Universidade Federal do ABC
Centro de Matemática, Computação e Cognição

Relatório final para o programa IC
Edital 03/2023

Algoritmos de aproximação para o problema da Árvore de Steiner

Aluno:
Paulo Victor Dias Soares

Orientador:
Carla Negri Lintzmayer

Santo André – SP
30/09/2024

Resumo

No problema da Árvore de Steiner temos um grafo com custo positivo nas arestas e um conjunto de vértices denominados terminais e queremos encontrar uma árvore que contém todos os vértices terminais e cuja soma dos custos das arestas seja a menor possível. Note que essa árvore pode conter outros vértices além dos terminais. Esse é um problema clássico e central em otimização combinatória, com diversas aplicações práticas e, infelizmente, NP-difícil. Este relatório contém detalhes sobre uma 11/6-aproximação para o problema da Árvore de Steiner e outros problemas e algoritmos de aproximação que foram estudados durante a iniciação científica do aluno.

Conteúdo

1 Introdução

O objetivo de problemas em Otimização Combinatória é encontrar a melhor solução a partir de um conjunto finito de soluções viáveis. Um exemplo clássico dessa gama de problemas é o problema da Árvore de Steiner (*Steiner Tree*), que é enfoque neste relatório. O problema foi, inicialmente, formulado por Gauss e possui diversas aplicações, tais como encontrar o comprimento mínimo de interconexões entre terminais ou construir árvores de filogenia em biologia computacional [6]. Outrossim, fica evidente o alcance de problemas a serem tratados a partir de generalizações do problema da Árvore de Steiner.

Este problema pode ser definido a partir de um grafo com custo positivo nas arestas e um conjunto de vértices denominados terminais, comumente chamados de *requeridos*. O objetivo é encontrar uma árvore que contenha todos os vértices terminais, tal qual o custo dessa árvore, sendo a soma das arestas escolhidas, seja mínimo. Além disso, vértices terminais podem aparecer na solução. Estes vértices são chamados de vértices de Steiner.

Ademais, o problema referido está presente no conjunto dos problemas NP-difíceis, isto é, a menos que P = NP, dificilmente existirá um algoritmo que devolva uma solução ótima em tempo polinomial para qualquer entrada [7]. No entanto, dada a sua relevância, é crucial encontrar uma abordagem para tratá-lo. Este é o caso dos algoritmos de aproximação, que garantem uma solução cujo custo está a um fator determinado do custo de uma solução ótima para o problema. Portanto, ainda que não se tenha o melhor resultado, tem-se garantia que será encontrado um a no máximo um fator em comparação a este último.

O restante deste relatório está dividido da seguinte forma. Na Seção 2, estão descritos conceitos fundamentais da área de estudo para entendimento completo dos problemas presentes. Mais do que isso, são conceitos necessários para descrição dos problemas. Além disso, o complemento desta seção encontra-se no Apêndice A. Ambas contam com uma síntese de um estudo voltado a Algoritmos de Aproximação em conjuntura com conceitos pontuais de Teoria dos Grafos. Na Seção 3 estão as análises para as duas aproximações estudadas e, além disso, uma análise sobre a tratabilidade da entrada para um grafo que não é métrico. Essa mesma análise foi feita para o problema do Caixeiro Viajante e de ser vista no Apêndice B. Essa é dividida entre o problema do Caixeiro Viajante e o problema do Escalonamento.

2 Fundamentação Teórica

O problema da Árvore de Steiner e outros problemas afins, que foram tratados neste projeto, são Problemas de Otimização. Para tratá-los, é necessário o estudo de conceitos

das áreas de Análise de Algoritmo e Teoria dos Grafos. Nesse sentido, esta seção é dedicada a esses conceitos fundamentais para o entendimento de tais problemas.

Foi utilizado, para compreensão conceitual dos temas de Classes de Complexidade e Redução, que foram descritos nas Seções A.2 e A.1, o livro "*The Design of Approximation Algorithms*" [7]. Além disso, o livro "*Graph Theory*" [1] foi base para a descrição conceitual relacionada a grafos, presente na Seção 2.2.

Definimos como *problema de decisão* um problema tal que para seu conjunto de instâncias de entrada sejam aceitos apenas "SIM" ou "NÃO" como resposta.

Já em um *problema de otimização*, recebemos na entrada algumas restrições que podem ser atendidas por várias soluções, que são chamadas de soluções viáveis. Sabemos também como calcular o valor de uma solução viável qualquer, de forma que buscamos aquelas que possuem o melhor valor associado, que é uma solução ótima.

2.1 Algoritmos de Aproximação

Como previamente contextualizado, muitos dos problemas da literatura fazem parte da classe dos problemas NP-difíceis. Para tais, a menos que P = NP, (i) não existe algoritmo que os resolva de forma eficiente, (ii) garantindo a otimalidade da solução (iii) para qualquer instância [7]. Dessa maneira, as abordagens no tratamento desses problemas utilizam meios que abdicam de alguma(s) condição(ões).

Nesse sentido, os algoritmos de aproximação surgem para contornar a dificuldade computacional em solucionar esses problemas. Sacrificando a otimalidade, esses algoritmos buscam eficiência em troca de uma aproximação razoável da melhor solução. Por isso, é cabível apontar um paradigma entre a perda na otimalidade e ganho em eficiência como base para esta área de estudo.

O objetivo de um algoritmo de aproximação é devolver, para qualquer instância, uma solução cujo valor seja garantidamente o mais próximo possível do valor de uma solução ótima para aquela instância. Dessa forma, a medição da qualidade desses algoritmos se dá por um fator de distância no valor de uma solução ótima para determinada instância.

Tome $\mathcal{I}$ como sendo o conjunto de instâncias para um problema de otimização combinatória $\mathcal{P}$ e $\mathcal{A}$ um algoritmo eficiente que devolve soluções viáveis para tal. Note que, para cada instância válida do problema, existe um conjunto finito de soluções viáveis. Seja $I \in \mathcal{I}$, denota-se $\mathcal{A}(I)$ o valor da solução devolvida por esse algoritmo para I. Também denota-se por $\text{OPT}_{\mathcal{P}}(I)$ o valor de uma solução ótima, cuja qual é a melhor para a entrada I do problema $\mathcal{P}$.

Sendo $\alpha > 1$ um valor qualquer, não necessariamente constante, tem-se que $\mathcal{A}$ é um algoritmo de aproximação para $\mathcal{P}$ se para qualquer $I \in \mathcal{I}$:

- $\mathcal{A}(I) \leq \alpha \text{OPT}(I)$ para um problema de minimização;
- $\mathcal{A}(I) \geq \frac{1}{\alpha}\text{OPT}(I)$ para um problema de maximização.

É dito que $\mathcal{A}$ é uma *α-aproximação*, ou mesmo uma *aproximação de fator α* para o problema.

Note que, quanto menor o valor de α, mais próxima será a solução do algoritmo em relação ao custo de uma solução ótima.

2.2 Grafos

Um *grafo* (simples) é um par (V, E), onde:

- V é um conjunto finito de elementos chamados de *vértices*;
- E é um conjunto finito de pares de elementos de V, chamados de *arestas*.

Pode-se dizer que cada vértice representa um objeto e cada aresta representa uma relação entre esses objetos. Dizemos que $|V|$ é a *ordem* do grafo. A Figura 1 representa um grafo de ordem 5, em que os vértices são representados por círculos e as arestas são representadas por linhas que conectam os círculos.

Em um grafo G qualquer, utilizamos $V(G)$ para representar o conjunto de vértices e $E(G)$ para representar o conjunto de arestas desse grafo. Denota-se a aresta $\{u, v\} \in E(G)$ simplesmente por uv. A aresta uv possui os vértices u e v como *extremos*. Além disso, a aresta uv *incide* nos vértices u e v. A existência dessa aresta implica, também, que u e v são *vizinhos* ou *adjacentes*. O conjunto $N_G(v)$, chamado *vizinhança*, contém todos os vizinhos de um vértice $v \in V(G)$. O *grau* de um vértice $v \in V(G)$, denotado por $d_G(v)$, é o número de arestas que incidem em v. Note que $d_G(v) = |N_G(v)|$.

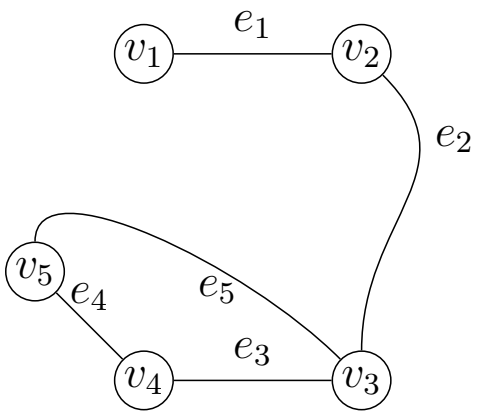

Figura 1: Representação gráfica de um grafo G com $V(G) = \{v_1, v_2, v_3, v_4, v_5\}$ e $E(G) = \{e_1, e_2, e_3, e_4, e_5\}$.

Dado um grafo G, chamamos de *subgrafo* de G qualquer grafo G' tal que $V(G') \subseteq V(G)$ e $E(G') \subseteq E(G)$. Tem-se que G' é *subgrafo gerador* de G se $V(G') = V(G)$.

Tome $S \subseteq V(G)$. Denota-se por $G[S]$ o *subgrafo induzido* de G por S, que é o grafo

tal que $V(G[S]) = S$ e $E(G[S]) = \{uv \in E(G) \colon u, v \in S\}$.

Um *passeio* é uma sequência $W = (v_1, v_2, \ldots, v_{k+1})$ tal que $v_i \in V(G)$ para $1 \leq i \leq k+1$, e $v_i v_{i+1} \in E(G)$ para $1 \leq i \leq k$. O *comprimento* de W é o número de arestas contidas nele, isto é, k. Uma *trilha* é um passeio que não repete arestas. Um *caminho* é um passeio que não repete vértices. Um *ciclo* é um passeio em que o vértice inicial é igual ao vértice final ($v_1 = v_{k+1}$), nenhum outro vértice é repetido e o comprimento é maior ou igual a 3.

Um grafo é dito *conexo* se existe um caminho entre qualquer par de vértices. Se um grafo é conexo e não possui ciclos, dizemos que esse grafo é uma *árvore*. Se um grafo tem aresta entre todos os pares de vértices, ele é dito *grafo completo*. Denota-se por K_n um grafo completo de ordem n.

Um *emparelhamento* em um grafo G é um conjunto $M \subseteq E(G)$ tal que nenhum par $e, f \in M$ tem extremos em comum. Dizemos que os vértices que são extremos das arestas em M estão *cobertos* pelo emparelhamento M. Um *emparelhamento perfeito* é um emparelhamento que cobre todos os vértices de um grafo.

Em vários problemas, particularmente nos que veremos, além de um grafo G na entrada também nos é dada uma função $w_G \colon E(G) \to \mathbb{Q}^+$ de pesos nas arestas e deseja-se encontrar algum subgrafo que satisfaça alguma dada propriedade e tal que a soma dos pesos das arestas desse subgrafo seja mínima. Quando for claro do contexto, podemos denotar a função w_G associada ao grafo G apenas por w.

Definimos como *árvore geradora* de um grafo G um subgrafo T que é uma árvore tal que $V(T) = V(G)$. Um T que é mínimo, ou seja, cuja soma dos pesos de suas arestas seja a menor possível, é chamado de *árvore geradora mínima*. No Problema da Árvore Geradora Mínima (MST, de *Minimum Spanning Tree*), dados um grafo G e uma função $w \colon E(G) \to \mathbb{Q}^+$, deseja-se encontrar um subgrafo $T \subseteq G$ que é uma árvore, é gerador e tenha custo mínimo na soma das arestas dentre todas as árvores geradoras de G.

O problema do emparelhamento máximo e o problema da árvore geradora mínima estão na classe P, ou seja, existem algoritmos eficientes que encontram solução ótima para tais.

Os resultados a seguir são fundamentais na teoria dos grafos.

