Algoritmos para o problema da árvore de Steiner com coleta de prêmios

Camila Mari Matsubara

Dissertação apresentada
ao
Instituto de Matemática e Estatística
da
Universidade de São Paulo
para
obtenção do título
de
Mestre em Ciências

Programa: Ciência da Computação

Orientador: Prof. Dr. José Coelho de Pina Jr.

— São Paulo, Dezembro de 2012 —

Algoritmos para o problema da árvore de Steiner com coleta de prêmios

Esta versão da dissertação contém as correções e alterações sugeridas pela Comissão Julgadora durante a defesa da versão original do trabalho, realizada em 14 de dezembro de 2012. Uma cópia da versão original está disponível no Instituto de Matemática e Estatística da Universidade de São Paulo.

Comissão Julgadora:

- Prof. Dr. José Coelho de Pina Jr. (orientador) - IME-USP
- Prof. Dr. Paulo Feofiloff - IME-USP
- Prof. Dr. Fábio Henrique Viduani Martinez - UFMS

Agradecimentos

Eu agradeço

ao meu orientador Professor José Coelho pela imensurável dedicação, disponibilidade e paciência. Foi o melhor orientador que eu poderia ter, por quem eu tenho muito respeito e admiração;

aos membros da banca de qualificação, Prof. Paulo Feofiloff e Prof. Carlos Eduardo Ferreira e da comissão julgadora Prof. Paulo Feofiloff e Prof. Fábio Henrique Viduani Martinez pela revisão e sugestões;

aos meus pais Sueli e Eduardo pelo amor incondicional e pela construção da pessoa que sou;

à minha irmã Emy por sua companhia, admiração e também por contribuições neste texto;

ao Alberto pela confiança, pelo apoio em todos os momentos e por ser a minha inspiração e a minha certeza;

a todos os amigos do BCC Marcela, César, João, Natan,... e do trabalho Dani, Carol, Marcos, Steven, Thiago, Bruno, César, Aline e Herbert pelo aprendizado e pela diversão, pela compreensão e pelo incentivo que me deram durante este estudo;

a todos que, direta ou indiretamente, contribuíram para a realização deste mestrado.

Muito obrigada!

Resumo

Algoritmos para o problema da árvore de Steiner com coleta de prêmios

Neste projeto estudamos algoritmos de aproximação para o problema da árvore de Steiner com coleta de prêmios. Trata-se de uma generalização do problema da árvore de Steiner, onde é dado um grafo com custos positivos nas arestas e penalidades positivas nos vértices. O objetivo é encontrar uma subárvore do grafo que minimize a soma dos custos das arestas mais a soma das penalidades dos vértices que não pertencem à subárvore. Em 2009, os autores Archer, Bateni, Hajiaghayi e Karloff obtiveram pela primeira vez um algoritmo com fator de aproximação estritamente menor do que 2. Além de analisarmos este algoritmo, estudamos também a implementação de algoritmos 2-aproximação para o problema da árvore de Steiner e da árvore de Steiner com coleta de prêmios.

Palavras-chave: árvore de Steiner, algoritmo de aproximação, problema com coleta de prêmios.

Abstract

Algorithms for the prize-collecting Steiner tree problem

In this project we analyze approximation algorithms for the prize-collecting Steiner tree problem. This is a generalization of the Steiner tree problem, in which it is given a graph with positive costs in edges and positive penalties in vertices. The goal is to find a subtree of the graph that minimizes the sum of costs of edges plus the sum of the penalties of the vertices that don't belong to the subtree. In 2009, the authors Archer, Bateni, Hajiaghayi e Karloff described for the first time an algorithm with approximation factor strictly less than 2. Besides analyzing this algorithm, we also study the implementation of 2-approximation algorithms for the Steiner tree problem and prize-collecting Steiner tree problem.

Keywords: Steiner tree, approximation algorithm, prize-collecting problem.

