além do desempenho de CR ter sido melhor, para o fator de dilatação três e para grafos de ordens 20 e 30, todas as instâncias foram resolvidas utilizando esta formulação, exceto para densidade de 20%. Para os outros valores de ordem, a Figura 7.6 ilustra o comportamento das duas formulações. Já para o fator quatro, quando as ordens são altas, quase nenhuma instância foi resolvida. Para ordens menores, veja a Figura 7.7. Para fatores de dilatação três e quatro, e para os valores de ordem testados, quando a densidade é mínima (20%), a formulação SR mostrou-se sempre melhor do que a formulação CR. Mais especificamente, com CR não foi possível resolver nenhuma instância, enquanto que com SR foi possível revolver a maioria das instâncias.

Realizamos experimentos com grafos de ordem maior: entre 60 e 80. Para o caso de pesos aleatórios menos espaçados, consideramos densidades de até 50%. Para pesos unitários, variamos a densidade entre 20% e 80%. Consideramos valores de densidade maiores para pesos unitários, em decorrência do comportamento das curvas nas densidades maiores apresentadas nas Figuras 7.1 e 7.2.

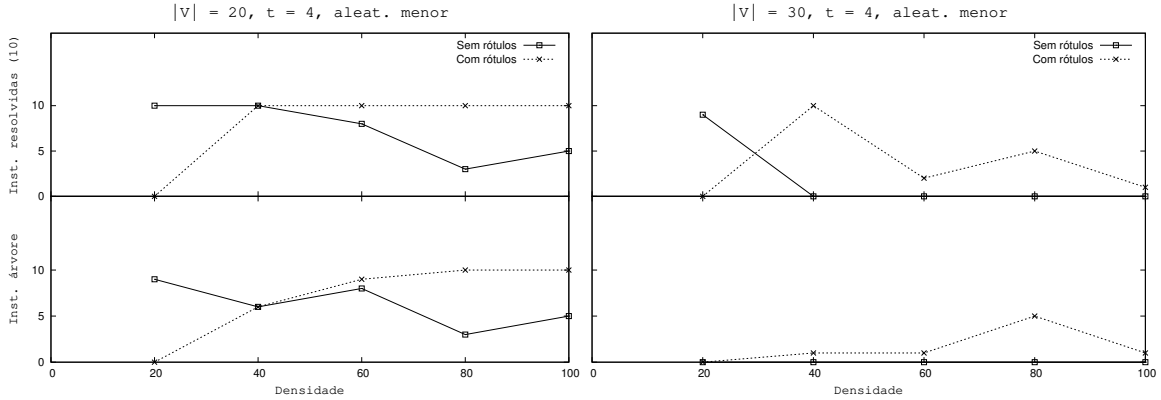

Figura 7.7: *Quantidade de instâncias resolvidas para $t = 4$, peso aleatório menos espaçado, ordem baixa e com tempo limite de 3600 segundos*

Para cada combinação, testamos 10 instâncias. Como mencionado no início do capítulo, o limite de tempo para este cenário foi de 3600 segundos.

Considerando pesos aleatórios menos espaçados, à medida que a ordem e/ou densidade e/ou fator de dilatação cresce, o número de instâncias resolvidas por CR decresce, como vemos na Tabela 7.1. Para o fator de dilatação quatro, ocultamos as linhas para as combinações para as quais não foi possível resolver nenhuma instância. Para a formulação SR, o comportamento é semelhante ao descrito acima, ilustrado na Tabela 7.2. Esta tabela não mostra dados para o fator de dilatação quatro pois nenhuma instância foi resolvida. Observamos que o comportamento do resolvedor (CPLEX) utilizando ambas as formulações para ordem e limite de tempo maiores é semelhante ao caso em que a ordem e o limite de tempo são menores, como abordado anteriormente.

t	Densidade (%)	Ordem	# Inst. resolvidas	Tempo (s)
3	20	60	10	1
	30	60	10	6
	40	60	9	14
	50	60	0	0
	20	70	10	3
	30	70	10	11
	40	70	0	0
	50	70	0	0
	20	80	10	8
	30	80	5	24
	40	80	0	0
	50	80	0	0
4	20	60	9	21
	20	70	7	66
	20	80	4	169

Tabela 7.1: *Quantidade de instâncias resolvidas com a formulação CR para peso aleatório menos espaçado*

t	Densidade (%)	Ordem	# Inst. resolvidas	Tempo (s)
3	20	60	10	9
	30	60	6	190
	40	60	1	303
	50	60	0	0
	20	70	10	87
	30	70	4	529
	40	70	0	0
	50	70	0	0
	20	80	10	160
	30	80	1	610
	40	80	0	0
	50	80	0	0

Tabela 7.2: *Quantidade de instâncias resolvidas com a formulação SR para peso aleatório menos espaçado*

Para o caso unitário, enquanto CR só conseguiu resolver algumas instâncias para o fator de dilatação três, SR só conseguiu resolver algumas instâncias para o fator de dilatação quatro (veja Tabelas 7.3 e 7.4) Em ambos os casos, só instâncias com densidades mínimas (20%) puderam ser resolvidas.

t	Densidade (%)	Ordem	# Inst. resolvidas	Tempo (s)
3	20	60	5	14
	20	70	1	32
	20	80	1	37

Tabela 7.3: *Quantidade de instâncias resolvidas com a formulação CR para peso unitário*

t	Densidade (%)	Ordem	# Inst. resolvidas	Tempo (s)
4	20	60	5	185
	20	70	1	618

Tabela 7.4: *Quantidade de instâncias resolvidas com a formulação SR para peso unitário*

Analisamos também os casos em que os fatores de dilatação são baixos. Mais especificadamente, executamos simulações para os valores 1,1 e 2. Para ambos os valores, só consideramos os casos em que os pesos são aleatórios (tanto os menos espaçados como os mais espaçados). Lembre-se que, para o fator de dilatação dois, o problema é fácil para o caso unitário (veja Tabela 3.1). Variamos as densidades entre 20% e 100%. Variamos as ordens entre 20 e 50. Para cada combinação, consideramos 20 instâncias (e tempo limite de execução de 1800 segundos).