Teorema 2.1 (Teorema do Aperto de Mãos)**.** *Para qualquer grafo G,*

$$\sum_{v \in V(G)} d_G(v) = 2|E(G)|\,.$$

Demonstração. Tome G como um grafo qualquer. Note que cada aresta contribui em duas unidades na soma de todos os graus dos vértices pois incide em exatamente dois

vértices. Então, quando realizamos o somatório $\sum_{v \in V(G)} d_G(v)$, estamos contando todas as arestas do grafo duas vezes, $2|E(G)|$. □

Corolário 2.1. *Para qualquer grafo, o número de vértices de grau ímpar é par.*

Demonstração. Do Teorema 2.1, podemos concluir que o somatório de todos os graus é sempre par. Além disso, podemos separar o somatório dos graus dos vértices ímpares dos pares, sendo $V_{imp}(G)$ o conjunto dos vértices de grau ímpar e $V_{par}(G)$ os de grau par:

$$\sum_{v \in V(G)} d_G(v) = \sum_{v \in V_{par}(G)} d_G(v) + \sum_{v \in V_{imp}(G)} d_G(v)\,. \tag{1}$$

Note que o lado esquerdo da Eq. (1) é par. Como $\sum_{v \in V_{par}(G)} d_G(v)$ também é par, pois a soma de graus pares também deve ser par, então $\sum_{v \in V_{imp}(G)} d_G(v)$ é igualmente par. Agora, observe que, como cada grau ímpar contribui com um número ímpar na soma, a soma dos graus ímpares é par se, e somente se, houver um número par de vértices com grau ímpar. Portanto, o número de vértices de grau ímpar é par. □

3 O Problema da Árvore de Steiner

Problema 3.1 (Árvore de Steiner)**.** *Dados um grafo G, uma função de custo $w\colon E(G) \to \mathbb{Q}^+$ e um conjunto $R \subseteq V(G)$, encontrar uma árvore $T \subseteq G$ tal que $R \subseteq V(T)$ e que minimize $w(T) = \sum_{e \in E(T)} w(e)$.*

Denotaremos tal problema por STP, do inglês *Steiner Tree Problem*. Os vértices de R são chamados de *terminais* ou *requeridos* enquanto que os vértices de $V(G) \setminus R$ são chamados de *vértices de Steiner*.

A seguir, mostraremos que o problema da Árvore de Steiner pode ser tratado, em termos de aproximação, se consideramos apenas uma variação dele. O problema da Árvore de Steiner Métrica é uma restrição do problema original em que, dado um grafo G, a função de peso $w\colon E(G) \to \mathbb{Q}^+$ satisfaz a propriedade da *desigualdade triangular*, isto é, para quaisquer $u, v, w \in V(G), w(uv) \leq w(uw) + w(wv)$. Denotaremos tal problema por MSTP, do inglês *Metric Steiner Tree Problem*.

Teorema 3.1. *Considere uma instância (G, w, R) para o problema da árvore de Steiner. Qualquer algoritmo de α-aproximação para o problema da Árvore de Steiner Métrico nos dá diretamente um algoritmo de α-aproximação para o problema da Árvore de Steiner.*

Demonstração. Vamos transformar, em tempo polinomial, uma instância $I = (G, w, R)$ de STP para uma instância I' de MSTP. Faça G' ser um grafo completo com $V(G') = V(G)$

em que, para cada $uv \in E(G')$, $w'(uv) = dist_G^w(u, v)$, sendo $dist_G^w(u, v)$ o caminho de custo mínimo entre os vértices u e v no grafo G. Tal grafo G' é chamado de *metric closure* de G, uma vez que a função de pesos w' definida dessa forma satisfaz a desigualdade triangular. Dessa forma, obtemos uma instância $I' = (G', w', R)$ para o MSTP.

Considere uma solução T' para a instância I', isto é, uma árvore em G' que contém os vértices de R. Vamos obter, em tempo polinomial, uma Árvore de Steiner T para I.

Substitua cada aresta presente em T' pelo caminho correspondente no grafo original G. Certamente os vértices terminais estão conectados. Porém, pela sua construção, é possível que sejam criados ciclos. Neste caso, remova as arestas necessárias. Como resultado, obtém-se uma árvore T.

Note que a transformação preserva o fator de aproximação pois a seguinte desigualdade é mantida:

$$w(T) \leq w'(T') \leq \alpha \mathrm{OPT}_{\mathsf{MSTP}}(I') \leq \alpha \mathrm{OPT}_{\mathsf{STP}}(I)\,.$$

A primeira parte da inequação vale porque na construção de T pode haver retirada de arestas, ou por formação de ciclo, ou porque estão repetidas. A segunda parte vale porque assumimos que T' é gerada por um algoritmo de α-aproximação para o problema da Árvore de Steiner Métrico. A última parte vale porque uma árvore de Steiner em I é uma árvore de Steiner em I', então, particularmente, uma árvore de Steiner ótima para I, que é uma solução viável para I', só pode ter custo maior do que uma árvore de Steiner ótima para I'. □

Por causa do resultado acima, podemos focar apenas no problema da Árvore de Steiner Métrica. A seguir, vamos apresentar um algoritmo de aproximação para o Problema da Árvore de Steiner Métrica, que está na classe de problemas NP-difíceis [6]. Para isso, usaremos o problema da Árvore Geradora Mínima.

Problema 3.2 (Árvore de Steiner Métrica)**.** *Dado um grafo G completo com $w\colon E(G) \to \mathbb{Q}^+$ que satisfaz a desigualdade triangular e o conjunto $R \subseteq V(G)$, nosso objetivo é encontrar uma árvore de custo mínimo que contenha todos os vértices de R.*

O que o algoritmo faz é construir uma árvore geradora mínima sobre $G[R], w$. Para isso, podemos usar diversos algoritmos conhecidos eficientes, como é o caso do Algoritmo de Kruskal ou o Algoritmo de Prim. Note que tal árvore é sempre uma solução viável para o problema da Árvore de Steiner, pois conecta os vértices terminais. Por isso,

$$\mathrm{OPT}_{\mathsf{MSTP}}(G, w, R) \leq \mathrm{OPT}_{\mathsf{MST}}(G[R], w)\,. \tag{2}$$

Ainda que o resultado não seja ótimo para o problema da Árvore de Steiner, uma vez que nem permite que a solução use vértices que não são terminais, veremos a seguir que ele

garante uma aproximação para o nosso problema.

No Algoritmo 1, apresentamos formalmente a ideia do algoritmo descrito acima.

Algoritmo 1 Algoritmo STEINER-MST, que recebe o grafo G completo, o subconjunto de vértices requeridos R, e uma função $w\colon E(G) \to \mathbb{Q}_+$ de peso das arestas que respeita a desigualdade triangular.

```
1: Função STEINER-MST(G, R, w)
2:     T ← ALGORITMO-MST(G[R], w)
3:     devolve T
```

Sejam $n = |V(G)|$ e $m = |E(G)|$. Note que o algoritmo leva tempo $O(m + n + m \log n) = O(m \log n)$, uma vez que cria o grafo $G[R]$ e chama um algoritmo como Kruskal ou Prim para encontrar uma árvore geradora mínima em tal grafo.

No teorema a seguir, provaremos que esse algoritmo garante um fator de aproximação para o problema da Árvore de Steiner Métrica.

Teorema 3.2. *O Algoritmo 1 é uma 2-aproximação para o problema da Árvore de Steiner Métrica.*

Demonstração. Seja $I = (G, w, R)$ uma instância para o problema e considere uma árvore de Steiner de valor ótimo T^*, isto é, $w(T^*) = \mathrm{OPT}_{\mathsf{MSTP}}(I)$.

Duplique todas as arestas de T^*. Assim, teremos um grafo T', que é (i) conexo, por ter sido construído a partir de uma árvore, e que (ii) para todo $v \in V(T')$, $d_{T'}(v)$ é par. Note que $w(T') = 2\,\mathrm{OPT}_{\mathsf{MSTP}}(I)$, pois o número de arestas foi dobrado.

Encontre uma trilha Euleriana $\mathcal{E}$ em T', isto é, uma trilha que contém todas as arestas de T'. Isso é possível ao satisfazermos as condições (i) e (ii). Tem-se, portanto, que

$$w(\mathcal{E}) = w(T') = 2\,\mathrm{OPT}_{\mathsf{MSTP}}(I)\,. \tag{3}$$

Obtenha um ciclo Hamiltoniano $\mathcal{H}$ a partir de $\mathcal{E}$ que contenha apenas os vértices terminais, ignorando os vértices de Steiner e os vértices terminais já visitados. Note que, por conta da propriedade métrica, tais "atalhos" não aumentam o valor do ciclo, isto é,

$$w(\mathcal{H}) \leq w(\mathcal{E}) = 2\,\mathrm{OPT}_{\mathsf{MSTP}}(I)\,. \tag{4}$$

Finalmente, delete uma aresta $e \in E(\mathcal{H})$ de $\mathcal{H}$ e obtenha uma árvore geradora em $G[R]$. Sendo T a árvore geradora mínima devolvida pelo algoritmo em $(G[R], w)$, vale que

$$w(T) \leq w(\mathcal{H})\,. \tag{5}$$

Sendo assim, como $w(\mathcal{H}) \leq 2\,\mathrm{OPT}_{\mathsf{MSTP}}(I)$ e o nosso algoritmo encontra uma árvore

geradora mínima em $G[R]$, vale que $\textsc{Steiner-MST}(I) \leq 2\,\mathrm{OPT}_{\mathsf{MSTP}}(I)$. □

A Figura 2 mostra um exemplo de devolução da função $\textsc{Steiner-MST}(G,\ R,\ w)$, tal-qualmente a solução para um grafo G representado. Note que a solução ótima para o problema em questão é quando as arestas que incidem no vértice de Steiner são escolhidas e temos custo ótimo 9, enquanto o custo das arestas em vermelho (escolhidas pelo algoritmo) tem custo total 10. Observe que ambas são viáveis pois conectam os vértices terminais.

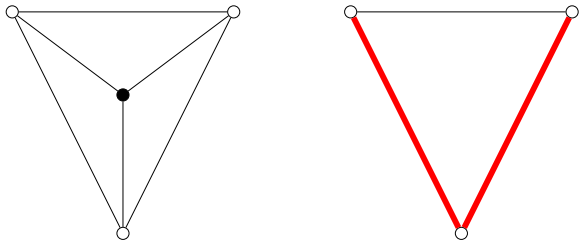

Figura 2: Nesses grafos, os vértices pintados são os vértices de Steiner, e os vértices em branco são os terminais. À esquerda, um grafo de entrada cujo qual as arestas que incidem no vértice de Steiner possuem custo 3 e as demais possuem custo 5. À direita, destacamos uma solução, em vermelho, devolvida pelo algoritmo (custo 8) após a transformação do grafo à esquerda induzido pelos terminais.

3.1 Algoritmo de aproximação de fator 11/6 para o STP

O algoritmo de aproximação e técnicas apresentadas nesta seção são resultados do estudo do artigo "*An 11/6-approximation algorithm for the network steiner problem*" [8], elaborado por Zelikovsky.

A abordagem utilizada anteriormente para o STP foi obter um grafo $G[R]$ a partir de um grafo G completo e que respeita a desigualdade triangular, sendo R o conjunto dos vértices requeridos e, então, aplicar um algoritmo de MST em $G[R]$. No entanto, mostraremos que é possível obter um fator melhor de aproximação se considerarmos a MST em $G[R \cup W]$ tal que W é um conjunto de vértices de Steiner escolhidos apropriadamente.

Uma *tripla* é a combinação de 3 vértices. Note que o grafo de entrada é completo, então toda tripla induz 3 arestas.

Na literatura [1], a *contração* de uma aresta $e = \{u, v\}$ em um grafo G qualquer é uma operação que cria um novo grafo denotado G/e, definido por $V(G/e) = V(G)\backslash\{u, v\}\cup\{x\}$, sendo x um novo vértice formado pela união de u e v, e $E(G/e) = \{f \in E(G)\colon u \notin f \text{ e } v \notin f\} \cup \{xy\colon uy \in E(G) \text{ ou } vy \in E(G)\}$. A Figura 3 ilustra a contração de uma aresta. Se o grafo G tem uma função $w_G\colon E(G) \to \mathbb{Q}_+$, então definimos $w_{G/e}$ de forma que $w_{G/e}(zx) = w_G(zu)$ para todo $z \in N_G(u)$ e $w_{G/e}(zx) = w_G(zv)$ para todo $z \in N_G(v)$.