Sumário

Lista de Abreviaturas

GW	autores Goemans e Williamson
ABHK	autores Archer, Bateni, Hajiaghayi e Karloff
MinST	Problema da árvore de Steiner
PCST	Problema da árvore de Steiner com coleta de prêmios
R-PCST	Problema da árvore de Steiner com coleta de prêmios enraizada
MinPath	Problema do caminho mínimo
MST	Problema da árvore geradora mínima
k-MST	Problema da k-árvore
CE	Problema da cobertura exata
MinST-GW	Algoritmo para MinST devido a GW
PCST-GW	Algoritmo para PCST devido a Feofiloff et al. e baseado em GW.
R-PCST-GW	Algoritmo para R-PCST devido a GW
R-PCST-ABHK	Algoritmo para R-PCST devido a ABHK

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

O problema da árvore de Steiner consiste em, dado um grafo com custo nas arestas e um conjunto de vértices denominados terminais, determinar um subgrafo conexo que contém todos os vértices terminais e cuja soma dos custos das arestas seja a menor possível. Ele pode conter outros vértices além dos terminais. Este subgrafo de custo mínimo existe para qualquer grafo conexo dado, e podemos supor que é uma árvore quando os custos das arestas são positivos.

O nome do problema é uma homenagem ao matemático suíço Jakob Steiner, de grande influência e destaque no estudo de geometria. Originalmente, o problema foi proposto pelo matemático francês Pierre de Fermat no início do século XVII com a seguinte descrição [Mac87]:

> Dado um triângulo ABC, localizar um ponto P cuja soma das distâncias PA, PB e PC seja mínima.

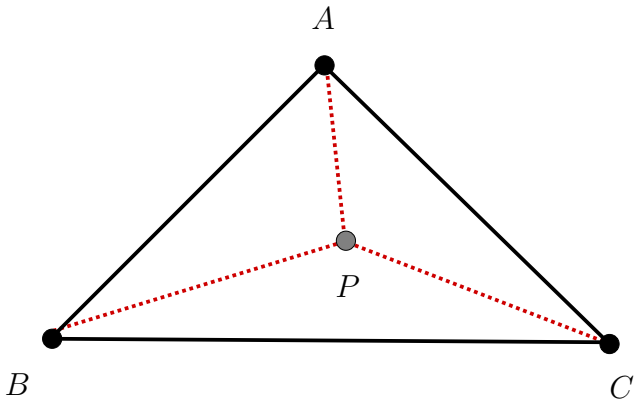


Figura 1.1: *Descrição de Fermat.*

Sabe-se que o físico e matemático italiano Evangelista Torricelli resolveu o problema antes de 1640. Ele afirmou que as circunferências que circunscrevem os triângulos equiláteros contruídos com os lados do triângulo dado intersectam o ponto procurado, conhecido como *ponto de Torricelli*.

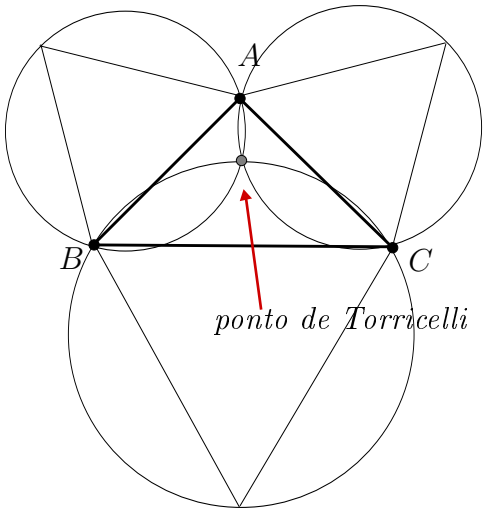


Figura 1.2: *Ponto de Torricelli.*

Para um triângulo com ângulo maior ou igual a 120^o, o ponto P coincide com o vértice deste ângulo.

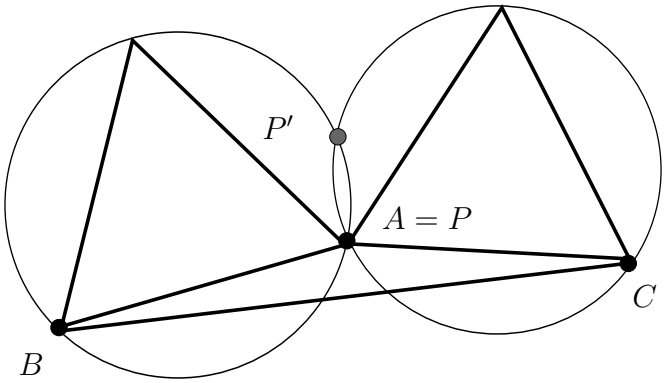


Figura 1.3: *Ponto P coincide com vértice de ângulo maior do que 120^o.*

No século XIX, Jakob Steiner escreveu sobre o problema de Fermat generalizado: localizar um ponto P que minimize a soma ponderada das distâncias de P a uma quantidade arbitrária de pontos no plano. Então, o problema se popularizou com nome problema de Steiner no livro *What is Mathematics?* publicado em 1941, de Richard Courant e Herbert Robbins [CR41].