Com a formulação CR todas as instâncias foram resolvidas em menos de três segundos. Apesar disso, nenhuma das instâncias consideradas admite uma árvore t-spanner como solução. A formulação SR teve pior desempenho também neste cenário. Pela Figura 7.8, percebemos que o tempo de execução foi bem diferente do que obtivemos com a formulação CR. Além disso, notamos que SR não escala bem com o aumento da densidade e/ou com o aumento da ordem. Ademais, para a combinação de ordem 50 com densidade pelo menos 80 e com pesos aleatórios menos espaçados,

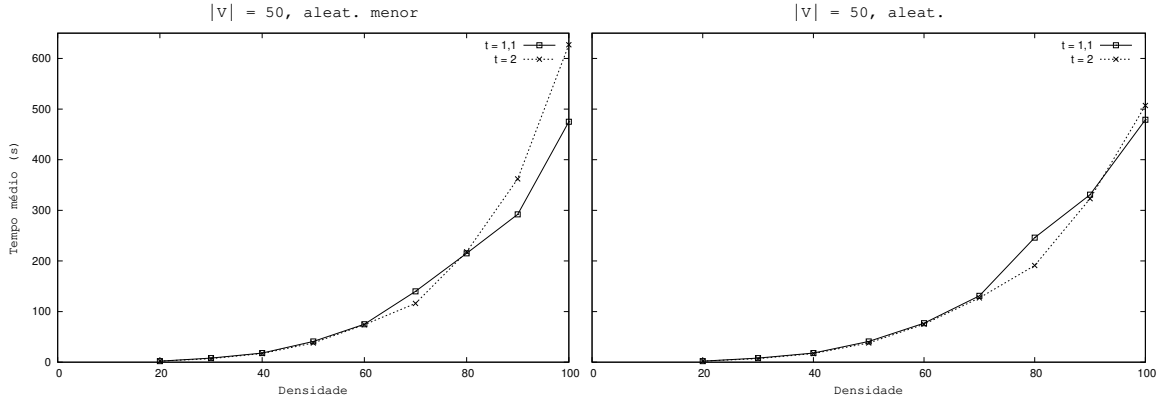

Figura 7.8: *Tempo médio para instâncias resolvidas com a formulação SR para* $|V| = 50$ *e* t *baixo*

algumas das instâncias não foram resolvidas, piorando a situação com o aumento da densidade. Isto nos dá um indicativo de que, para este tempo limite de execução, para ordens maiores do que 50 e densidades altas, provavelmente já não seria possível resolver tais instâncias.

Tendo por base os experimentos que realizamos em diversos cenários, apresentamos na Tabela 7.5 a nossa recomendação quanto ao uso de cada formulação, dependendo dos tipos das instâncias. Representamos os valores (representados por adjetivos) dos parâmetros por meio dos seguintes grupos:

- Ordem (Ord.): *pequena* para valores até 40; *grande* para valores maiores do que 40;
- Densidade (Dens.): *baixa* para valores até 40%; *alta* para valores maiores do que 40%;
- Fator de dilatação (t): *baixo* para valores até três; *alto* para o valor quatro;
- Peso (P.): *unitário* (unit.); *aleatório* (aleat.) para pesos aleatórios menos espaçados e mais espaçados.

Na Tabela 7.5, para cada combinação de valores de parâmetros, é indicada a formulação que achamos mais adequada. Quando duas formulações são mostradas, a segunda (dentro de parênteses) é utilizada em situações específicas. Quando o sinal de igual (=) é mostrado, significa que o desempenho das duas formulações são semelhantes.

Pelos experimentos, observamos que quanto maior a variação no peso das arestas, maior a vantagem em utilizar CR (podemos observar isso nas colunas da Tabela 7.5 relativas a pesos aleatórios). O domínio de CR fica claro quando se considera pesos aleatórios mais espaçados (apenas no caso de pesos aleatórios menos espaçados, t alto, densidade de 20% e ordem de 50, a formulação SR mostrou-se melhor).

Para pesos unitários e com densidade alta, em geral, SR é melhor (exceto para o caso de ordem pequena). Neste caso, apenas para t baixo ($t = 3$), densidade alta e ordem 20, a formulação CR mostrou-se melhor.

O último caso específico ilustrado na Tabela 7.5 é para pesos unitários, com densidade baixa, t alto e ordem 40. Neste cenário, SR mostrou-se melhor do que CR.

Outro fato exibido na Tabela 7.5 é que, para densidades baixas, quase sempre a formulação CR é melhor. O único caso (dos oito possíveis) em que SR foi melhor foi para t alto, pesos unitários,

		t baixo		t alto	
		P. unit.	P. aleat.	P. unit.	P. aleat.
Dens. alta	Ord. grande	SR	CR	SR	=
	Ord. peq.	SR (CR)	CR	CR	CR
Dens. baixa	Ord. grande	CR	CR	SR	CR (SR)
	Ord. peq.	CR	CR	CR (SR)	CR

Tabela 7.5: *Recomendação de uso das formulações SR e CR*

Grau	4						8					
Ordem	16	32	64	80	90	100	16	32	64	80	90	100
$t = 1{,}1$	(10)	(10)	(10)	-	-	-	(10)	(10)	(10)	-	-	-
$t = 2$	(10)	(10)	(10)	-	-	-	(10)	(10)	6(10)	-	-	-
$t = 3$	(10)	(10)	10(10)	8(10)	8(10)	202(10)	(10)	290(10)	134 (7)	591 (8)	372 (6)	1147 (5)
$t = 4$	(10)	(10)	100(10)	-	-	-	23(10)	710 (6)	1157 (1)	-	-	-

Tabela 7.6: *Quantidade de instâncias resolvidas por B&P e tempo médio de execução para peso aleatório*

densidade baixa e ordem grande.

7.4 Resultados para t-spanner de peso mínimo

Nesta seção apresentamos os resultados dos experimentos para testar a implementação do algoritmo de *branch-and-price* para o problema de t-spanner de peso mínimo, representada pela sigla *B&P*. Nós não apresentaremos experimentos para a formulação apresentada no Capítulo 5 pois alguns experimentos iniciais mostraram que a formulação não apresentava um bom desempenho. Com relação à implementação do algoritmo de *branch-and-price*, utilizamos como base os mesmos experimentos realizados por Sigurd e Zachariasen [SZ04]. Além dos valores de parâmetros utilizados pelos autores (que são subconjuntos dos valores apresentados na Seção 7.2), adicionamos novos valores de parâmetros. Mais especificamente, para a ordem dos grafos, em alguns casos, além dos valores do conjunto $\{16, 32, 64\}$, também testamos os valores 80, 90 e 100. Para o fator de dilatação, além dos valores do conjunto $\{1{,}1; 2; 4\}$, também incluímos o valor três. Além destes novos valores de parâmetros, também variamos a densidade do grafo, entre valores de 20% e 80%. Por fim, consideramos experimentos com mais tempo limite de execução, passando a ter como limite o valor de 3600 segundos. Como mencionado na Seção 7.2, para cada combinação de parâmetros foram criadas 10 instâncias. Ao longo desta seção, quando nos referirmos aos pesos aleatórios, estaremos nos referindo apenas à classe dos mais espaçados.