Porém, a contração que utilizamos neste texto segue uma abordagem diferente, que é a mesma de Zelikovski [8]. Denotamos $G_{[e]}$ o grafo criado nessa nova definição de contração,

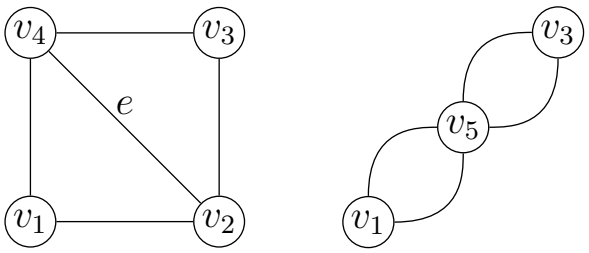

Figura 3: Representação gráfica, à esquerda, de um grafo G com $V(G) = \{v_1, v_2, v_3, v_4\}$ e uma aresta e, que será contraída. À direita, representação gráfica do grafo $H = G/e$ com $V(H) = \{v_1, v_5, v_3\}$ com a aresta $e \in E(G)$ contraída.

que apenas atribui peso 0 para a aresta e contraída. Dessa maneira, $V(G_{[e]}) = V(G)$, $E(G_{[e]}) = E(G)$, e $w_{G_{[e]}}$ é definido da seguinte forma: para cada aresta $f \in E(G)$, se $f = e$, então $w_{G_{[e]}}(f) = 0$; senão, $w_{G_{[e]}}(f) = w_G(f)$. Observe que nessa nova definição de contração de arestas, não é necessário de fato criar um novo grafo quando estamos pensando em implementação. Para não sobrecarregar a notação, denotaremos contrações consecutivas de duas arestas e e f em G como $G_{[e][f]}$ ao invés de $G_{[e]_{[f]}}$.

Também será bastante utilizada a contração de uma tripla z de vértices. Denotamos a contração da tripla z, formada pelos vértices u, v, w, como $G_{[z]}$. A contração de uma tripla é definida como a contração das três arestas que a compõem, isto é, $G_{[z]} = G_{[uv][vw][uw]}$. Para simplificar, usaremos a mesma notação de Zelikovski, e escreveremos apenas $G_{[uv][vw]}$. Note que como não deletamos e nem criamos nenhum vértice com essa nova definição de contração, as notações anteriores de fato fazem sentido.

A ideia do algoritmo de Zelikovski [8] é criar um subgrafo $F \subseteq G$ tal que $R \subseteq V(F)$ e devolver uma árvore geradora mínima de F. Inicialmente, $F \leftarrow G[R]$. A cada iteração, o algoritmo testa um novo vértice de Steiner para ser acrescentado a F de forma que diminua o custo da árvore que ele vai devolver. Os vértices escolhidos para minimizar o custo da árvore dada pelo algoritmo são atribuídos a um conjunto W. Para escolher esses vértices, o algoritmo cria um conjunto $\mathcal{Z}$ de todas as combinações de tamanho 3 de vértices terminais. Para cada tripla $z \in \mathcal{Z}$, encontra o vértice $v(z)$ que minimiza o valor $\sum_{s \in z} w_G(s, v(z))$. Tal custo é definido com o $w(z)$. Formalmente, a cada iteração, o algoritmo compara o custo de uma árvore geradora mínima em F, isto é, $\mathsf{MST}(F)$, com o custo de uma árvore geradora mínima em F com o vértice de Steiner $v(z)$ escolhido, isto é, $(\mathsf{MST}(F_{[z]} + w(z)))$. Essas ideias estão formalizadas no Algoritmo 2.

A seguir, discutiremos a análise do fator de aproximação do algoritmo. Denotaremos, para um conjunto Z de triplas, o valor $w(Z) = \sum_{z \in Z} w(z)$. Para um conjunto A compostos de arestas e triplas, definimos $F_{[A]}$ recursivamente da seguinte forma: $F_{[\emptyset]} = F$ e $F_{[A \cup e]} = F_{[A][e]}$, sendo e uma aresta qualquer. Para um grafo G e tripla z, definimos $\mathsf{win}_G(z) = \mathsf{MST}(G) - \mathsf{MST}(G_{[z]}) - w(z)$. Note que esse é o "ganho" que temos ao substituir arestas entre os vértices da tripla pelas três arestas que conectam o vértice $v(z)$ aos três

Algoritmo 2 Algoritmo STEINER-APROX, que recebe o grafo G completo, o subconjunto de vértices requeridos R, e uma função $w\colon E(G) \to \mathbb{Q}_+$ de peso das arestas que respeita a desigualdade triangular.

Função STEINER-APROX(G, R, w)
    $F \leftarrow G[R]$;
    $W \leftarrow \emptyset$;
    $\mathcal{Z} \leftarrow \{z \subset R\colon |z| = 3\}$
    Para cada $z \in \mathcal{Z}$ **faça**
        encontre $v \in V(G) \setminus R$ tal que $\sum_{s\in z} w_G(v, s)$ seja mínima
        $v(z) \leftarrow v$
        $w(z) \leftarrow \sum_{s\in z} w_G(v, s)$
    $z \leftarrow \emptyset$
    Faça
        $F \leftarrow F_{[z]}$
        $W = W \cup \{v(z)\}$
        encontre $z \in \mathcal{Z}$ que maximiza $\mathsf{win} = \mathsf{MST}(F) - \mathsf{MST}(F_{[z]}) - w(z)$
    enquanto $\mathsf{win} > 0$
    $T \leftarrow$ ALGORITMO_MST($G[R \cup W]$, $w_{G[R\cup W]}$)
    devolve T

da tripla.

Tome $(z_1, z_2, \ldots, z_n)$ como uma sequência de triplas. Dizemos que ela é uma *sequência gulosa em G* se satisfaz a seguinte propriedade:

- se $\mathsf{win}_G(z) \leq 0$ para todo $z \in \mathcal{Z}$, então $n = 0$;
- senão, $\mathsf{win}_G(z_1) \geq \mathsf{win}_G(z)$ para todo $z \in \mathcal{Z}$, e $(z_2, \ldots, z_n)$ é uma sequência gulosa em $G_{[z_1]}$.

Portanto, a sequência gerada pelo Algoritmo 2 é gulosa em G.

Zelikovsky [8] provou que a melhor redução de custos que pode ser feita sobre uma árvore geradora mínima em $G[R]$ aproxima $\mathrm{OPT}_{\mathsf{MSTP}}(I)$ por um fator de $\frac{5}{3}$. Mostra, também, que o Algoritmo 2 aproxima essa melhor redução possível por um fator de 2 (por não ser possível encontrar a melhor redução possível). O Teorema 3.3 mostra que o algoritmo guloso devolve esse resultado. Antes de vermos esse teorema e sua demonstração, precisamos de alguns resultados auxiliares. Vamos definir $\mathsf{save}_G(e) = \mathsf{MST}(G) - \mathsf{MST}(G_{[e]})$. O Lema 3.1 é um resultado para definição de *save*. O Lema 3.2 é um resultado necessário em uma das etapas de demonstração do Teorema 3.3.

Lema 3.1 (Zelikovsky [8])**.** *Seja $G_{\leq x}$ um grafo com os mesmos vértices que G e com todas as arestas de comprimento no máximo x. Então $\mathsf{save}_G(e)$ é o valor mínimo x tal que ambas as extremidades de e estão na mesma componente de $G_{\leq x}$.*

Demonstração. Precisaremos dos seguintes resultados auxiliares.

Afirmação 3.1. *Seja T uma MST de $(G_{[e]}, w_{G_{[e]}})$. Então $e \in E(T)$ ou $T' = T + e - f$ é*

MST de $(G_{[e]}, w_{G_{[e]}})$*, com* f *sendo a aresta de maior custo no ciclo em* $T + e$*.*

Demonstração. Se $e \in E(T)$, não há o que provar. Vamos para o caso em que $e \notin E(T)$ e seja f como no enunciado. Seja $T' = T + e - f$ e note que $w_{G_{[e]}}(T') = w_{G_{[e]}}(T) - w_{G_{[e]}}(f)$. Se T' não é MST de $(G_{[e]}, w_{G_{[e]}})$, então $w_{G_{[e]}}(T') > w_{G_{[e]}}(T)$. Sendo assim, necessariamente $w_{G_{[e]}}(f) < 0$ pela equação anterior, o que é um absurdo dadas as restrições do nosso problema. ♢

Afirmação 3.2. *Seja* T *uma MST de* (G, w_G) *e* $e \in E(T)$*, então* T *é MST de* $(G_{[e]}, w_{G_{[e]}})$*.*

Demonstração. Suponha que T não é MST de $(G_{[e]}, w_{G_{[e]}})$ e seja T^* uma MST de $(G_{[e]}, w_{G_{[e]}})$, então $w_{G_{[e]}}(T) > w_{G_{[e]}}(T^*)$. Pela Afirmação 3.1, vamos assumir que $e \in E(T^*)$, então $w_G(T^*) = w_{G_{[e]}}(T^*) + w_G(e)$. Da mesma forma, $w_G(T) = w_{G_{[e]}}(T) + w_G(e)$. Usando $w_{G_{[e]}}(T) > w_{G_{[e]}}(T^*)$, concluímos que $w_G(T) > w_G(T^*)$. Absurdo, pois T é MST de (G, w_G). Portanto, T é MST de $(G_{[e]}, w_{G_{[e]}})$. ♢

Afirmação 3.3. *Seja* T *uma MST de* (G, w_G) *e* $e \notin E(T)$*. Se* $e = uv$ *e* f *é a aresta de maior custo no ciclo existente em* $T + e$*, então* $T' = T + e - f$ *é MST de* $(G_{[e]}, w_{G_{[e]}})$*.*

Demonstração. Se T for MST em $(G_{[e]}, w_{G_{[e]}})$, então pela Afirmação 3.1 tem-se diretamente que T' também é MST em $(G_{[e]}, w_{G_{[e]}})$. Logo, T não é MST em $(G_{[e]}, w_{G_{[e]}})$ e seja T^* uma MST em $(G_{[e]}, w_{G_{[e]}})$, então $w_{G_{[e]}}(T^*) < w_{G_{[e]}}(T)$. Pela Afirmação 3.1, podemos assumir que $e \in E(T^*)$. Sejam T_u^* e T_v^* as duas componentes de $T^* - e$. Note que $(V(T_u^*), V(T_v^*))$ é um corte em G e que e pertence a esse corte com $u \in V(T_u^*)$ e $v \in V(T_v^*)$. Note ainda que deve existir ao menos mais uma aresta do ciclo existente em $T + e$ que também pertence a esse corte, digamos g. Não é possível afirmar que $g = f$, porém pela escolha de f sabemos que $w_G(f) \geq w_G(g)$. Note que $T^* + g - e$ é uma MST. Como T é mínima em (G, w_G), vale que $w_G(T^* + g - e) \geq w_G(T)$. Por outro lado,

$$\begin{aligned} w_{G_{[e]}}(T^* + g - e) &= w_{G_{[e]}}(T^*) + w_{G_{[e]}}(g) \\ &\leq w_{G_{[e]}}(T^*) + w_{G_{[e]}}(f) \\ &< w_{G_{[e]}}(T') + w_{G_{[e]}}(f) \\ &= w_{G_{[e]}}(T) \\ &= w_G(T), \end{aligned}$$

o que é uma contradição. ♢

Tome T como MST de (G, w_G). Vamos observar, primeiro, o caso em que e não está em T. Tome P como o único caminho existente em T que conecta as extremidades de e,

e f como a aresta de maior peso no ciclo existente em $T + e$. Note que esse ciclo é $P + e$, portanto f é a maior aresta em $P + e$.

Seja $T' = T + e - f$ e note que T' é uma árvore geradora mínima para $G_{[e]}$ pela Afirmação 3.3. Assim, por definição temos que $\mathsf{save}_G(e) = w_G(T) - w_{G_{[e]}}(T') = w_G(f) = w_{G_{[e]}}(f) = x$.

Vamos considerar $G_{\leq x}$ e note que o caminho P existe em $G_{\leq x}$. Suponha que exista $x' < x$ tal que $G_{\leq x'}$ tem caminho de u a v. Tome esse caminho como $P' \neq P$. Nesse sentido, deve haver $f' \in P'$ tal que $w_G(T - f + f') < w_G(T)$, mas T é MST em G. Absurdo.

Para o segundo caso, vamos considerar que e está em T. Note que pela Afirmação 3.2, T é MST em $G_{[e]}$.