Os progressos no problema da árvore de Steiner foram rápidos desde 1990, quando Alexander Zelikovsky [Zel93] apresentou o fator de aproximação de 1,833 – o primeiro algoritmo a melhorar o ingênuo fator 2. Piotr Berman e Viswanathan Ramaiyer [BR92] diminuíram o fator para 1,746 em 1992. Zelikovsky [Zel95] alcançou uma 1,693-aproximação em 1997, seguida da melhoria de Hans Jürgen Prömel e Angelika Steger [PS00] para 1,667 e de Marek Karpinski e Zelikovsky [KZ97] para 1,644 em 1997. Em 1999, Stefan Hougardy e Prömel [HP99] apresentaram uma 1,598-aproximação.

Em 2005, Gabriel Robins e Zelikovsky [RZ05] publicaram o fator de aproximação de 1,55 que foi o menor encontrado para o problema da árvore de Steiner até o ano de 2010, quando Jaroslaw Byrka, Fabrizio Grandoni, Thomas Rothvoße Laura Sanità [BGRS10] apresentaram a melhoria para o fator de 1,39.

No problema da árvore de Steiner com coleta de prêmios (PCST, *Prize-Collecting Steiner Tree*), além do custo das arestas, o grafo também possui um valor de penalidade para cada vértice. Não é dado um conjunto de vértices terminais. Desta feita, o objetivo é obter uma árvore que minimize a soma dos custos de suas arestas e as penalidades dos vértices que não pertencem à árvore. O problema da árvore de Steiner pode ser visto como um caso particular do PCST, quando os vértices terminais têm valor de penalidade bem grande e os demais vértices têm penalidade zero.

Este problema tem aplicações no projeto de circuitos elétricos e redes de comunicação. Além de ser uma ferramenta teórica útil para ajudar a resolver outros problemas de otimização, foi aplicado pela empresa AT&T com bons resultados para otimização de redes de telecomunicações do mundo real [ABHK09]. Ivana Ljubic, René Weiskircher, Ulrich Pferschy, Gunnar Klau, Petra Mutzel e Matteo Fischetti utilizaram o PCST para modelar a instalação de cabos de fibra ótica na Alemanha [LWP+05].

O primeiro algoritmo de aproximação para o PCST foi obtido por Daniel Bienstock, Michel Xavier Goemans, David Simchi-Levi e David Paul Williamson [BGSLW93] em 1993, e seu fator de aproximação é 3. Este algoritmo encontra uma solução aproximada através do arredondamento de uma solução da relaxação do programa linear. Mais tarde, em 1995, Goemans e Williamson [GW95] desenvolveram um algoritmo com fator de aproximação $(2-\frac{1}{n-1})$ baseado em um esquema primal-dual para a versão enraizada do problema, onde n é o número de vértices do grafo dado.

Executando o algoritmo de Goemans e Williamson para todas as possibilidades de raiz, obtém-se uma $(2-\frac{1}{n-1})$-aproximação que consome tempo $O(n^3 \log n)$.

David Stifler Johnson, Maria Minkoff e Steven Phillips [JMP00] propuseram, em 2000, uma modificação no algoritmo que permite executar o esquema primal-dual apenas uma vez resultando em um tempo de execução de $O(n^2 \log n)$. Entretanto, este algoritmo não mantém o mesmo fator de aproximação do algoritmo de Goemans e Williamson, como demonstrado por Paulo Feofiloff, Cristina Gomes Fernandes, Carlos Eduardo Ferreira e José Coelho de Pina [FFFdP07] em 2006. Estes mesmos autores, em 2007 [FFFdP07] publicaram uma modificação no algoritmo de Goemans e Williamson baseado em um programa linear sutilmente diferente, que resulta em um fator de aproximação de $(2-\frac{2}{n})$ para a versão não-enraizada e pode ser implementada com tempo de execução de $O(n^2 \log n)$. Mais recentemente, em 2009, Aaron Archer, Mohammad Hossein Bateni, Mohammad Taghi Hajiaghayi e Howard Jeffrey Karloff [ABHK09, ABHK11] obtiveram um fator de aproximação menor do que 2 para o PCST.