Inicialmente apresentamos os resultados computacionais obtidos com o B&P, para instâncias cujo parâmetro variado foi o grau médio. Posteriormente mencionamos os resultados para instâncias

Grau	4						8					
Ordem	16	32	64	80	90	100	16	32	64	80	90	100
$t = 1{,}1$	(10)	(10)	(10)	-	-	-	(10)	(10)	(10)	-	-	-
$t = 2$	(10)	(10)	(10)	-	-	-	(10)	(10)	5(10)	-	-	-
$t = 3$	(10)	(10)	(10)	5(10)	(10)	8(10)	(10)	49(10)	675 (9)	745 (9)	596(10)	667 (8)
$t = 4$	(10)	6(10)	312(10)	-	-	-	164(10)	1708 (7)	0	-	-	-

Tabela 7.7: *Instâncias resolvidas por B&P e tempo médio de execução para distância euclidiana*

Grau	4			8		
Ordem	16	32	64	16	32	64
$t = 2$	(10)	45 (9)	359 (7)	1794 (3)	0	0
$t = 3$	92 (10)	674 (4)	0	0	0	0
$t = 4$	10 (10)	0	0	1266 (10)	0	0

Tabela 7.8: *Quantidade de instâncias resolvidas e tempo de execução para peso unitário*

de densidades variadas.

As Tabelas 7.6 e 7.7 e 7.8 mostram a quantidade de instâncias resolvidas (de um total de 10), juntamente com o tempo médio de execução das instâncias resolvidas. A quantidade de instâncias resolvidas está dentro de parênteses, enquanto o tempo médio é mostrado ao lado esquerdo. Tempos médios menores do que 5 segundos não são mostrados nas tabelas. Quando nenhuma instância é resolvida, é mostrado o valor zero. Para as combinações em que não houve experimentos, aparece um traço (−).

Pelas Tabelas 7.6 e 7.7 podemos observar que as instâncias são resolvidas rapidamente quando o fator de dilatação é 1,1. Ainda considerando tanto pesos aleatórios como distâncias euclidianas, e fator de dilatação dois, todas as instâncias são resolvidas rapidamente, mas já há uma diferença considerável para instâncias de grau médio 8 e ordem 64. Acreditamos que, a partir desta ordem (e para grau médio oito), o tempo de execução aumentará drasticamente e o número de instâncias resolvidas cairá. A partir do fator três, os resultados começam a diferir para os dois tipos de peso. Para pesos aleatórios, a partir da ordem 100, os tempos aumentam consideravelmente. Além disso, para grau médio oito, é perceptível a queda no número de instâncias resolvidas à medida que aumenta a ordem. Já para as distâncias euclidianas, a implementação escala bem para grau médio quatro; e para grau médio oito, apesar do aumento no tempo, percebemos que ainda podemos resolver instâncias de ordens maiores. Considerando o fator de dilatação $t = 4$, percebemos que a implementação do B&P se comporta melhor quando os pesos são aletórios. Para grau médio oito, para ambos os tipos de peso, percebemos que a ordem 64 é o limite para o tempo de execução que consideramos.

Para o caso de pesos unitários, os resultados são semelhantes aos apresentados nos experimentos realizados por Sigurd e Zachariasen (veja Tabela 7.8). Para o fator de dilatação dois, tanto para o grau quatro (e ordem 64) como para grau oito (e ordem 16), foi possível resolver instâncias, diferentemente do que ocorreu nos experimentos de Sigurd e Zachariasen.

Vamos agora analisar o desempenho da implementação ao variar a densidade dos grafos. Assim como quando o parâmetro variado era o grau médio, as instâncias com ambos os tipos de peso nas arestas são resolvidas rapidamente, mesmo para densidades altas, quando o fator de dilatação é 1,1. Para os fatores de dilatação dois e três, as Figura 7.9 e Figura 7.10 mostram o tempo médio

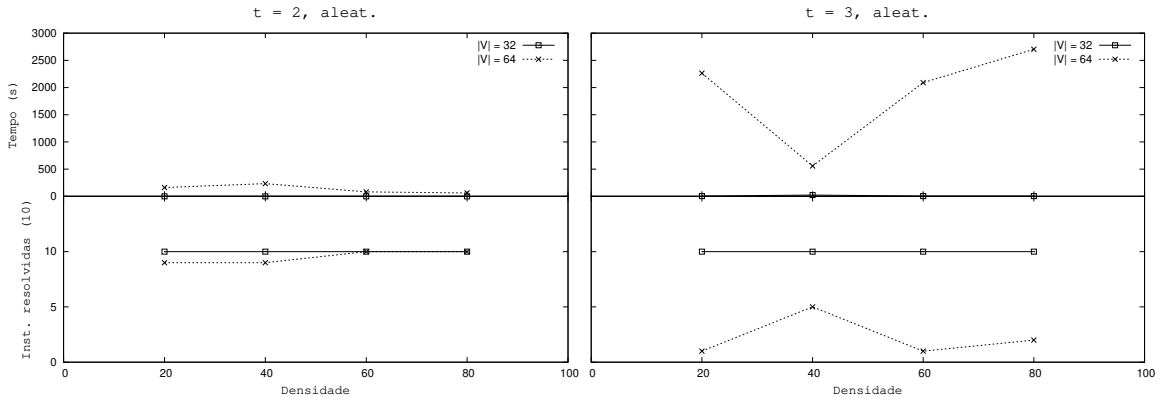

Figura 7.9: *Tempo médio de execução e quantidade de instâncias resolvidas por B&P para peso aleatório*

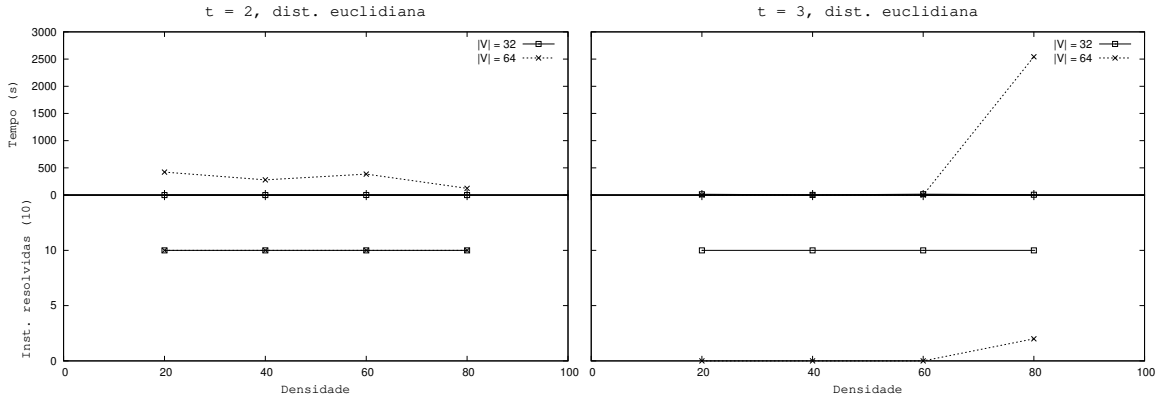

Figura 7.10: *Tempo médio de execução e quantidade instâncias resolvidas por B&P para distância euclidiana*

de execução e a quantidade de instâncias resolvidas no caso dos pesos aleatórios e no caso das distâncias euclidianas, respectivamente. Para o fator dois, no caso de quaisquer destes tipos de peso, quase todas as instâncias são resolvidas. Neste cenário, ao considerarmos ordem 64, o tempo cresce consideravelmente quando comparado à ordem 32. Para fator três, ordem 64, e ambos os tipos de peso, ocorre uma queda considerável (quando comparado à ordem 32) no número de instâncias que podem ser resolvidas e um aumento substancial no tempo de execução. Entre os dois tipos de peso, a implementação do B&P tem um desempenho melhor no caso de pesos aleatórios.