Sendo assim, existe apenas uma aresta no uv-caminho em T, sendo a própria e. Portanto, é válido dizer que $\mathsf{save}_G(e) = \mathsf{MST}(G) - \mathsf{MST}(G_{[e]}) = x = w_G(e)$ e note que e é valor mínimo tal que as extremidades de e em $G_{\leq x}$ estão conectadas.

Nesse sentido, vamos considerar $G_{\leq x}$ e note que o caminho P existe em $G_{\leq x}$, sendo a própria aresta e. Suponha que exista $x' < x$ tal que $G_{\leq x'}$ tem caminho de u a v. Tome esse caminho como $P' \neq P$. Nesse sentido, deve haver $f' \in P'$ tal que $w_G(T - e + f') < w_G(T)$, mas T é MST. Absurdo. □

Lema 3.2 (Zelikovsky [8])**.** *Tome Z como um conjunto qualquer de triplas. Então, para cada aresta e, ou*

- $\mathit{win}_{G_{[e]}}(Z) = \mathit{win}_G(Z)$*, ou*
- *existe $z \in Z$ tal que $\mathit{win}_{F_{[e]}}(Z - z) \geq \mathit{win}_{F_{[z]}}(Z - z)$.*

Teorema 3.3 (Zelikovsky [8])**.** *Se H é uma sequência gulosa, então para toda sequência de triplas Z vale que $2\mathit{win}_G(H) \geq \mathit{win}_G(Z)$.*

Demonstração. Vamos provar este resultado por indução no tamanho de H.

Note que quando $H = \emptyset$, então $\mathsf{win}_G(z) \leq 0$ para todo $z \in \mathcal{Z}$.Então claramente vale $2\mathsf{win}_G(H) = 0 \geq \mathsf{win}_G(Z) = 0$.

Agora seja $H \neq \emptyset$ e seja h a primeira tripla que aparece em H tal que $h = \{a \cup b\}$, onde a e b são duas arestas. Suponha que vale a hipótese indutiva, isto é, $2\mathsf{win}_{G_{[h]}}(H - h) \geq \mathsf{win}_G(Z)$.

Pelo Lema 3.2, vale para a aresta a que

(a.i) $\mathsf{win}_{G_{[a]}}(Z) = \mathsf{win}_G(Z)$, ou
(a.ii) $\exists z_a \in Z$ t.q. $\mathsf{win}_{G_{[a]}}(Z - z_a) \geq \mathsf{win}_{G_{[z_a]}}(Z - z_a)$.

E, para a aresta b, vale que

(b.i) $\mathsf{win}_{G_{[b]}}(Z) = \mathsf{win}_G(Z)$, ou

(b.ii) $\exists z_b \in Z$ t.q. $\mathsf{win}_{G_{[b]}}(Z - z_b) \geq \mathsf{win}_{G_{[z_b]}}(Z - z_b)$.

Note também que

$$\begin{aligned}
\mathsf{win}_{G_{[h]}}(Z) &= \mathsf{win}_{G_{[a][b]}}(Z) && (6)\\
&= \mathsf{MST}(G_{[a][b]}) - \mathsf{MST}(G_{[Z][a][b]}) - w(Z) && (7)\\
&= \mathsf{MST}(G_{[a]}) - \mathsf{save}_{G_{[a]}}(b) - \mathsf{MST}(G_{[Z][a][b]}) - w(Z) && (8)\\
&\geq \mathsf{MST}(G_{[a]}) - \mathsf{save}_G(b) - \mathsf{MST}(G_{[Z][a][b]}) - w(Z) && (9)\\
&= \mathsf{MST}(G_{[a]}) - \mathsf{MST}(G) + \mathsf{MST}(G_{[b]}) - \mathsf{MST}(G_{[Z][a][b]}) - w(Z) && (10)\\
&\geq \mathsf{MST}(G_{[a]}) - \mathsf{MST}(G) + \mathsf{MST}(G_{[b]}) - \mathsf{MST}(G_{[Z][a]})\\
&\quad - \mathsf{MST}(G_{[Z][b]}) + \mathsf{MST}(G_{[Z]}) - w(Z) && (11)\\
&= \mathsf{win}_{G_{[a]}}(Z) + \mathsf{win}_{G_{[b]}}(Z) - \mathsf{win}_G(Z)\,, && (12)
\end{aligned}$$

onde (7) vale pela definição de win; (8) vale pela definição de save; (9) vale pois $\mathsf{save}_G(b) \geq \mathsf{save}_{G_{[a]}}(b)$, já que $G_{[a]}$ possui contração; (11) vale pela construção de $\mathsf{MST}(G_{[Z][a][b]})$.

Precisaremos do seguinte resultado como auxiliar em um dos casos a seguir.

Afirmação 3.4. *$\mathsf{win}_G(A \cup B) = \mathsf{win}_G(A) + \mathsf{win}_{G_{[A]}}(B)$.*

Demonstração. Por definição, $\mathsf{win}_G(A \cup B) = \mathsf{MST}(G) - \mathsf{MST}(G_{[A][B]}) - w(A \cup B)$. Segue que

$$\begin{aligned}
\mathsf{win}_G(A \cup B) &= \mathsf{MST}(G) - \mathsf{MST}(G_{[A][B]}) - w(A \cup B) + \mathsf{MST}(G_{[A]}) - \mathsf{MST}(G_{[A]}) && (13)\\
&= \mathsf{MST}(G) - \mathsf{MST}(G_{[A]}) - w(A) + \mathsf{MST}(G_{[A]}) - \mathsf{MST}(G_{[A][B]}) - w(B) && (14)\\
&= \mathsf{win}_G(A) + \mathsf{win}_{G_{[A]}}(B)\,, && (15)
\end{aligned}$$

onde (14) é apenas uma reescrita de (13). ♢

Para continuar a desenvolver a Eq. (12), seja $\star = \mathsf{win}_{G_{[a]}}(Z) + \mathsf{win}_{G_{[b]}}(Z) - \mathsf{win}_G(Z)$. Vamos considerar a combinação de casos de acordo com o Lema 3.2.

Se (a.i) e (b.i) valem, então

$$\star \geq \mathsf{win}_G(Z) \geq \mathsf{win}_G(Z) - 2\mathsf{win}_G(Z)\,. \tag{16}$$

Se (a.i) e (b.ii) valem, então

$$\star = \mathsf{win}_{G_{[b]}}(Z) \geq \mathsf{win}_{G_{[b]}}(Z - z_b) \geq \mathsf{win}_{G_{[z_b]}}(Z - z_b) = \mathsf{win}_G(Z) - \mathsf{win}_G(z_b) \geq \mathsf{win}_G(Z) - 2\mathsf{win}_G(h)\,. \tag{17}$$

Se (a.ii) e (b.i) valem, então

$$\begin{aligned}\star &= \mathsf{win}_{G_{[a]}}(Z) \geq \mathsf{win}_{G_{[a]}}(Z - z_a) \geq \mathsf{win}_{G_{[z_a]}}(Z - z_a) \\ &= \mathsf{win}_G(Z) - \mathsf{win}_G(z_a) \geq \mathsf{win}_G(Z) - 2\mathsf{win}_G(h)\,. \end{aligned} \tag{18}$$

Se (a.ii) e (b.ii) valem, então

$$\begin{aligned}\star &\geq \mathsf{win}_{G_{[z_a]}}(Z - z_a) + \mathsf{win}_{G_{[z_b]}}(Z - z_b) - \mathsf{win}_G(Z) \\ &= 2\mathsf{win}_G(Z) - \mathsf{win}_G(z_a) - \mathsf{win}_G(z_b) \geq \mathsf{win}_G(Z) - 2\mathsf{win}_G(h)\,. \end{aligned} \tag{19}$$

Com isso, conclui-se que

$$\mathsf{win}_{G_{[h]}}(Z) \geq \mathsf{win}_G(Z) - 2\mathsf{win}_G(h)\,. \tag{20}$$

Note que a igualdade nas Eqs. (17) e (18) valem pela Afirmação 3.4.

Por fim, pela Equação (20) e hipótese indutiva, tem-se que

$$2\mathsf{win}_G(H) = 2\mathsf{win}_{G_{[h]}}(H - h) + 2\mathsf{win}_G(h) \tag{21}$$

$$\geq \mathsf{win}_{G_{[h]}}(Z) + 2\mathsf{win}_G(h) \tag{22}$$

$$\geq \mathsf{win}_G(Z)\,. \tag{23}$$

□

No lema a seguir, temos como resultado que a melhor redução de custos que pode ser feita sobre uma árvore geradora mínima em $G[R]$ aproxima $\mathrm{OPT}_{\mathsf{MSTP}}(I)$ por um fator de $\frac{5}{3}$. Uma *árvore binária perfeita* é uma árvore binária na qual todos os nós internos possuem exatamente dois filhos, e todas as folhas estão no mesmo nível.

Lema 3.3. *Existe um conjunto de triplas Z tal que*

$$3\left(\mathit{MST}(G[R] - \mathit{win}(Z))\right) \leq 5\, \mathit{OPT}_{\mathit{MSTP}}(I)\,.$$

Demonstração. Dada a instância (G, w_G, R) do problema da Árvore de Steiner métrica, fazemos uma transformação na instância para conseguir uma solução ótima que seja uma árvore binária perfeita da seguinte forma. Replicamos alguns vértices de forma que a distância dos vértices replicados para os originais é zero e algumas cópias podem também ser terminais. Vamos continuar chamando o novo grafo modificado de G e sua função de custos de w_G, para não sobrecarregar notações. Assim sendo, garantimos que uma árvore de Steiner mínima, T_{min}, possui a forma de uma árvore binária perfeita, sendo o conjunto de folhas S dessa árvore tal que $S \subseteq R$.

Seja r a profundidade de T_{min} e seja $B = \{0, 1\}$. Definimos B^* como o conjunto de todas as cadeias finitas formadas pelos elementos de B. Assim, rotulamos os vértices de T_{min} com palavras de B^*. Para cada vértice x, seus filhos são rotulados por $x0$ e $x1$, e $|x| \leq r$, sendo λ (palavra vazia) a raiz. Denotamos por L o conjunto de rótulos dos vértices internos de T_{min}. Para cada vértice x de T_{min}, definimos recursivamente ρ_x como 0 se x é uma folha, senão $\rho_x = w_G(x, x0) + \rho_{x0}$. A construção da nossa árvore se dá de forma que $\rho_x \leq w_G(x, x1) + \rho_{x1}$, o que garante que ρ_x é o custo do menor caminho do vértice x para uma folha da sua subárvore, que chamaremos de s_x. Denotamos ainda por h_x o valor do somatório dos custos das duas arestas que incidem nos seus filhos, isto é, $h_x = w_G(x, x0) + w_G(x, x1)$.

Para o conteúdo a seguir, usaremos b tal que $b \in B$, isto é, b é um dígito binário. Note que $\bar{b}$ é o elemento que representa o complemento de b. Dessa forma, se $b = 0$, $\bar{b} = 1$.

Para todo vértice interno $x \in L$, definimos a aresta $e_x = \{s_{x0}, s_{x1}\}$. Note que a árvore $T_F = \{e_x : x \in L\}$ é uma árvore geradora de $G[R]$. As arestas e_{xb} e e_x estão na árvore T_F e juntas formam a tripla $\{s_{xb0}, s_{xb1}, s_{x\bar{b}}\}$ de vértices. Essas arestas serão substituídas por arestas auxiliares que conectam os vértices da tripla por meio de xb se isso melhorar o custo dessa árvore. Formalmente, vamos mostrar que podemos melhorar tal árvore usando triplas da forma

$$z_{xb} = e_{xb} \cup e_x = \{s_{xb0}, s_{xb1}, s_{x\bar{b}}\}$$

.

Sejam e_1, e_2, e_3 essas arestas que conectam xb aos vértices da tripla, isto é, e_1 é a aresta $\{xb, s_{xb1}\}$, e_2 é a aresta $\{xb, s_{xb0}\}$ e e_3 é a aresta $\{xb, s_{x\bar{b}}\}$. Uma representação gráfica dessa discussão pode ser visualizada na Figura 4.