Organização do texto

O texto está organizado da seguinte forma: No capítulo 2 apresentamos algumas definições básicas de teoria dos grafos, programação linear e algoritmos de aproximação e fixamos parte da notação que é empregada ao longo do texto.

Em seguida, no capítulo 3, descrevemos o problema da árvore de Steiner, um algoritmo

para resolvê-lo e a análise do seu fator de aproximação.

O problema da árvore de Steiner com coleta de prêmios é apresentado no capítulo 4. Aqui vemos primeiramente a versão não-enraizada do problema e os conceitos básicos que o envolvem. Além disso, descrevemos um algoritmo de 2-aproximação juntamente com sua análise. Em seguida, consideramos a versão do problema da árvore de Steiner com coleta de prêmios com raiz. Fazemos uma análise comparativa da relaxação do programa linear e uma adaptação do algoritmo anterior para resolver instâncias desta versão.

O capítulo 5 descreve resultados recentes de Archer, Bateni, Hajiaghayi e Karloff que obtiveram um algoritmo com fator de aproximação estritamente menor do que 2 para o problema da árvore de Steiner com coleta de prêmios.

Finalmente, no capítulo 6, escrevemos as nossas considerações finais, contribuições e possíveis trabalhos futuros.

Capítulo 2

Preliminares

Este capítulo apresenta notações básicas, conceitos e definições envolvendo grafos, algoritmos de aproximação e programação linear que são utilizados ao longo da dissertação. A notação e as definições de grafos utilizadas são as de Paulo Feofiloff, Yoshiharu Kohayakawa e Yoshiko Wakabayashi [FKW04]. Já as de algoritmos de aproximação e programação linear são as mesmas de Cristina Gomes Fernandes, Flávio Keidi Miyazawa, Márcia Cerioli e Paulo Feofiloff [dCCD+01].

2.1 Notação básica

O conjunto dos números racionais é denotado por $\mathbb{Q}$ e o conjunto dos números racionais não-negativos por $\mathbb{Q}_{\geq}$.

Dada uma função f que associa um número $\mathbb{Q}_{\geq}$ ou $\mathbb{Q}$ a cada elemento de um conjunto S, denotamos o valor que f associa a um elemento s de S por f_s. Além disso, se X é um subconjunto de S, utilizamos a abreviatura $f(X)$ para representar o valor de $\sum_{s \in X} f_s$.

2.2 Grafos

Um **grafo** (não-dirigido) é um par $G = (V_G, E_G)$, onde V_G é um conjunto arbitrário finito e E_G é um conjunto de pares não-ordenados de elementos de V_G. Os elementos de V_G são chamados de **vértices** e os de E_G são chamados de **arestas**. Denotaremos uma aresta $\{u, v\}$ simplesmente por uv ou e_{uv}. Se u e v são vértices em V_G e uv é uma aresta em E_G, dizemos que uv **incide** em u e em v e que u e v são as **pontas** ou **extremidades** da aresta. Além disso, dizemos que os vértices u e v são **adjacentes**.

O número de vértices do grafo G é denotado por $n_G := |V_G|$. Entretanto, usamos a notação simplificada n quando nos referimos ao grafo do contexto.

Um grafo é **completo** se uv é uma aresta do grafo para todo par de vértices distintos u e v.

Dados um grafo $G = (V_G, E_G)$ e um subconjunto de vértices S, definimos o **corte** $\delta_G(S)$ como o conjunto de arestas que têm uma extremidade em S e outra em $V_G \setminus S$. Dizemos que um corte $\delta_G(S)$ **separa** dois vértices u e v, se u está em S e v em $V_G \setminus S$, ou vice-versa.

O **grau** de um vértice v é o número de arestas que incidem em v. Em outras palavras, é a cardinalidade do corte $\delta_G(\{v\})$.