Ainda com relação aos grafos com ambos os pesos nas arestas, a Figura 7.11 mostra os resultados para o fator de dilatação quatro. Resultados para grafos de ordem 64 não são mostrados pois não foi possível resolver nenhuma instância. Para ordens menores, assim como para os resultados apresentados para $t = 3$, a implementação do B&P é melhor quando o peso das arestas é aleatório.

No caso dos pesos unitários, todas as instâncias são resolvidas rapidamente quando a ordem do grafo é 16, exceto quando a densidade é 80% (quando então o tempo de execução cresce consideravelmente). Para $t = 2$, a implementação consegue resolver algumas instâncias de ordem 32, mas com um tempo bastante alto. Todavia, não resolve nenhuma instância de ordem 64. Para valores maiores de fator de dilatação, nenhuma instância é resolvida quando a ordem é pelo menos 32.

Quando o fator de dilatação é grande (pelo menos três) e/ou a densidade é grande, em geral, o

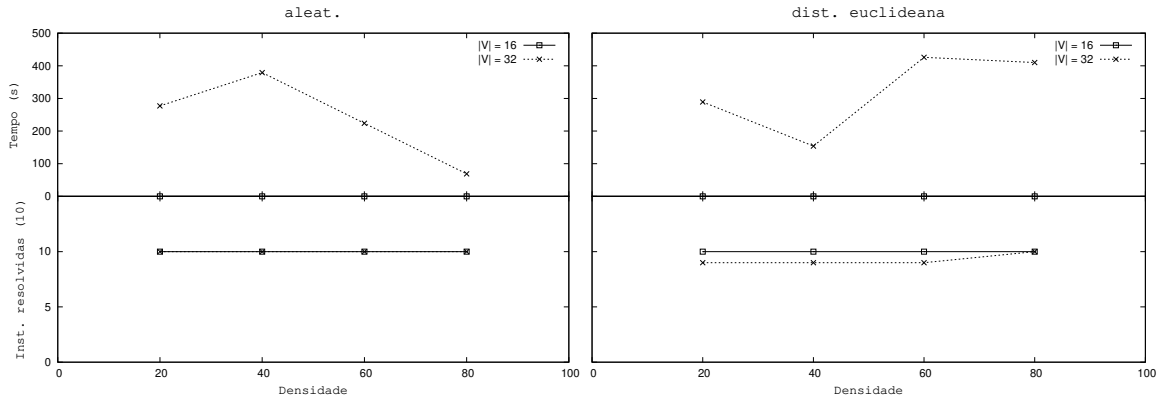

Figura 7.11: *Tempo médio de execução e quantidade instâncias resolvidas por B&P para* $t = 4$ *e arestas com peso*

número de soluções viáveis (t-spanners) cresce muito, exigindo um tempo de execução bem grande. Os diversos testes computacionais dão ao leitor uma indicação de tipos de instâncias que a implementação do B&P pode resolver. Mas, é difícil prever como será o desempenho da implementação em instâncias que não são do tipo que consideramos. Em breve, deixaremos disponível na internet os códigos de nossa implementação para que os interessadas possam fazer uso.

Capítulo 8

Considerações finais

Neste capítulo tecemos algumas considerações sobre nossas contribuições para os problemas que focamos, e mencionamos algumas linhas ou questões que poderiam ser investigadas.

No Capítulo 4, propusemos duas formulações lineares inteiras para o problema da árvore t-spanner de peso mínimmo (MWTSP). Estas são as primeiras formulações lineares específicas para este problema. Elas são baseadas numa ideia interessante, que é a de modelar via arborescências (requerendo lidar com digrafos), que facilita a identificação dos caminhos entre cada par de vértices na árvore (que resulta da "sobreposição das arborescências"). Ambas as formulações são de tamanho polinomial no tamanho de G. Elas se diferem quanto ao uso ou não de variáveis (rótulos) para representar as distâncias entre os pares de vértices. Os experimentos computacionais mostraram que a formulação com rótulos tem um desempenho melhor (do que a formulação sem rótulos explícitos). Essa foi uma descoberta que os experimentos computacionais forneceram, mostrando que, apesar de usar mais variáveis, em geral, vale a pena usar a formulação com rótulos. Neste caso também seria interessante encontrar inequações válidas para fortalecer a formulação linear correspondente.

Na Seção 4.2, abordamos uma heurística para o MWTSP, mas nos experimentos realizados a heurística foi desabilitada. Seria interessante analisar em quais cenários ela é útil.

No Capítulo 5 apresentamos uma formulação linear inteira para o problema de t-spanner de peso mínimo (MWSP), sobre a qual nada mencionamos em termos de testes computacionais ou outros aspectos. Nesta formulação, não existe variável associada a caminhos, uma ideia que seria natural, já que restrições sobre comprimentos de caminhos têm que ser impostas. As variáveis usadas, além daquelas associadas às arestas, são indexadas por pares formados por arestas e cortes do grafo. Estas últimas levam a uma formulação com um número exponencial de restrições, cujo problema de separação — considerando o programa linear relaxado correspondente — pode ser resolvido em tempo polinomial (no tamanho de V). Este fato, a princípio promissor, acabou não se mostrando muito útil: alguns experimentos iniciais mostraram que a formulação não apresentava um bom desempenho. Nesta direção, uma das ideias a ser explorada mais sistematicamente seria a de tentar descobrir inequações válidas (preferencialmente facetas) para fortalecer o programa relaxado. Para isso, poderíamos utilizar *softwares* como Porta [Sch18], Panda [LR15] e Skeleton [Zol12]. Embora não tenhamos insistido muito nesta direção, pois focamos outras alternativas, talvez seja uma direção que vale a pena ser considerada.

No Capítulo 6 apresentamos uma implementação para um algoritmo de *branch-and-price* para

este mesmo problema (o MWSP). O algoritmo foi proposto por Sigurd e Zachariasen [SZ04], mas esses autores não disponibilizaram o código. Na implementação que apresentamos aqui utilizamos pré-processamentos e heurísticas para o cálculo de limitantes primais e duais. Para o RMP associado ao programa linear considerado, as restrições (6.5) do RMP podem ser removidas para encontrar mais rapidamente uma solução viável. Como mencionado por Sigurd e Zachariasen [SZ04], o teste para saber se a solução encontrada viola uma destas restrições é rápido. Os autores ainda observaram que somente um pequeno subconjunto das inequações precisou ser adicionado ao RMP, acelerando bastante a resolução do problema. Poderíamos fazer o mesmo, possivelmente conseguindo resultados melhores do que os apresentados no Capítulo 7. Um outro aspecto a ser explorado seria investigar outras heurísticas tanto para o cálculo de limitantes primais como de limitantes duais.