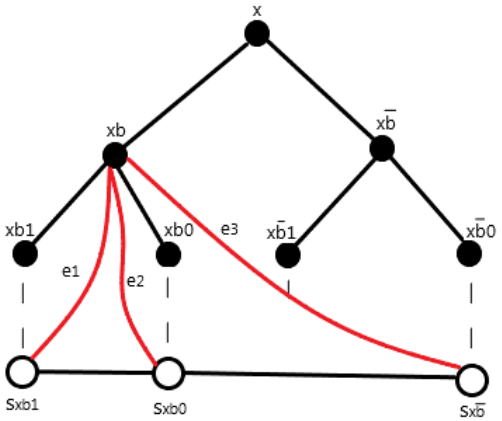

Figura 4: Representação gráfica da subárvore enraizada em x com as arestas auxiliares e_1, e_2 e e_3 e folhas s_{xb1}, s_{xb0} e $s_{x\bar{b}}$.

Note que $w_G(e_1) + w_G(e_2) \leq h_{xb} + \rho_{xb0} + \rho_{xb1}$ e $w_G(e_3) \leq h_x + \rho_{x\bar{b}}$. Destas desigualdades, temos que $w_G(e_1) + w_G(e_2) + w_G(e_3) \leq h_{xb} + \rho_{xb0} + \rho_{xb1} + h_x + \rho_{x\bar{b}}$ e definiremos $\mathsf{save}_{xb} =$

$(w_G(e_x) + w_G(e_{xb})) - (w_G(e_1) + w_G(e_2) + w_G(e_3))$. Portanto,

$$w_G(e_x) + w_G(e_{xb}) - \mathsf{save}_{xb} \leq h_{xb} + \rho_{xb0} + \rho_{xb1} + h_x + \rho_{x\bar{b}}\,. \tag{24}$$

Queremos agora avaliar o somatório da expressão em (24) para cada vértice $xb \in L \setminus \{\lambda\}$, isto é:

$$\sum_{xb \in L\setminus\{\lambda\}} (w_G(e_x) + w_G(e_{xb}) - \mathsf{save}_{xb}) \leq \sum_{xb \in L\setminus\{\lambda\}} (h_{xb} + \rho_{xb0} + \rho_{xb1} + h_x + \rho_{x\bar{b}})\,. \tag{25}$$

Avaliaremos a expressão (24) para alguns vértices de $L\setminus\{\lambda\} = \{0, 1, 00, 01, 10, 11, 000, \dots\}$, sendo 0 e 1 os filhos de λ:

$$\begin{aligned}
wb = 0: &\quad w_G(e_0) + w_G(e_\lambda) - \mathsf{save}_0 \leq h_0 + h_\lambda + \rho_{00} + \rho_{01} + \rho_1\,;\\
wb = 1: &\quad w_G(e_1) + w_G(e_\lambda) - \mathsf{save}_1 \leq h_1 + h_\lambda + \rho_{10} + \rho_{11} + \rho_0\,;\\
wb = 00: &\quad w_G(e_{00}) + w_G(e_0) - \mathsf{save}_{00} \leq h_{00} + h_0 + \rho_{000} + \rho_{001} + \rho_{01}\,;\\
wb = 01: &\quad w_G(e_{01}) + w_G(e_0) - \mathsf{save}_{01} \leq h_{01} + h_0 + \rho_{010} + \rho_{011} + \rho_{00}\,;\\
wb = 10: &\quad w_G(e_{10}) + w_G(e_1) - \mathsf{save}_{10} \leq h_{10} + h_1 + \rho_{100} + \rho_{101} + \rho_{11}\,;\\
wb = 11: &\quad w_G(e_{11}) + w_G(e_1) - \mathsf{save}_{11} \leq h_{11} + h_1 + \rho_{110} + \rho_{111} + \rho_{10}\,;\\
wb = 000: &\quad w_G(e_{000}) + w_G(e_{00}) - \mathsf{save}_{000} \leq h_{000} + h_{00} + \rho_{0000} + \rho_{0001} + \rho_{001}\,.
\end{aligned}$$

Note que, se $x \neq \lambda$, d_x aparece em 3 expressões e d_λ aparece em duas, então

$$\sum_{xb \in L\setminus\{\lambda\}} (w_G(e_x) + w_G(e_{xb})) = \left(\sum_{x\in L} 3w_G(e_x)\right) - w_G(e_\lambda)\,.$$

O mesmo acontece para h_x, de forma que

$$\sum_{xb \in L\setminus\{\lambda\}} (h_{xb} + h_x) = \left(\sum_{x\in L} 3h_x\right) - h_\lambda\,.$$

Note que cada ρ_x aparece em 2 expressões e ρ_λ não aparece nas expressões, então

$$\sum_{x \in L\setminus\{\lambda\}} (\rho_{xb0} + \rho_{xb1} + \rho_{x\bar{b}}) = \left(\sum_{x\in L} 2\rho_x\right) - 2\rho_\lambda\,.$$

Agora, utilizaremos os resultados anteriores em (25)). No lado esquerdo da inequação,

temos que

$$\sum_{x\in L\setminus\{\lambda\}} (w_G(e_{xb}) + w_G(e_x) - \mathsf{save}_{xb}) = 3\sum_{x\in L} w_G(e_x) - w_G(e_\lambda) - \sum_{xb\in L\setminus\{\lambda\}} \mathsf{save}_{xb}\,.$$

E, para o lado direito,

$$\sum_{xb\in L\setminus\{\lambda\}} (h_{xb} + h_x + \rho_{xb0} + \rho_{xb1} + \rho_{x\bar{b}}) = 3\sum_{x\in L} h_x - h_\lambda + 2\sum_{x\in L} \rho_x - 2\rho_\lambda\,.$$

Finalmente, obtemos

$$3\sum_{x\in L} w_G(e_x) - w_G(e_\lambda) - \sum_{xb\in L\setminus\{\lambda\}} \mathsf{save}_{xb} \leq 3\sum_{x\in L} h_x - h_\lambda + 2\sum_{x\in L} \rho_x - 2\rho_\lambda\,. \tag{26}$$

Observe, também, que, por definição,

$$\sum_{x\in L} h_x = \mathrm{OPT}_{\mathsf{MSTP}}(I)\,, \tag{27}$$

assim como

$$\sum_{x\in L} w_G(e_x) = w_G(T_S)\,. \tag{28}$$

Tome Π_u como o custo do caminho mais longo até uma folha na subárvore a partir de u. Note que, portanto, $\rho_u \leq \Pi_u$. Vamos, agora, provar por indução na altura da árvore enraizada em u que

$$\sum_{x\in L\cap uB^*} \rho_x \leq \sum_{x\in L\cap uB^*} h_x - \Pi_u\,, \tag{29}$$

onde uB^* é um abuso de notação que representa o conjunto $\{x \in B^* \colon x \text{ começa com } u\}$. Assim, $L \cap uB^*$ é o conjunto de vértices não-folhas contidos na subárvore enraizada em u.

Note que quando a altura é zero, u é folha, então $L \cap uB^*$ só contém u. Portanto, por definição, $\rho_u = 0$, $h_u = 0$ e $\Pi_u = 0$, logo $\rho_u \leq h_u - \Pi_u$.

Seja, agora, u um vértice em altura maior que 0 tal que a propriedade valha para vértices com altura menor do que a de u. Note que $h_u = w_G(\{u, u0\}) + w_G(\{u, u1\})$, $\rho_u = \rho_{u0} + w_G(\{u, u0\})$ e $\Pi_u = w_G(\{u, u1\}) + \Pi_{u1}$. Note também que $\sum_{x\in L\cap uB^*} \rho_x = \rho_u + \sum_{x\in L\cap u1B^*} \rho_x + \sum_{x\in L\cap u0B^*} \rho_x$, e que $\sum_{x\in L\cap B^*} h_x = h_u + \sum_{x\in L\cap u1B^*} h_x + \sum_{x\in L\cap u0B^*} h_x$.

Utilizando a hipótese indutiva nos filhos de u e as igualdades anteriores, temos:

$$\sum_{x\in L\cap uB^*}\rho_x = \rho_u + \sum_{x\in L\cap u1B^*}\rho_x + \sum_{x\in L\cap u0B^*}\rho_x \tag{30}$$

$$\leq \rho_{u0} + w_G(\{u,u0\}) + \sum_{x\in L\cap u1B^*} h_x - \Pi_{u1} + \sum_{x\in L\cap u0B^*} h_x - \Pi_{u0} \tag{31}$$

$$= \rho_{u0} + w_G(\{u,u0\}) + \sum_{x\in L\cap u1B^*} h_x + w_G(\{u,u1\}) - \Pi_u + \sum_{x\in L\cap u0B^*} h_x - \Pi_{u0} \tag{32}$$

$$= \rho_{u0} + h_u + \sum_{x\in L\cap u1B^*} h_x + \sum_{x\in L\cap u0B^*} h_x - \Pi_u - \Pi_{u0} \tag{33}$$

$$= \sum_{x\in L\cap uB^*} h_x - \Pi_u + \rho_{u0} - \Pi_{u0} \tag{34}$$

$$\leq \sum_{x\in L\cap uB^*} h_x - \Pi_u\,, \tag{35}$$

onde (35) vale pois, por definição, $\Pi_{u0} \geq \rho_{u0}$.

De (27) e (29), concluímos que

$$\sum_{x\in L}\rho_x \leq \mathrm{OPT}_{\mathsf{MSTP}}(I) - \Pi_\lambda\,. \tag{36}$$

Além disso, temos que

$$d_\lambda \leq 2\Pi_\lambda\,, \tag{37}$$

que vale pela desigualdade triangular.

Finalmente, de (26), (27), (28), (36) e (37), obtemos

$$3w_G(T_S) - \sum_{xb\in L\setminus\{\lambda\}} \mathsf{save}_{xb} \leq 5\mathrm{OPT}_{\mathsf{MSTP}}(I)\,. \tag{38}$$

Vimos que podemos melhorar o custo da árvore T_F com adição de triplas da forma z_{xb}. Observe que temos uma redução no valor de save_{xb} em T_F nesse caso. Note que se realizadas as substituições reveladas anteriormente para cada tripla z_{xb}, não é garantido que as trocas não afetem uma mesma aresta da árvore. Assim, vamos dizer que A é um conjunto *legal* se, para todo par $x, y \in A$, z_x e z_y não afetam uma mesma aresta. Dessa forma, garantimos que, ao final, o custo da nossa árvore com as reduções será $w(T_F) - \sum_{x\in A}\mathsf{save}_x$. Note, por fim, que $A \subseteq L \setminus \{\lambda\}$.

Agora note que se observarmos o grafo auxiliar formado pelos vértices de $L\setminus\{\lambda\}$ e as arestas entre x, xb e $x\bar{b}$ para todo $x \in L\setminus\{\lambda\}$, podemos fazer a observação de que um conjunto A é legal se e somente se ele é um conjunto independente neste grafo auxiliar. A intuição por trás disso acontece porque qualquer aresta nesse grafo auxiliar contém dois vértices x, y tais que z_x e z_y poderiam afetar uma mesma aresta.

Um grafo é *3-colorível* se seus vértices podem ser particionados em três conjuntos independentes A_1, A_2, A_3. O grafo auxiliar mencionado no parágrafo anterior é 3-colorível, isto é, cada A_i é um conjunto legal de substituições. Consequentemente, para cada conjunto legal A_i definimos $Z_i = \{z_x \colon x \in A_i\}$, garantindo que todas as arestas em T_S possam ser substituídas de forma independente dentro de cada conjunto, preservando o ganho $\sum_{xb \in A_i} \mathsf{save}_{xb}$. Nesse sentido, podemos afirmar que

$$\sum_{i=1}^{3} (\mathsf{MST}(G[R]) - \mathsf{win}(Z_i)) \leq 3w_G(T_S) - \sum_{xb \in L \setminus \{\lambda\}} \mathsf{save}_{xb}\,. \tag{39}$$

E, então, de (38) e (39), concluímos que deve haver um conjunto Z_i para $i \in \{1, 2, 3\}$ que satisfaça o lema, já que $3 \min\{\mathsf{MST}(G[R]) - \mathsf{win}(Z_1), \mathsf{MST}(G[R]) - \mathsf{win}(Z_2), \mathsf{MST}(G[R]) - \mathsf{win}(Z_3)\} \leq \sum_{i=1}^{3} (\mathsf{MST}(G[R]) - \mathsf{win}(Z_i))$. □

Finalmente, temos todas as ferramentas para mostrar que o Algoritmo 1 é uma 11/6-aproximação para o problema da árvore de Steiner. Lembre-se que o algoritmo devolve uma árvore com custo $\mathsf{MST}(G[R \cup W])$ e note que H é uma sequência gulosa de triplas em G:

$$\begin{aligned}
6\mathsf{MST}(G[R \cup W]) &\leq 6(\mathsf{MST}(G[R]) - w(H)) && (40)\\
&= 3\mathsf{MST}(G[R]) + 3[\mathsf{MST}(G[R]) - 2w(H)] && (41)\\
\textit{Teorema (3.3)} \quad &\leq 3\mathsf{MST}(G[R]) + 3[\mathsf{MST}(G[R]) - w(Z)] && (42)\\
\textit{Lema (3.3)} \quad &\leq 3\mathsf{MST}(G[R]) + 5(\mathrm{OPT}_{MTSP}(I)) && (43)\\
\textit{Teorema (3.2)} \quad &\leq 6(\mathrm{OPT}_{MTSP}(I)) + 5(\mathrm{OPT}_{MTSP}(I)) = 11(\mathrm{OPT}_{MTSP}(I)). && (44)
\end{aligned}$$

Observe que (40) vale pois pode haver intersecção de arestas entre as triplas pertencentes a W, então o ganho com as triplas que possuem alguma intersecção não será integral. Em H, esta intersecção é desconsiderada.