Um **caminho** em um grafo G é uma sequência $\langle v_0, a_1, v_1, a_2, v_2, \ldots, a_p, v_p\rangle$ tal que v_i é um vértice em V_G para todo $i = 0, 1, 2, \ldots, p$, os vértices são dois a dois distintos, a_i é uma aresta em E_G e vale que $a_i = v_{i-1}v_i$ para todo $i = 1, 2, \ldots, p$. Dizemos que v_0 e v_p são as **pontas** ou **extremidades do caminho** e os demais vértices são chamados **internos**. Podemos denotar um caminho por sua sequência de vértices.

Se c é uma função que associa um custo c_e em $\mathbb{Q}_{\geq}$ a cada aresta e de E_G e P é um caminho, denotamos por $c(P)$ o **custo do caminho** P, que é a soma dos custos de todas as arestas em P.

O problema do caminho mínimo (*Shortest Path Problem*) é definido da seguinte maneira:

> **Problema MinPath(G, c, s, t):** Dados um grafo conexo G, um custo c_e em $\mathbb{Q}_{\geq}$ para cada e em E_G e dois vértices s e t de V_G, encontrar um caminho entre s e t de custo mínimo.

Um grafo é **conexo** se, para qualquer par de vértices u e v, existe um caminho conectando estes vértices.

Um **subgrafo** de um grafo G é qualquer grafo H contido em G, isto é, um grafo H tal que $V_H \subseteq V_G$ e $E_H \subseteq E_G$. Um subgrafo conexo maximal de um grafo é chamado **componente**. Se um grafo é conexo, ele possui apenas um componente. Um subgrafo $H = (V_H, E_H)$ é chamado **gerador** de G se $V_H = V_G$.

Seja $X \subseteq V_G$ um subconjunto de vértices de um grafo G. Definimos o subgrafo **induzido** por X como o grafo $H = (X, E_H)$, onde o conjunto de vértices de H é o conjunto X e as arestas de H são todas as arestas de E_G que têm as duas pontas em X. Este subgrafo induzido é denotado por $G[X]$.

Um **circuito** em um grafo G é uma sequência $\langle v_1, a_1, v_2, a_2, \ldots, v_p, a_p\rangle$ tal que v_i é um vértice em V_G para todo $i = 1, 2, \ldots, p$, os vértices são dois a dois distintos, a_i é uma aresta em E_G para todo $i = 1, 2, \ldots, p$, $a_i = v_iv_{i+1}$ para $i = 1, 2, \ldots, p-1$ e $a_p = v_pv_1$. Assim como os caminhos, podemos denotar os circuitos por sua sequência de vértices.

Uma **floresta** é um grafo que não contém circuitos. Um grafo que não tem circuitos e é conexo denomina-se **árvore**. Assim, cada componente de uma floresta é uma árvore. Um vértice que tem grau 1 em uma árvore é chamado de **folha**.

Se c é uma função que associa um custo c_e em $\mathbb{Q}_{\geq}$ a cada aresta e de E_G e F é uma floresta, denotamos por $c(F)$ o **custo da floresta** F, que é a soma dos custos de todas as arestas em F.

Uma **árvore geradora** de um grafo G é uma árvore que é subgrafo gerador de G. Em outras palavras, é uma árvore de G que contém todos os seus vértices.

O problema da árvore geradora mínima (*Minimum Spanning Tree Problem*) é definido da seguinte maneira:

> **Problema MST(G, c):** Dados um grafo conexo G e um custo c_e em $\mathbb{Q}_{\geq}$ para cada e em E_G, encontrar uma árvore geradora de custo mínimo.

Um grafo D é **dirigido**, ou chamado de **digrafo**, quando suas arestas são pares ordenados do tipo uv. Arestas dirigidas são chamadas de **arcos**. O vértice u é chamado de **extremidade inicial** do arco e o vértice v, de **extremidade final** do arco.

Para grafos dirigidos existe o conceito de arborescência. Uma **arborescência** é uma sequência da forma $\langle v_0, a_1, v_1, a_2, v_2, \ldots, a_k, v_k\rangle$, onde $v_0, \ldots, v_k$ são vértices distintos dois a dois, $a_1, \ldots, a_k$ são arcos distintos dois a dois e, para cada j, a_j é um arco da forma $v_i v_j$, com $i < j$. O primeiro vértice da sequência, v_0, é a **raiz da arborescência**. Cada vértice da sequência, com exceção da raiz, é extremidade final de exatamente um arco da arborescência. A raiz não é extremidade final de nenhum arco.