Nos experimentos realizados, os grafos de entrada são gerados de maneira aleatória, e as arestas possuem um dos três tipos de pesos adotados (veja Seção 7.2). Levando em consideração que spanners são adotados em oráculos de distâncias (veja a Seção 3.1), utilizar outros grafos de entradas como mapas que representam distâncias entre cidades seria bastante útil. Um outro cenário interessante que configura um bom exemplo de entrada são as rotas utilizadas por robôs, visto que spanners são bastante utilizados na área de planejadores de roteiros em robótica.

Por fim, realizar experimentos com um número maior de instâncias deixaria mais claro o comportamento das curvas apresentadas nos gráficos. Em alguns experimentos, percebemos que poderíamos fazer testes para valores maiores de alguns parâmetros. Conhecer os limites em termos de variação de valores dos parâmetros traria uma informação relevante a respeito das implementações.

Esperamos em breve disponibilizar os códigos das implementações que fizemos, para que pessoas interessadas nos problemas focados possam se beneficiar disso. Ao iniciar os nossos estudos, esbarramos na dificuldade de obter dados e de poder comparar nossas implementações com outras. Acreditamos assim que este trabalho poderá contribuir na direção de facilitar outros estudos sobre os problemas aqui tratados.

Referências Bibliográficas

[ABFF09] Mikhail J. Atallah, Marina Blanton, Nelly Fazio e Keith B. Frikken. Dynamic and efficient key management for access hierarchies. *ACM Trans. Inf. Syst. Secur.*, 12(3):18:1–18:43, Janeiro 2009. 18

[ACC+96] Srinivasa Rao Arikati, Danny Z. Chen, L. Paul Chew, Gautam Das, Michiel H. M. Smid e Christos D. Zaroliagis. Planar spanners and approximate shortest path queries among obstacles in the plane. Em *Proceedings of the Fourth Annual European Symposium on Algorithms*, ESA '96, páginas 514–528, London, UK, 1996. Springer-Verlag. 1, 15

[ADD+93] Ingo Althöfer, Gautam Das, David Dobkin, Deborah Joseph e José Soares. On sparse spanners of weighted graphs. *Discrete Comput. Geom.*, 9(1):81–100, Janeiro 1993. 2, 18, 19, 22, 26, 49

[AGM12] Kook Jin Ahn, Sudipto Guha e Andrew McGregor. Graph sketches: Sparsification, spanners, and subgraphs. Em *Proceedings of the 31st ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, PODS '12, páginas 5–14, New York, NY, USA, 2012. ACM. 18

[APMS+99] Giorgio Ausiello, M. Protasi, A. Marchetti-Spaccamela, G. Gambosi, P. Crescenzi e V. Kann. *Complexity and Approximation: Combinatorial Optimization Problems and Their Approximability Properties*. Springer-Verlag, Berlin, Heidelberg, 1st edição, 1999. 13

[Awe85] Baruch Awerbuch. Complexity of network synchronization. *J. ACM*, 32(4):804–823, Outubro 1985. 1, 15, 16

[Bal05] Egon Balas. Projection, lifting and extended formulation in integer and combinatorial optimization. *Annals of Operations Research*, 140(1):125, Nov 2005. 8

[BBG+14] Piotr Berman, Arnab Bhattacharyya, Elena Grigorescu, Sofya Raskhodnikova, David P. Woodruff e Grigory Yaroslavtsev. Steiner transitive-closure spanners of low-dimensional posets. *Combinatorica*, 34(3):255–277, 2014. 25

[BBM+11] Piotr Berman, Arnab Bhattacharyya, Konstantin Makarychev, Sofya Raskhodnikova e Grigory Yaroslavtsev. Improved approximation for the directed spanner problem. Em *Automata, Languages and Programming*, páginas 1–12, Berlin, Heidelberg, 2011. Springer-Verlag. 17

[BBM+13] Piotr Berman, Arnab Bhattacharyya, Konstantin Makarychev, Sofya Raskhodnikova e Grigory Yaroslavtsev. Approximation algorithms for spanner problems and directed steiner forest. *Information and Computation*, 222(0):93–107, 2013. 38th International Colloquium on Automata, Languages and Programming (ICALP 2011). 25

[BCD99] Andreas Brandstädt, Victor Chepoi e Feodor Dragan. Distance approximating trees for chordal and dually chordal graphs. *J. Algorithms*, 30(1):166–184, Janeiro 1999. 20, 23

[BDL+07] Andreas Brandstadt, Feodor F. Dragan, Hoang-Oanh Le, Van Bang Le e Ryuhei Uehara. Tree spanners for bipartite graphs and probe interval graphs. *Algorithmica*, 47(1):27–51, Janeiro 2007. 20, 24

[BDLL04] Andreas Brandstädt, Feodor F. Dragan, Hoàng-Oanh Le e Van Bang Le. Tree spanners on chordal graphs: complexity and algorithms. *Theor. Comput. Sci.*, 310(1-3):329–354, Janeiro 2004. 7, 20

[BGJ+09] Arnab Bhattacharyya, Elena Grigorescu, Kyomin Jung, Sofya Raskhodnikova e David P. Woodruff. Transitive-closure spanners. Em *Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '09, páginas 932–941, Philadelphia, PA, USA, 2009. SIAM. 17, 18

[BGJ+10] Arnab Bhattacharyya, Elena Grigorescu, Madhav Jha, Kyomin Jung, Sofya Raskhodnikova e David P. Woodruff. Lower bounds for local monotonicity reconstruction from transitive-closure spanners. Em *Proceedings of the 13th International Conference on Approximation, and 14 the International Conference on Randomization, and Combinatorial Optimization: Algorithms and Techniques*, APPROX/RANDOM'10, páginas 448–461, Berlin, Heidelberg, 2010. Springer-Verlag. 17

[BH97] Ulrik Brandes e Dagmar Handke. NP-completeness results for minimum planar spanners. Em *Graph-Theoretic Concepts in Computer Science*, volume 1335 of *Lecture Notes in Computer Science*, páginas 85–99. Springer-Verlag, 1997. 23, 24

[BJN+98] Cynthia Barnhart, Ellis L. Johnson, George L. Nemhauser, Martin W. P. Savelsbergh e Pamela H. Vance. Branch-and-price: Column generation for solving huge integer programs. *Operations Research*, 46(3):316–329, 1998. 11

[BKKL17] Ruben Becker, Andreas Karrenbauer, Sebastian Krinninger e Christoph Lenzen. Near-Optimal Approximate Shortest Paths and Transshipment in Distributed and Streaming Models. Em *31st International Symposium on Distributed Computing (DISC 2017)*, volume 91 of *Leibniz International Proceedings in Informatics (LIPIcs)*, páginas 7:1–7:16, Dagstuhl, Germany, 2017. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. 16, 18