3.2 Implementação da 11/6-aproximação para o Problema da Árvore de Steiner

O Código 1 é uma implementação do Algoritmo 2. É possível visitá-lo no GitHub [1]. Neste, foi utilizada a linguagem Python por conta de suas bibliotecas, que possuem diversos recursos para trabalhar com grafos. Em específico, foi utilizada a biblioteca SageMath[2], que possui diversas funções implementadas.

[1] https://github.com/diastoff1/steiner-approximation-algorithm
[2] https://doc.sagemath.org/html/en/reference/graphs/sage/graphs/generic_graph.html

A Figura 5 representa as saídas do Código 1 para a implementação do Algoritmo 2 sobre um grafo específico. As primeiras três figuras são as contrações, em ordem, das triplas escolhidas pelo algoritmo. A última figura representa a saída final, sendo a árvore devolvida com os vértices auxiliares que promoveram ganho positivo e, consequentemente, uma redução no custo do MST em cima de $G[R]$.

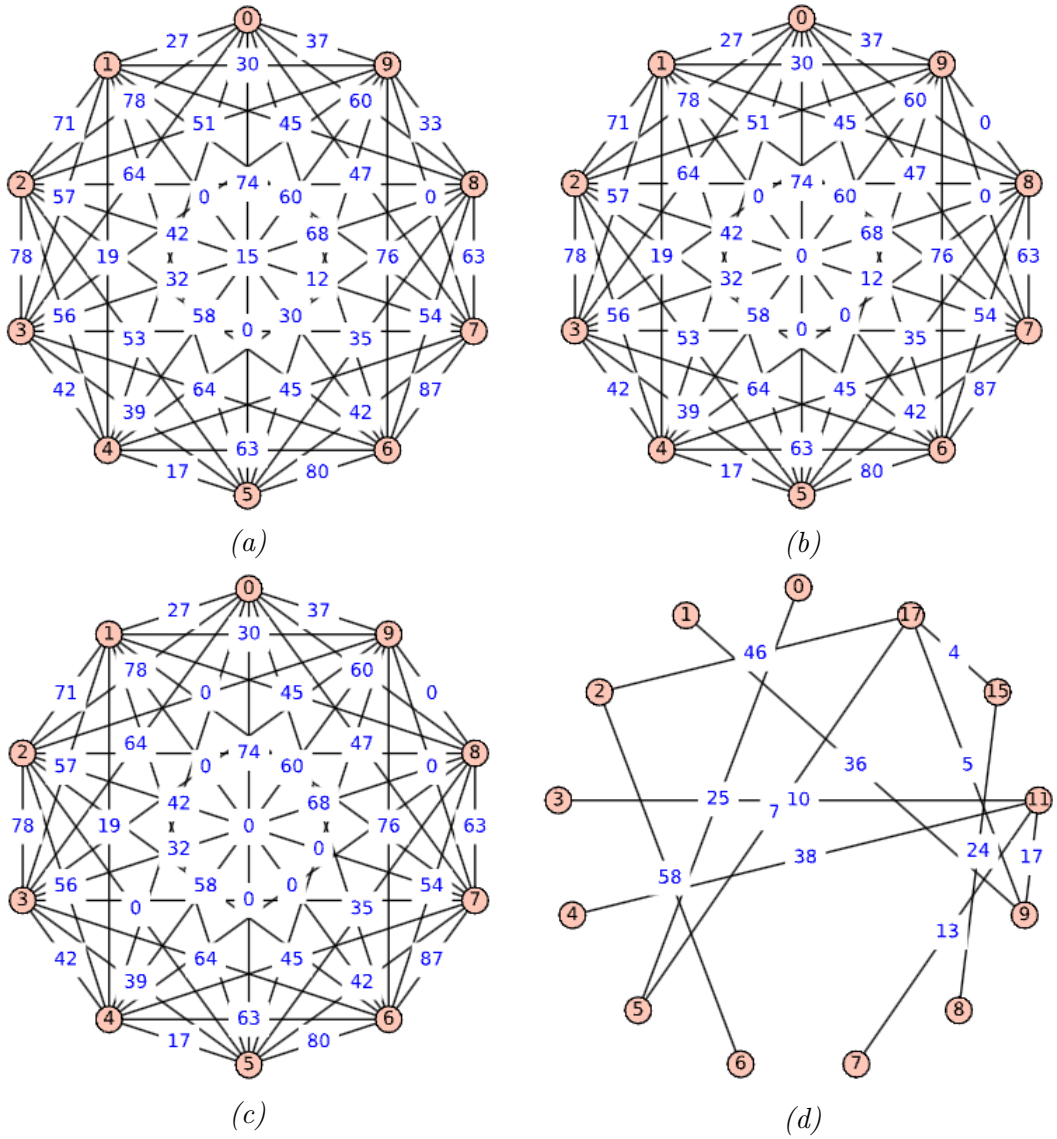

Figura 5: Representação gráfica das saídas do Código 1. Das figuras (a) a (c) acontecem as contrações no grafo induzido pelos vértices terminais. Na figura (d) está a saída final, uma árvore, com os vértices de Steiner auxiliares escolhidos pelo algoritmo, sendo estes os vértices 11, 15 e 17.

Código 1: Implementação do algoritmo de 11/6-aproximação para o STP.

```python
import random
from itertools import combinations

def mst(G):
    return G.min_spanning_tree(by_weight=True)

# metric closure de um grafo G, usa uma função que cria um grafo completo e depois
↪  adiciona peso às arestas baseado na distância mínima
```

```
8  def metric_closure(G):
9      MC = graphs.CompleteGraph(G.order())
10     for u, v in MC.edges(labels=False):
11         MC.set_edge_label(u, v, G.distance(u, v, by_weight=True))
12     return MC
13
14 # calcula o peso total das arestas de um vetor de arestas
15 def weight(edges):
16     return sum(weight for (u, v, weight) in edges)
17
18 # contrai as 3 arestas de uma dada tripla
19 def contract_triple(G, triple):
20     # adiciona todas as arestas da tripla ao vetor edges
21     edges = [(triple[i], triple[j], G.edge_label(triple[i], triple[j])) for i in
       ↪  range(2) for j in range(i+1, 3)]
22     # define o peso de todas as três arestas como 0
23     for u, v, _ in edges:
24         G.set_edge_label(u, v, 0)
25     return G
26
27 # calcula a soma das arestas entre um vértice específico v e um conjunto de vértices
28 def total_distance(G, v, vertices):
29     return sum(G.distance(v, s, by_weight=True) for s in vertices)
30
31 def steiner_tree_11-6approximation(G, n, t):
32     # divide os vértices: os t primeiros são terminais, os restantes de steiner
33     vertices = list(range(n))
34     terminal_vertices = vertices[:t]
35     steiner_vertices = vertices[t:]
36
37     G_mc = metric_closure(G)
38     G_ind = G_mc.subgraph(vertices=terminal_vertices)
39
40     G.plot(layout="circular", edge_labels=True).show()
41     G_mc.plot(layout="circular", edge_labels=True).show()
42     G_ind.plot(layout="circular", edge_labels=True).show()
43
44     W = set()
45     F = G_ind.copy()
46     Triples = list(combinations(terminal_vertices, 3))
47     d = {}
48     v = {}
49     # para cada tripla, encontra o vértice de Steiner que minimiza o somatório das
       ↪  distâncias, e então adiciona ao dicionário
50     for z in Triples:
51         v[z] = min(steiner_vertices, key=lambda vert: total_distance(G, vert, z))
52         d[z] = total_distance(G, v[z], z)
53
54     while True:
55         # encontra a tripla que maximiza 'win'
56         z = max(Triples, key=lambda z: weight(mst(F)) -
           ↪  weight(mst(contract_triple(F.copy(), z))) - d[z])
57         win = weight(mst(F)) - weight(mst(contract_triple(F.copy(), z))) - d[z]
58
59         if win <= 0:
60             break
61
62         F = contract_triple(F, z)
63         W.add(v[z])
64
65     induced_subgraph = G.subgraph(vertices=list(W) + terminal_vertices)
66     mst_edges = mst(induced_subgraph)
67     mst_graph = Graph(mst_edges, weighted=True)
```

A função `metric_closure` cria um grafo completo onde as arestas são ponderadas pelas menores distâncias do grafo original. Em seguida, o algoritmo itera sobre triplas de

vértices terminais, selecionando vértices de Steiner que maximizam o ganho, ou seja, que maximizam win. As arestas dessas triplas são contraídas e o processo é repetido até que não haja mais ganho na contração de triplas. Observe que podemos acompanhar o peso do grafo pela função `graph_weight`. Ademais, foi utilizado um dicionário para salvar $v(z)$ e $d(z)$ que aparecem no Algoritmo 2. Por fim, é gerada e visualizada uma MST para o subgrafo induzido pelos vértices de Steiner selecionados com os vértices terminais ($\mathsf{MST}(G[R \cup W])$).

4 Conclusão

O principal resultado desse projeto está presente na Seção 3 e foi o detalhamento do problema da Árvore de Steiner e algoritmos de aproximação para tal. Em primeiro plano, foi feita uma análise para a 2-aproximação desse problema e suas técnicas. As mesmas etapas foram realizadas para a 11/6-aproximação desse mesmo problema em uma análise do artigo de Zelikovsky [8], que trata sobre uma aproximação com melhor fator de aproximação em relação a primeira.

Ao comparar a 2-aproximação e a 11/6-aproximação, o primeiro algoritmo se baseia em técnicas clássicas e, embora sejam mais fáceis e simples de implementar, sacrificam possíveis ganhos ao não considerar a inclusão dos nós de Steiner. Por outro lado, a 11/6-aproximação de Zelikovsky [8] é mais sofisticada e complexa ao considerar mais os vértices de Steiner e possíveis ganhos em suas adições. Método, este, que reduz a lacuna entre a abordagem da 2-aproximação, oferecendo maior precisão em troca de uma maior complexidade computacional. Note que, ainda assim, a aproximação de fator 11/6 mantém-se como um algoritmo eficiente, o que o torna atraente para uma melhor minimização de custos, sem comprometer a tratabilidade em cenários de grande escala.

Atualmente, a melhor aproximação conhecida para o problema de Steiner é de 1,39, obtida por Byrka *et al.* [3]. Além disso, Chlebík e Chlebíková demonstraram que, a menos que $P = NP$, não é possível obter uma aproximação melhor que 1,01063 [5]. É possível observar que ainda existe uma lacuna entre a aproximação e o limitante inferior até então encontrado para esse problema.

Em suma, neste relatório estão abordados algoritmos e suas análises para o problema de enfoque, também para outros problemas de abordagem similar. Estes últimos, presentes no Apêndice B. Vale ressalva para o Problema do Caixeiro Viajante nessa seção. Além do estudo de duas aproximações para este problema, foi feita em detalhe a prova para sua inaproximabilidade para uma entrada de um grafo não completo. Contraposto a isso, também foi realizada a prova para aproximabilidade de um grafo não completo como entrada para o problema da árvore de Steiner.