Estendemos este conceito para grafos não-dirigidos: uma **arborescência** é uma árvore com raiz que induz uma direção das arestas na qual a extremidade inicial sempre é o vértice mais próximo da raiz.

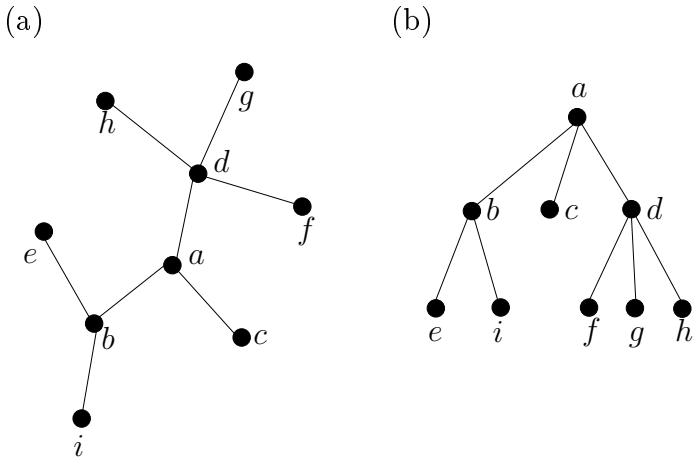


Figura 2.1: (a) *Árvore.*
(b) *Arborescência induzida com raiz a.*

2.3 Algoritmos de aproximação

Um **problema de otimização** consiste de:

- um conjunto de **instâncias**;
- um conjunto de **candidatos a solução**, ou **soluções viáveis**, para cada instância $\mathcal{I}$; e
- uma função que define o **valor** ou **custo** de um candidato a solução.

Um problema de otimização pode ser **de minimização** ou **de maximização**. O MIN-PATH e MST apresentados na seção anterior são exemplos de problemas de minimização, que interessam-se pelos candidatos a solução (caminhos e árvores, respectivamente) de valor mínimo. Em ambos os problemas, a função valor é definida como a soma dos custos das arestas que pertencem ao caminho ou à árvore.

Um candidato a solução cujo valor é mínimo em um problema de minimização ou máximo em um problema de maximização é chamada **solução ótima**, ou simplesmente **solução**. Seu valor é dito **ótimo** e é denotado por $\text{opt}_{\mathcal{I}}$ para uma dada instância $\mathcal{I}$.

Um **algoritmo de aproximação** é um algoritmo polinomial para problemas de otimização. Ao invés de buscar uma solução para o problema, ele busca um candidato a solução com um valor que se aproxima do valor ótimo. Por exemplo, considere um problema de minimização e um algoritmo de aproximação que devolve um candidato a solução $C_{\mathcal{I}}$ para uma instância $\mathcal{I}$. Se $S_{\mathcal{I}}$ é uma solução desta instância e vale que:

$$valor(C_{\mathcal{I}}) \leq \alpha\, valor(S_{\mathcal{I}})$$

para **toda** instância $\mathcal{I}$, dizemos que este algoritmo é uma α**-aproximação** para o problema.

Para problemas de maximização, a definição é equivalente: um algoritmo que é α-aproximação devolve um candidato a solução $C_{\mathcal{I}}$ para uma instância cuja solução é $S_{\mathcal{I}}$ e vale para toda instância $\mathcal{I}$:

$$valor(C_{\mathcal{I}}) \geq \alpha\, valor(S_{\mathcal{I}}) \quad .$$

Dizemos que α é o **fator de aproximação** do algoritmo. Note que em problemas de minimização $\alpha \geq 1$ e em problemas de maximização, $0 < \alpha \leq 1$. Se $\alpha = 1$, tem-se um **algoritmo exato**, que encontra uma solução do problema. Portanto, quanto mais próximo de 1 for o fator de aproximação, melhor é o algoritmo.