[BKMP10] Surender Baswana, Telikepalli Kavitha, Kurt Mehlhorn e Seth Pettie. Additive spanners and (α, β)-spanners. *ACM Trans. Algorithms*, 7(1):5:1–5:26, Dezembro 2010. 47

[BLS99] Andreas Brandstädt, Van Bang Le e Jeremy P. Spinrad. *Graph Classes: A Survey*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1999. 7

[BLWN17] G. Borradaile, H. Le e C. Wulff-Nilsen. Minor-free graphs have light spanners. Em *2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS)*, páginas 767–778, Oct 2017. 26

[BM08] A. Bondy e U.S.R. Murty. *Graph Theory*. Graduate Texts in Mathematics. Springer-Verlag London, 2008. 6

[Bol98] B. Bollobás. *Modern Graph Theory*. Graduate texts in mathematics. Springer, 1998. 6

[Bon89] J. A. Bondy. Trigraphs. *Discrete Math.*, 75(1-3):69–79, Setembro 1989. 20

[Bra17] H. Braga. Minimum weight tree spanner problem. Em *II Encontro de Teoria da Computação (ETC)*, volume 2, São Paulo, SP, Brasil, 2017. SBC. 36

[BRR10] Piotr Berman, Sofya Raskhodnikova e Ge Ruan. Finding sparser directed spanners. Em *FSTTCS*, volume 8 of *LIPIcs*, páginas 424–435. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2010. 25

[BT97] Dimitris Bertsimas e John Tsitsiklis. *Introduction to Linear Optimization*. Athena Scientific, 1st edição, 1997. 45

[BW15] Gregory Bodwin e Virginia Vassilevska Williams. Very sparse additive spanners and emulators. Em *Proceedings of the 2015 Conference on Innovations in Theoretical Computer Science*, ITCS '15, páginas 377–382, New York, NY, USA, 2015. ACM. 26

[Cai92] Leizhen Cai. *Tree spanners: spanning trees that approximate distances*. Tese de Doutorado, University of Toronto, Toronto, Ont., Canada, 1992. UMI Order No. GAXNN-73862. 20, 22

[Cai94] Leizhen Cai. NP-completeness of minimum spanner problems. *Discrete Applied Mathematics*, 48(2):187 – 194, 1994. 1, 18, 19, 22, 24

[CC95] Leizhen Cai e Derek G. Corneil. Tree spanners. *SIAM J. Discret. Math.*, 8(3):359–387, Agosto 1995. 1, 18, 20, 21, 22, 24, 27

[CCZ13] Michele Conforti, Gérard Cornuéjols e Giacomo Zambelli. Extended formulations in combinatorial optimization. *Annals OR*, 204(1):97–143, 2013. 31

[CDK12] Eden Chlamtac, Michael Dinitz e Robert Krauthgamer. Everywhere-sparse spanners via dense subgraphs. *CoRR*, abs/1205.0144, 2012. 25

[CDNS92] Barun Chandra, Gautam Das, Giri Narasimhan e José Soares. New sparseness results on graph spanners. Em *Proceedings of the Eighth Annual Symposium on Computational Geometry*, SCG '92, páginas 192–201, New York, NY, USA, 1992. ACM. 19

[Che86] P Chew. There is a planar graph almost as good as the complete graph. Em *Proceedings of the Second Annual Symposium on Computational Geometry*, SCG '86, páginas 169–177, New York, NY, USA, 1986. ACM. 22

[Che89] L. P. Chew. There are planar graphs almost as good as the complete graph. *J. Comput. Syst. Sci.*, 39(2):205–219, Outubro 1989. 22

[CHKPY16] Keren Censor-Hillel, Telikepalli Kavitha, Ami Paz e Amir Yehudayoff. Distributed construction of purely additive spanners. Em *Distributed Computing*, páginas 129–142, Berlin, Heidelberg, 2016. Springer-Verlag. 26

[CK93] Leizhen Cai e J. Mark Keil. Degree-bounded spanners. *Parallel Processing Letters*, 3:457–468, 1993. 23

[CK94] Leizhen Cai e Mark Keil. Spanners in graphs of bounded degree. *Networks*, 24(4):233–249, 1994. 22, 24

[Cla99] Jens Clausen. Branch and bound algorithms – principles and examples. Relatório técnico, Department of Computer Science, University of Copenhagen, 1999. 51

[CLRS09] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest e Clifford Stein. *Introduction to Algorithms, Third Edition*. The MIT Press, 3rd edição, 2009. 29

[CMS13] Michael S. Crouch, Andrew McGregor e Daniel Stubbs. Dynamic graphs in the sliding-window model. Em *Algorithms – ESA 2013*, páginas 337–348, Berlin, Heidelberg, 2013. Springer-Verlag. 18

[CWN16] Shiri Chechik e Christian Wulff-Nilsen. Near-optimal light spanners. Em *Proceedings of the Twenty-seventh Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '16, páginas 883–892, Philadelphia, PA, USA, 2016. SIAM. 19, 26

[DFG11a] Feodor F. Dragan, Fedor V. Fomin e Petr A. Golovach. Approximation of minimum weight spanners for sparse graphs. *Theor. Comput. Sci.*, 412(8-10):846–852, Março 2011. 24

[DFG11b] Feodor F. Dragan, Fedor V. Fomin e Petr A. Golovach. Spanners in sparse graphs. *J. Comput. Syst. Sci.*, 77(6):1108–1119, Novembro 2011. 20

[DH98] Michael J. Demmer e Maurice Herlihy. The arrow distributed directory protocol. Em *Proceedings of the 12th International Symposium on Distributed Computing*, DISC '98, páginas 119–133, London, UK, 1998. Springer-Verlag. 16

[Die17] Reinhard Diestel. *Graph Theory*, volume 173 of *Graduate Texts in Mathematics*. Springer, Heidelberg; New York, fifth edição, 2017. 6

[DK11a] Michael Dinitz e Robert Krauthgamer. Directed spanners via flow-based linear programs. Em *Proceedings of the Forty-third Annual ACM Symposium on Theory of Computing*, STOC '11, páginas 323–332, New York, NY, USA, 2011. ACM. 2, 24, 38

[DK11b] Michael Dinitz e Robert Krauthgamer. Fault-tolerant spanners: Better and simpler. Em *In PODC*, 2011. 24

[DK14] Feodor F. Dragan e Ekkehard Köhler. An approximation algorithm for the tree t-spanner problem on unweighted graphs via generalized chordal graphs. *Algorithmica*, 69(4):884–905, Agosto 2014. 25

[DKR15] Michael Dinitz, Guy Kortsarz e Ran Raz. Label cover instances with large girth and the hardness of approximating basic k-spanner. *ACM Trans. Algorithms*, 12(2):25:1–25:16, Dezembro 2015. 18, 19