Referências

[1] Bondy, J., Murty, U.: Graph Theory with Applications. Elsevier, New York (1976)

[2] Brucker, P.: Scheduling Algorithms. Springer-Verlag, Berlin, Heidelberg (2001)

[3] Byrka, J., Grandoni, F., Rothvoß, T., Sanità, L.: An improved lp-based approximation for steiner tree. In: Proceedings of the Forty-Second ACM Symposium on Theory of Computing. p. 583–592. STOC '10, Association for Computing Machinery, New York, NY, USA (2010). DOI 10.1145/1806689.1806769, https://doi.org/10.1145/1806689.1806769

[4] de Carvalho, M., Cerioli, M., Dahab, R., Feofiloff, P., Fernandes, C., Ferreira, C., Guimarães, K., Miyazawa, F., de Pina Jr., J., Soares, J., Wakabayashi, Y.: Uma Introdução Sucinta a Algoritmos de Aproximação. Publicações Matemáticas do IMPA, São Paulo, Brazil (2001)

[5] Chlebík, M., Chlebíková, J.: The steiner tree problem on graphs: Inapproximability results. Theoretical Computer Science **406**(3), 207–214 (2008). DOI https://doi.org/10.1016/j.tcs.2008.06.046, https://www.sciencedirect.com/science/article/pii/S0304397508004660, algorithmic Aspects of Global Computing

[6] Vazirani, V.V.: Approximation Algorithms. Springer-Verlag, Berlin, Heidelberg (2001)

[7] Williamson, D., Shmoys, D.: The Design of Approximation Algorithms. Cambridge University Press, Cambridge, England (2011)

[8] Zelikovsky, A.Z.: An 11/6-approximation algorithm for the network steiner problem. Algorithmica **9**, 463–470 (1993). DOI 10.1007/BF01187035

A Complemento da Fundamentação Teórica

A.1 Redução

Redução é uma técnica em projeto de algoritmos utilizada para transformar instâncias de um problema A em instâncias de outro problema B de tal forma que se possa utilizar algoritmos que resolvam o problema B para resolver o problema A. Aqui consideraremos que os problemas em questão são de decisão.

Dado um problema X qualquer, denotaremos por I_X uma possível instância para X.

Dizemos que um problema A *reduz polinomialmente* para um problema B, e denotamos isso por $A \rightarrow B$, se existe um algoritmo que transforme qualquer instância I_A em uma instância I_B em tempo polinomial e tal que I_A é SIM se e somente se I_B é SIM.

Intuitivamente, um problema X ser redutível a um problema Y significa que Y é pelo menos tão difícil quanto X. Da mesma maneira, pode-se dizer que X é no máximo tão difícil quanto Y. Congruente a isso, a redução nos permite aumentar o conjunto de problemas tratáveis, analogamente pode-se dizer o mesmo para com os problemas intratáveis.

A.2 Classes de Complexidade

A classe P (de *polynomial*) contém problemas de decisão para os quais existe algum algoritmo que os resolva em tempo polinomial no tamanho de suas entradas.

Outro grupo de problemas comumente citados são os problemas de decisão na classe NP (de *nondeterministic polynomial*). Estes problemas são os quais, a partir de um certificado positivo (conjunto de dados extra), é possível determinar, em tempo polinomial, se uma dada instância é SIM.

Em contraposição, existem os problemas pertencentes à classe NP-difícil (*NP-hard*). Um problema X está em NP-difícil se, para todo problema em NP, este é redutível para X. Estes problemas são pelo menos tão difíceis quanto os problemas mais difíceis em NP e podem ou não estar em NP.

Outra classificação importante são os problemas NP-completos. Um problema é dito NP-completo se (i) ele faz parte da classe NP-difícil e, (ii) está em NP. Ou seja, os problemas NP-difíceis que estão em NP são chamados de NP-completos.

Pela definição, se descoberta a solução para um problema NP-difícil ou um problema NP-completo, isso significará que P = NP.

A.3 Problemas de Otimização

Problemas de Otimização são aqueles que buscam o melhor cenário possível considerando determinadas restrições. Tal-qualmente observado na teoria, a majoritária parte dos problemas

que foram tratados durante a pesquisa se tratam de problemas da classe NP-difícil, ou seja, não possuem solução conhecida que possa ser encontrada em tempo polinomial. Em prol do entendimento, problemas que são resolvidos em tempo polinomial são aqueles que possuem algum algoritmo cuja execução é limitada por uma função polinomial baseada no tamanho de sua entrada.

B Algoritmos Complementares

Nesta seção, apresentamos alguns algoritmos fundamentais no estudo de algoritmos de aproximação. Para estes algoritmos e demais complementações teóricas, foram utilizados os livros "*The Design of Approximation Algorithms*" [7] e "Uma Introdução Sucinta a Algoritmos de Aproximação" [4].

B.1 O Problema do Caixeiro Viajante

Um *Ciclo Hamiltoniano* em um grafo G é um ciclo que visita cada vértice do grafo exatamente uma vez, retornando ao vértice inicial. Formalmente, dado um grafo $G = (V, E)$, onde V é o conjunto de vértices e E é o conjunto de arestas, um Ciclo Hamiltoniano é uma sequência de vértices $v_1, v_2, \ldots, v_n$, tal que cada vértice $v_i \in V$ é distinto, $(v_i, v_{i+1}) \in E$ para $1 \leq i < n$, e $(v_n, v_1) \in E$, completando o ciclo.

Problema B.1 (Caixeiro Viajante)**.** *Dados um grafo G, uma função de custo $w\colon E(G) \to \mathbb{Q}^+$, encontrar um ciclo hamiltoniano $\mathcal{H}$, ou seja, um ciclo que visita cada vértice exatamente uma vez, que minimize $w(\mathcal{H}) = \sum_{e \in E(\mathcal{H})} w(e)$.*

Denotaremos tal problema por TSP, do inglês *Travelling Salesman Problem*.

A seguir, mostraremos que o problema em questão é inaproximável, isto é, não existe um algoritmo eficiente que possa encontrar solução com determinada garantia de distância da solução ótima em seu caso geral, a menos que $P = NP$

Teorema B.1 ([6])**.** *Para qualquer função computável em tempo polinomial $\alpha(n)$, onde n representa o número de vértices do grafo, o Problema do Caixeiro Viajante não pode ser aproximado dentre de um fator $\alpha(n)$, a menos que $P = NP$.*

Demonstração. Suponha, para fins de contradição, que exista um algoritmo eficiente de fator $\alpha(n)$, $\mathcal{A}$, para o Problema do Caixeiro Viajante.

Vamos mostrar que $\mathcal{A}$ pode ser utilizado para resolver o problema de decisão do Ciclo Hamiltoniano. Por este ser um problema NP-completo, chegaremos em P = NP.

Reduziremos do problema do Ciclo Hamiltoniano para o TSP. Seja G um grafo de ordem n no qual deseja-se descobrir se há um Ciclo Hamiltoniano. Transforme o grafo G original em um grafo G' completo. Além disso, para cada $e \in E(G')$, faça:

- se $e \in E(G)$, atribua $w(e) = 1$;
- se $e \notin E(G)$, atribua $w(e) = \alpha(n) \cdot n$.

Note que se G tem ciclo Hamiltoniano, então $\mathrm{OPT}_{TSP}(G', w) = n$. Caso contrário, $\mathrm{OPT}_{TSP}(G', w) > \alpha(n) \cdot n$.

Tome S como a solução devolvida por $\mathcal{A}$ para o Problema do Caixeiro Viajante sobre G', w. Então vale que $\mathrm{OPT}_{TSP}(G', w) \leq w(S) \leq \alpha(n)\,\mathrm{OPT}_{TSP}(G', w)$. Sendo assim, usando a observação anterior, têm-se dois casos:

- $w(S) > \alpha(n) \cdot n$, então não existe ciclo Hamiltoniano em G; ou
- $w(S) \leq \alpha(n) \cdot n$, então existe ciclo Hamiltoniano em G.

Portanto, a menos que P = NP, não é possível que haja uma aproximação para o TSP no caso geral. □

Pelo resultado anterior, não há esperanças de se conseguir um algoritmo de aproximação para o TSP. Por isso, na literatura, costuma-se lidar com variações mais restritas desse problema. No restante dessa seção, lidaremos com o problema do Caixeiro Viajante Métrico, no qual recebe-se um grafo G e uma função de pesos nas arestas $w\colon E(G) \to \mathbb{Q}^+$ que satisfaz a desigualdade triangular.

B.1.1 Algoritmo de aproximação de fator 2 para o TSP Métrico

A princípio, comecemos estabelecendo um limitante inferior para o valor de uma solução ótima do Problema do Caixeiro Viajante Métrico.

Tome $\mathcal{S}$ como solução ótima para a instância (G, w). Faça $T = \mathcal{S} - e$, onde $e \in E(G)$ é uma aresta qualquer (por exemplo, uma de menor custo). Sendo assim, se T^* é uma árvore geradora mínima de (G, w) então, certamente,

$$w(T^*) \leq w(\mathcal{S})\,, \tag{45}$$

isto é,

$$\mathrm{OPT}_{\mathsf{MST}}(G, w) \leq \mathrm{OPT}_{TSP}(G, w)\,. \tag{46}$$

A ideia do algoritmo é construir uma árvore geradora mínima, "passear" por suas arestas, visitando cada aresta exatamente duas vezes, e transformar esse passeio em um ciclo que não repita vértices. Ele encontra-se formalizado no Algoritmo 3.

Vamos analisar o tempo do Algoritmo 3, e seja $n = |V(G)|$ e $m = |E(G)|$. Tem-se que, para encontrar uma MST, leva-se tempo $O(m \log n)$ utilizando o algoritmo de Prim. Já para encontrar uma trilha Euleriana, podemos usar um algoritmo similar a uma busca em profundidade, em tempo $O(n+m)$. Finalmente, para encontrar o ciclo Hamiltoniano, basta percorrer $\mathcal{E}$, ignorando as repetições, o que leva tempo $O(m)$. Por isso, o algoritmo tem tempo total $O(m \log n)$, que é o mesmo tempo de encontrar a MST.

Algoritmo 3 Algoritmo APROX-2-TSP, que recebe um grafo G e uma função de peso $w\colon E(G) \to \mathbb{Q}^+$.

1: **Função** APROX-2-TSP(G, w)
2: Encontre MST T^* em G, w
3: Duplique cada aresta em T^* para obter um grafo Euleriano $\mathcal{T}$
4: Encontre uma trilha Euleriana $\mathcal{E}$ em $\mathcal{T}$
5: Seja $\mathcal{H}$ um ciclo Hamiltoniano obtido a partir de $\mathcal{E}$ ignorando vértices repetidos
6: **Devolve** $\mathcal{H}$

Teorema B.2. *O Algoritmo 3 é uma 2-aproximação para o Problema do Caixeiro Viajante Métrico.*

Demonstração. O segundo passo da função APROX-2-TSP é duplicar as arestas da árvore geradora mínima T^* e a trilha Euleriana contém exatamente as mesmas arestas. Sendo assim,

$$w(\mathcal{E}) = w(\mathcal{T}) = 2\, w(T^*)\,. \tag{47}$$

Ademais, a desigualdade triangular garante que os "atalhos" realizados para gerar o ciclo não aumentam o custo de $\mathcal{H}$. Por isso,

$$w(\mathcal{H}) \leq w(\mathcal{E})\,. \tag{48}$$

De (48) e (47), tem-se que

$$w(\mathcal{H}) \leq 2\, w(T^*)\,. \tag{49}$$

Finalmente, de (49) e (46), obtém-se:

$$w(\mathcal{H}) \leq 2\,\mathrm{OPT}_{TSP}(G, w) \quad .$$

□

B.1.2 Algoritmo de aproximação de fator $\frac{3}{2}$ para o TSP Métrico

Para a seção anterior, vimos um algoritmo de aproximação para esse problema com fator de aproximação 2. A ideia anterior foi utilizar algum algoritmo que encontrasse uma árvore geradora mínima, e então duplicarmos suas arestas a fim de obter um grafo com todos os vértices possuindo grau par. Obter um grafo com essa condição é necessário, pois uma trilha Euleriana só existe se cada vértice do grafo tem grau par.

Nesse sentido, para esta seção, mostraremos um algoritmo de aproximação que obedece essa restrição. Já que precisamos de um grafo com grau par para todos os vértices, nos preocuparemos com aqueles que não o possuem.