[DL05] Jacques Desrosiers e Marco E. Lübbecke. *A Primer in Column Generation*, páginas 1–32. Springer US, Boston, MA, 2005. 11

[DS88] Martin Desrochers e Francois Soumis. A generalized permanent labelling algorithm for the shortest path problem with time windows. *INFOR: Information Systems and Operational Research*, 26(3):191–212, 1988. 46

[DZ00] W. Duckworth e M. Zito. Sparse hypercube 3-spanners. *Discrete Applied Mathematics*, 103(1–3):289 – 295, 2000. 23, 24

[Elk05] Michael Elkin. Computing almost shortest paths. *ACM Trans. Algorithms*, 1(2):283–323, Outubro 2005. 16

[EN17] Michael Elkin e Ofer Neiman. Efficient algorithms for constructing very sparse spanners and emulators. Em *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '17, páginas 652–669, Philadelphia, PA, USA, 2017. SIAM. 26

[ENS15] Michael Elkin, Ofer Neiman e Shay Solomon. Light spanners. *SIAM J. Discrete Math.*, 29(3):1312–1321, 2015. 19, 26

[Ent18] Scilab Enterprises. Scilab. Website, 2018. https://www.scilab.org. 55

[EP04] Yuval Emek e David Peleg. Approximating minimum max-stretch spanning trees on unweighted graphs. Em *Proceedings of the fifteenth annual ACM-SIAM symposium on Discrete algorithms*, SODA '04, páginas 261–270, Philadelphia, PA, USA, 2004. SIAM. 23

[EP05] Michael Elkin e David Peleg. Approximating k-spanner problems for $k > 2$. *Theoretical Computer Science*, 337(1–3):249 – 277, 2005. 19

[EP07] Michael Elkin e David Peleg. The hardness of approximating spanner problems. *Theor. Comp. Sys.*, 41(4):691–729, Dezembro 2007. 19

[EP08] Y. Emek e D. Peleg. Approximating minimum max-stretch spanning trees on unweighted graphs. *SIAM Journal on Computing*, 38(5):1761–1781, 2008. 21, 23, 26

[EP16] Michael Elkin e Seth Pettie. A linear-size logarithmic stretch path-reporting distance oracle for general graphs. *ACM Trans. Algorithms*, 12(4):50:1–50:31, Agosto 2016. 16, 26

[ES16] Michael Elkin e Shay Solomon. Fast constructions of lightweight spanners for general graphs. *ACM Trans. Algorithms*, 12(3):29:1–29:21, Abril 2016. 18

[EZ06] Michael Elkin e Jian Zhang. Efficient algorithms for constructing $(1 + \epsilon,\beta)$-spanners in the distributed and streaming models. *Distributed Computing*, 18(5):375–385, Abril 2006. 16

[FK01] Sándor P. Fekete e Jana Kremer. Tree spanners in planar graphs. *Discrete Applied Mathematics*, 108(1–2):85 – 103, 2001. Workshop on Graph Theoretic Concepts in Computer Science. 20, 21, 23

[FW96] C.E. Ferreira e Y. Wakabayashi. Combinatória poliédrica e planos-de-corte faciais. Em *10a Escola de Computação*. UNICAMP - Instituto de Computação, 1996. 8, 9

[Gal01] Giulia Galbiati. On min-max cycle bases. Em *Algorithms and Computation*, volume 2223 of *Lecture Notes in Computer Science*, páginas 116–123. Springer-Verlag, 2001. 21

[GJ90] Michael R. Garey e David S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1990. 28, 46

[GLS88] M. Grötschel, L. Lovász e A. Schrijver. *Geometric algorithms and combinatorial optimization*. Algorithms and combinatorics. Springer-Verlag, 1988. 9

[GP17] Ofer Grossman e Merav Parter. Improved Deterministic Distributed Construction of Spanners. Em *31st International Symposium on Distributed Computing (DISC 2017)*, volume 91 of *Leibniz International Proceedings in Informatics (LIPIcs)*, páginas 24:1–24:16, Dagstuhl, Germany, 2017. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. 26

[GYZ13] Jonathan L. Gross, Jay Yellen e Ping Zhang. *Handbook of Graph Theory, Second Edition*. Chapman & Hall/CRC, 2nd edição, 2013. 7

[IBM18] IBM. Ibm ilog cplex. Website, 2018. https://www.ibm.com/analytics/cplex-optimizer. 55

[IZN02] Takehiro Ito, Xiao Zhou e Takao Nishizeki. Partitioning trees of supply and demand. Em *Algorithms and Computation*, volume 2518 of *Lecture Notes in Computer Science*, páginas 612–623. Springer-Verlag, 2002. 25

[Kel17] A.A. Keller. *Mathematical Optimization Terminology: A Comprehensive Glossary of Terms*. ACADEMIC PRESS, 2017. 11

[Kle06] Philip N. Klein. A subset spanner for planar graphs, with application to subset tsp. Em *Proceedings of the Thirty-eighth Annual ACM Symposium on Theory of Computing*, STOC '06, páginas 749–756, New York, NY, USA, 2006. ACM. 18

[KNPR15] Hartmut Klauck, Danupon Nanongkai, Gopal Pandurangan e Peter Robinson. Distributed computation of large-scale graph problems. Em *Proceedings of the Twenty-sixth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '15, páginas 391–410, Philadelphia, PA, USA, 2015. SIAM. 26

[Knu17] Mathias Baek Tejs Knudsen. Additive Spanners and Distance Oracles in Quadratic Time. Em *44th International Colloquium on Automata, Languages, and Programming (ICALP 2017)*, volume 80 of *Leibniz International Proceedings in Informatics (LIPIcs)*, páginas 64:1–64:12, Dagstuhl, Germany, 2017. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. 16, 26

[Kor01] G. Kortsarz. On the hardness of approximating spanners. *Algorithmica*, 30(3):432–450, 2001. 18, 19

[KP94] G. Kortsarz e D. Peleg. Generating sparse 2-spanners. *Journal of Algorithms*, 17(2):222 – 236, 1994. 18, 19

[KP98] Guy Kortsarz e David Peleg. Generating low-degree 2-spanners. *SIAM J. Comput.*, 27(5):1438–1456, Outubro 1998. 23, 25

[KV12] Bernhard Korte e Jens Vygen. *Combinatorial Optimization: Theory and Algorithms*. Springer-Verlag, 5th edição, 2012. 41

[KW14] Michael Kapralov e David Woodruff. Spanners and sparsifiers in dynamic streams. Em *Proceedings of the 2014 ACM Symposium on Principles of Distributed Computing*, PODC '14, páginas 272–281, New York, NY, USA, 2014. ACM. 18

[LR15] Stefan Lörwald e Gerhard Reinelt. Panda: a software for polyhedral transformations. *EURO Journal on Computational Optimization*, 3(4):297–308, Novembro 2015. 67