O algoritmo de aproximação tratado nessa seção, ao invés de duplicar todas arestas de MST, adiciona um emparelhamento perfeito de custo mínimo sobre os vértices de grau ímpar da MST.

Tome T^* como essa MST.

Para que esse emparelhamento seja possível, devemos ter um grafo no qual a quantidade de vértices seja par. Tomamos $O \subseteq V(T^*)$ como sendo o conjunto de vértices de grau ímpar em T^*. Como $|O|$ é par, pelo Corolário 2.1, podemos de fato encontrar um emparelhamento perfeito de custo mínimo em $G[O]$. Tome esse emparelhamento como M^*.

Depois disso, combine T^* e M^*. Obtemos um grafo Euleriano para, por fim, construir uma trilha Euleriana $\mathcal{E}$ e então obter um ciclo Hamiltoniano removendo os vértices repetidos de $\mathcal{E}$. Essa ideia é formalizada no Algoritmo 4.

Algoritmo 4 Algoritmo APROX-3/2-TSP, que recebe um grafo G e uma função de peso $w\colon E(G) \to \mathbb{Q}^+$.

1: Encontre MST T^* em G, w
2: Seja O o conjunto dos vértices de grau ímpar de T^*
3: Encontre um emparelhamento perfeito de peso mínimo M^* em $G[O]$
4: Combine as arestas de T^* e M^* para obter um grafo Euleriano $\mathcal{T}$
5: Encontre uma trilha Euleriana $\mathcal{E}$ em $\mathcal{T}$
6: Seja $\mathcal{H}$ um ciclo Hamiltoniano obtido a partir de $\mathcal{E}$ ignorando vértices repetidos
7: **Devolve** $\mathcal{H}$

Vamos analisar o tempo do Algoritmo 4, e seja $n = |V(G)|$ e $m = |E(G)|$. Tem-se que, para encontrar uma MST, leva-se tempo $O(m \log n)$ utilizando o algoritmo de Prim. Para encontrar um emparelhamento perfeito de custo mínimo em um grafo completo, podemos usar o algoritmo de Edmonds, que leva tempo $O(n^2 m)$. Já para encontrar uma trilha Euleriana, podemos usar um algoritmo similar a uma busca em profundidade, em tempo $O(n+m)$. Finalmente, para encontrar o ciclo Hamiltoniano, basta percorrer $\mathcal{E}$, ignorando as repetições, o que leva tempo $O(m)$. Por isso, o algoritmo tem tempo total $O(n^2 m)$, que é o mesmo tempo de encontrar o emparelhamento.

Teorema B.3. *O Algoritmo 4 é uma aproximação de fator $\frac{3}{2}$ para o Problema do Caixeiro Viajante Métrico.*

Demonstração. Seja $\mathcal{S}$ uma solução ótima do TSP para a instância G, w. Construa um ciclo $\mathcal{S}'$ a partir de $\mathcal{S}$ ignorando todos os outros vértices e deixando apenas os vértices de O. Certamente $w(\mathcal{S}') \leq w(\mathcal{S})$ pois os atalhos não aumentam o custo do ciclo.

A partir de $\mathcal{S}'$, podemos construir dois emparelhamentos perfeitos M_1 e M_2 tomando-se arestas alternadas. Note que M_1 e M_2 são emparelhamentos em $G[O]$, de forma que seus custos, individualmente, são maiores do que o custo de M^*, que é o mínimo em $G[O]$. Sendo assim,

$$w(M^*) \leq \min\{w(M_1), w(M_2)\} \leq \frac{w(M_1) + w(M_2)}{2} = \frac{w(\mathcal{S}')}{2}\,. \tag{50}$$

E como

$$\frac{w(\mathcal{S}')}{2} \leq \frac{w(\mathcal{S})}{2} = \frac{\mathrm{OPT}_{TSP}(G, w)}{2}\,, \tag{51}$$

então temos que

$$w(M^*) \leq \frac{\mathrm{OPT}_{TSP}(G,w)}{2}\,. \tag{52}$$

Pelo exposto e usando (46):

$$w(\mathcal{H}) \leq w(\mathcal{E}) = w(M^*)+w(T^*) \leq \frac{\mathrm{OPT}_{TSP}(G,w)}{2}+\mathrm{OPT}_{TSP}(G,w) = \frac{3}{2}\mathrm{OPT}_{TSP}(G,w)\,. \tag{53}$$

□

B.2 Escalonamento

O problema do Escalonamento de Tarefas é um problema bastante clássico e bastante estudado, principalmente no âmbito de sistemas operacionais [2]. Descreveremos o seguinte caso: temos m máquinas idênticas e n tarefas com tempo de duração variado, sendo t_i o tempo de executar a tarefa i. Dizemos que um *escalonamento* é uma partição de $\{1,2,\ldots,n\}$ em m subconjuntos $M_1,\ldots,M_m$. Definimos $t(M_j) = \sum_{i\in M_j} t_i$ como sendo o tempo de execução da máquina j. O valor desse escalonamento é chamado de *makespan* e é definido como o tempo máximo entre as m máquinas, ou seja, $\max_j\{t(M_j)\}$.

Problema B.2 (Escalonamento)**.** *Dados inteiros positivos m, n, e um tempo $t_i \in \mathbb{Q}_+$ para cada $i \in \{1,2,\ldots,n\}$, encontrar um escalonamento de makespan mínimo.*

A ideia do algoritmo é gulosamente atribuir cada tarefa à maquina que tenha o menor tempo naquele momento. Isso está formalizado no Algoritmo 5.

Algoritmo 5 Algoritmo ESCALONA, que recebe m máquinas idênticas, n tarefas de tempo variado, pré definidos em t.

```
1: Função ESCALONA(m, n, t)
2:     Faça M_j ← ∅ para todo j ∈ {1,...,m}
3:     Para i de 1 a n faça
4:         seja k uma máquina tal que t(M_k) é mínimo
5:         M_k ← M_k ∪ {i}
6:     Devolve M_1,...,M_m
```

A Figura 6 representa a execução do Algoritmo ESCALONA sobre uma instância específica do problema.

Observe que, ao final do Algoritmo 5, $M_1,\ldots,M_m$ são uma partição de $\{1,2,\ldots,n\}$, melhor dizendo, um escalonamento. Para analisar o tempo de execução, observe que podemos representar cada máquina por meio de um vetor de tamanho n que indica, na i-ésima posição, com o valor 0 ou 1, se a tarefa i está naquela máquina. Também podemos utilizar um vetor de tamanho m para armazenar, na posição j, o tempo total das tarefas que estão na posição j. Com isso, note que o algoritmo leva tempo total $O(nm)$, pois, na linha 4, demora $O(m)$ para encontrar a máquina de tempo mínimo e faz isso n vezes. Por conta da implementação anteriormente exposta, uma única execução de cada linha toma tempo constante.

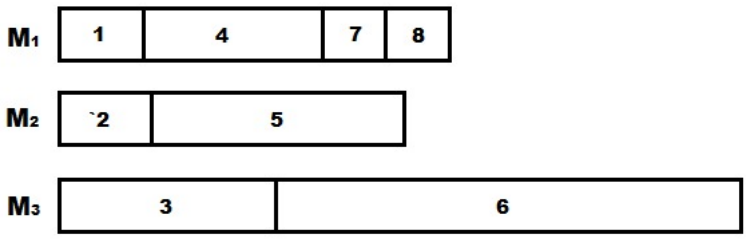

Figura 6: Ilustração de execução do Algoritmo Escalona *em uma instância de $m = 3$ máquinas e $n = 8$ tarefas. O tempo de execução de cada tarefa é mostrado de forma proporcional ao tamanho do retângulo que a representa. Os números dentro do retângulo são os rótulos das tarefas.*

Note, também, que podemos estabelecer dois limitantes inferiores para o valor da solução ótima do nosso problema. Em primeiro lugar, podemos utilizar a tarefa que possui maior tempo, o qual denotamos como $\max(t_i)$, como limitante, uma vez que ela deve estar em alguma máquina. Então

$$\max(t_i) \leq \mathrm{OPT}_{ESC}(m, n, t)\,. \tag{54}$$

Além disso, podemos usar do caso em que podemos fracionar as tarefas, e utilizar disso para limitarmos inferiormente o caso geral:

$$\frac{1}{m}\sum_{i=1}^{n} t_i \leq \mathrm{OPT}_{ESC}(m, n, t)\,. \tag{55}$$

Note que o lado esquerdo da Equação (55) é o valor do tempo de qualquer máquina do caso fracionário, ou seja, do caso em que podemos "quebrar" as tarefas. Por esse motivo podemos usar o tempo de qualquer máquina, já que são iguais, pois as tarefas podem ser quebradas e, assim, distribuídas igualmente.

Teorema B.4. *O Algoritmo 5 é uma aproximação de fator 2 para o Problema do Escalonamento.*

Demonstração. Tome $\tau = t(M_k)$ como sendo o valor do *makespan* da solução existente até a i-ésima execução do laço, isto é, é o tempo da máquina k imediatamente antes da execução da linha 5. Note que a tarefa t_i será, portanto, alocada à máquina k. Isso significa também que, para toda máquina M_j, com $j = \{1, 2, \ldots, m\}$, vale que $\tau \leq t(M_j)$, pela escolha do algoritmo.

Sendo assim, $\tau \cdot m \leq \sum_{j=1}^{m} t(M_j)$, já que todas as máquinas estão cheias até o tempo τ. Além disso, $\sum_{j=1}^{m} t(M_j) \leq \sum_{x=1}^{n} t_x$. Essas expressões juntamente com o limitante da Equação (55) nos permite concluir que $\tau \leq \mathrm{OPT}_{ESC}(m, n, t)$.

Agora, com a execução da linha 5, tem-se

$$t(M_k) = \tau + t_i \leq \mathrm{OPT}_{ESC}(m, n, t) + t_i\,. \tag{56}$$

Sabe-se ainda que, pela Equação (54), o tempo de qualquer tarefa é menor ou igual ao valor

da solução ótima. Por isso,

$$t(M_k) \leq 2\,\mathrm{OPT}_{ESC}(m,n,t)\,. \tag{57}$$

Como isso vale ao final de qualquer iteração, também vale logo após a última iteração e para a máquina que determina o *makespan* da solução. □

Teorema B.5. *O Algoritmo 5 é uma aproximação de fator* $\left(2-\frac{1}{m}\right)$ *para o Problema do Escalonamento.*

Demonstração. Tome $\tau = t(M_k)$ como sendo o valor do *makespan* da solução existente até a i-ésima execução do laço, isto é, é o tempo da máquina k imediatamente antes da execução da linha 5. Note que a tarefa t_i será, portanto, alocada à máquina k. Isso significa também que, para toda máquina M_j, com $j = \{1, 2, \ldots, m\}$, vale que $\tau \leq t(M_j)$, pela escolha do algoritmo.

Sendo assim, $\tau \cdot m \leq \sum_{j=1}^{m} t(M_j)$, já que todas as máquinas estão cheias até o tempo τ. Além disso, $\sum_{j=1}^{m} t(M_j) \leq (\sum_{x=1}^{n} t_x) - t_i$, já que a tarefa i ainda não foi alocada. Vale então que

$$\tau \leq \frac{1}{m}\left(\sum_{x=1}^{n} t_x - t_i\right) = \frac{1}{m}\sum_{x=1}^{n} t_x - \frac{1}{m}t_i \leq \mathrm{OPT}_{ESC}(m,n,t) - \frac{1}{m}t_i\,, \tag{58}$$

sendo que a última inequação vale por causa do limitante da Equação (55).

Agora, com a execução da linha (5), tem-se

$$t(M_k) = \tau + t_i \leq \mathrm{OPT}_{ESC}(m,n,t) - \frac{1}{m}t_i + t_i = \mathrm{OPT}_{ESC}(m,n,t) - \left(1-\frac{1}{m}\right)t_i\,. \tag{59}$$

Sabe-se ainda que, pela Equação (54), o tempo de qualquer tarefa é menor ou igual ao valor da solução ótima. Por isso,

$$t(M_k) \leq \left(2-\frac{1}{m}\right)\,\mathrm{OPT}_{ESC}(m,n,t)\,. \tag{60}$$

Como isso vale ao final de qualquer iteração, também vale logo após a última iteração e para a máquina que determina o *makespan* da solução. □