[LW08] Christian Liebchen e Gregor Wünsch. The zoo of tree spanner problems. *Discrete Appl. Math.*, 156(5):569–587, Março 2008. 24

[MB11a] J. D. Marble e K. E. Bekris. Computing spanners of asymptotically optimal probabilistic roadmaps. Em *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, páginas 4292–4298, 2011. 1, 17

[MB11b] James D. Marble e Kostas E. Bekris. Asymptotically near-optimal is good enough for motion planning. Em *Proc. of the 15th International Symposium on Robotics Research (ISRR-11)*, 2011. 1, 17

[MB12] J. D. Marble e K. E. Bekris. Towards small asymptotically near-optimal roadmaps. Em *2012 IEEE International Conference on Robotics and Automation*, páginas 2557–2562, May 2012. 17

[MB13] James D. Marble e Kostas E. Bekris. Asymptotically near-optimal planning with probabilistic roadmap spanners. *Trans. Rob.*, 29(2):432–444, Abril 2013. 17

[McG14] Andrew McGregor. Graph stream algorithms: A survey. *SIGMOD Rec.*, 43(1):9–20, Maio 2014. 18

[MJSS16] David R. Morrison, Sheldon H. Jacobson, Jason J. Sauppe e Edward C. Sewell. Branch-and-bound algorithms: A survey of recent advances in searching, branching, and pruning. *Discrete Optimization*, 19:79–102, 2016. 11, 50, 51

[MTZ60] C. E. Miller, A. W. Tucker e R. A. Zemlin. Integer programming formulation of traveling salesman problems. *J. ACM*, 7(4):326–329, Outubro 1960. 34

[MVR96] M.S. Madanlal, G. Venkatesan e C.Pandu Rangan. Tree 3-spanners on interval, permutation and regular bipartite graphs. *Information Processing Letters*, 59(2):97–102, 1996. 20, 23, 24

[NR15] N.S. Narayanaswamy e G. Ramakrishna. Tree t-spanners in outerplanar graphs via supply demand partition. *Discrete Applied Mathematics*, 195:104–109, 2015. 12th Cologne-Twente Workshop on Graphs and Combinatorial Optimization (CTW 2013). 20, 25

[NW99] G.L. Nemhauser e L.A. Wolsey. *Integer and combinatorial optimization*. Wiley-Interscience series in discrete mathematics and optimization. Wiley, 1999. 10, 11

[oCOE16] Egerváry Research Group on Combinatorial Optimization (EGRES). Lemon. Website, 2016. https://lemon.cs.elte.hu/trac/lemon. 55

[OP05] Carlos A. S. Oliveira e Panos M. Pardalos. A survey of combinatorial optimization problems in multicast routing. *Comput. Oper. Res.*, 32(8):1953–1981, Agosto 2005. 1, 15, 16

[Pap13] Ioannis E. Papoutsakis. *Tree Spanners of Simple Graphs*. Tese de Doutorado, University of Toronto, 2013. 20, 26

[Pap14] Ioannis Papoutsakis. Tree 3-spanners of diameter at most 5. *CoRR*, abs/1402.3573, 2014. 20

[Pap18] Ioannis Papoutsakis. Tree spanners of bounded degree graphs. *Discrete Appl. Math.*, 236(C):395–407, Fevereiro 2018. 20

[Pel00] David Peleg. *Distributed Computing: A Locality-sensitive Approach*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2000. 47, 48, 49

[PR99] David Peleg e Eilon Reshef. A variant of the arrow distributed directory with low average complexity (extended abstract). Em *Automata, Languages and Programming*, volume 1644 of *Lecture Notes in Computer Science*, páginas 615–624. Springer-Verlag, 1999. 1, 15, 16, 21

[PS89] David Peleg e Alejandro A. Schäffer. Graph spanners. *Journal of Graph Theory*, 13(1):99–116, 1989. 18, 19, 22, 24

[PU87] David Peleg e Jeffrey D. Ullman. An optimal synchronizer for the hypercube. Em *8th ACM Symposium on Principles of Distributed Computing*, PODC '87, páginas 77–85. ACM, 1987. 22

[PU88] David Peleg e Eli Upfal. A tradeoff between space and efficiency for routing tables. Em *Proceedings of the twentieth annual ACM symposium on Theory of computing*, STOC '88, páginas 43–52, New York, NY, USA, 1988. ACM. 1, 15, 16

[PU89] David Peleg e Jeffrey D. Ullman. An optimal synchronizer for the hypercube. *SIAM J. Comput.*, 18(4):740–747, Agosto 1989. 1, 15, 16, 22

[RTZ05] Liam Roditty, Mikkel Thorup e Uri Zwick. Deterministic constructions of approximate distance oracles and spanners. Em *Proceedings of the 32Nd International Conference on Automata, Languages and Programming*, ICALP'05, páginas 261–272, Berlin, Heidelberg, 2005. Springer-Verlag. 16

[RTZ08] Iam Roditty, Mikkel Thorup e Uri Zwick. Roundtrip spanners and roundtrip routing in directed graphs. *ACM Trans. Algorithms*, 4(3):29:1–29:17, Julho 2008. 16

[Sch18] Sebastian Schenker. Porta. Website, 2018. http://porta.zib.de. 67

[Som16] Christian Sommer. All-pairs approximate shortest paths and distance oracle preprocessing. Em *43rd International Colloquium on Automata, Languages and Programming (ICALP)*, páginas 55:1–13, 2016. 16, 26

[SZ04] Mikkel Sigurd e Martin Zachariasen. Construction of minimum-weight spanners. Em *Algorithms – ESA 2004*, volume 3221 of *Lecture Notes in Computer Science*, páginas 797–808. Springer-Verlag, 2004. 2, 19, 23, 38, 43, 46, 49, 53, 55, 63, 68

[TZ05] Mikkel Thorup e Uri Zwick. Approximate distance oracles. *J. ACM*, 52(1):1–24, Janeiro 2005. 16

[VRM+97] G. Venkatesan, U. Rotics, M.S. Madanlal, J.A. Makowsky e C. Pandu Rangan. Restrictions of minimum spanner problems. *Information and Computation*, 136(2):143 – 164, 1997. 20, 22, 24

[WBC15] Weifu Wang, Devin Balkcom e Amit Chakrabarti. A fast online spanner for roadmap construction. *Int. J. Rob. Res.*, 34(11):1418–1432, Setembro 2015. 17

[Wol98] L.A. Wolsey. *Integer Programming*. Wiley Series in Discrete Mathematics and Optimization. Wiley, 1998. 10, 11

[WS11] David P. Williamson e David B. Shmoys. *The Design of Approximation Algorithms*. Cambridge University Press, New York, NY, USA, 1st edição, 2011. 13

[Zol12] N. Yu. Zolotykh. New modification of the double description method for constructing the skeleton of a polyhedral cone. *Computational Mathematics and Mathematical Physics*, 52(1):146–156, Janeiro 2012. 67