2.4 Programação linear

Um **problema de programação linear** ou **programa linear** de minimização consiste em:

> **Problema PL(A, b, c):** Dada uma matriz real A indexada por $M \times N$, um vetor real b indexado por M, um vetor real c indexado por N e partições $\{M_1, M_2, M_3\}$ e $\{N_1, N_2, N_3\}$ de M e N respectivamente, podendo um ou mais dos M_i ou N_i ser vazio, encontrar um vetor x indexado por N que

$$\begin{array}{rrcll} \text{minimize} & cx & & & \\ \text{sob as restrições} & (Ax)_i & \geq & b_i & \text{para cada } i \text{ em } M_1\ , \\ & (Ax)_i & = & b_i & \text{para cada } i \text{ em } M_2\ , \\ & (Ax)_i & \leq & b_i & \text{para cada } i \text{ em } M_3\ , \\ & x_j & \geq & 0 & \text{para cada } j \text{ em } N_1\ , \\ & x_j & \leq & 0 & \text{para cada } j \text{ em } N_3\ . \end{array} \tag{2.1}$$

Neste PL, cx é chamada função objetivo e, para cada linha i da matriz A, a relação entre $(Ax)_i$ e b_i corresponde a uma restrição linear sobre o valor de x.

Um vetor x que satisfaz todas estas restrições é chamado de **candidato a solução**, ou **solução viável**. Se tal vetor x existe, dizemos que o problema é **viável**, senão o problema é **inviável**.

Existe uma importante relação de **dualidade** entre programas lineares. O **dual** de um programa de minimização, como o $\mathrm{PL}(A, b, c)$ (2.1), é um programa de maximização da seguinte forma:

> **Problema PD(A, b, c):** Dada uma matriz real A indexada por $M \times N$, um vetor real b indexado por M, um vetor real c indexado por N e partições $\{M_1, M_2, M_3\}$ e $\{N_1, N_2, N_3\}$ de M e N respectivamente, podendo um ou mais dos M_i ou N_i ser vazio, encontrar um vetor y indexado por M que

$$\begin{array}{rlcll} \text{maximize} & yb & & & \\ \text{sob as restrições} & (yA)_j & \geq & c_j & \text{para cada } j \text{ em } N_1\ , \\ & (yA)_j & = & c_j & \text{para cada } j \text{ em } N_2\ , \\ & (yA)_j & \leq & c_j & \text{para cada } j \text{ em } N_3\ , \\ & y_i & \geq & 0 & \text{para cada } i \text{ em } M_1\ , \\ & y_i & \leq & 0 & \text{para cada } i \text{ em } M_3\ . \end{array} \tag{2.2}$$

Convenciona-se chamar o programa do qual o dual se originou de **primal**. Cada componente do vetor x é uma **variável primal** e cada componente do vetor y é uma **variável dual**.

Existe uma relação fundamental entre os candidatos a solução de um programa primal e os candidatos a solução do seu dual:

Lema 2.1 (da dualidade) *Para todo candidato a solução x de um programa primal e todo candidato a solução y do programa dual, vale que $cx \geq yb$.* ■

Este lema também é conhecido como **teorema fraco da dualidade**.

Um programa linear é denominado **inteiro** quando são adicionadas restrições que exigem que algumas variáveis sejam inteiras.

O programa linear obtido de um programa linear inteiro ignorando-se as restrições de integralidade é uma **relaxação linear**.

É comum formular um problema de otimização como um programa linear inteiro. Relaxações lineares de problemas de otimização são, por sua vez, muito usadas no projeto de algoritmos de aproximação.

2.5 Linguagem algorítmica e invariantes

A linguagem algorítmica adotada nesta dissertação é a de Feofiloff [Feo97, Feo03]. Abaixo encontra-se um exemplo onde esta notação é utilizada.

> **Algoritmo** BUSCA(n, v, x): Recebe um inteiro positivo n, um vetor $v[1\,..\,n]$ de números inteiros em ordem não-decrescente e um inteiro x. Devolve TRUE se existe um índice k em $[1\,..\,n]$ tal que $v[k] = x$, e, em caso contrário, devolve FALSE.

O algoritmo é iterativo e no início de cada iteração tem-se inteiros i e j em $[0\,..\,n{+}1]$ e um vetor $v[0\,..\,n{+}1]$ onde supomos que $v[0] = -\infty$ e $v[n{+}1] = +\infty$. No início da primeira iteração $i = 0$ e $j = n{+}1$.

Cada iteração consiste no seguinte:

Caso 1: $i > j$

Devolva FALSE e pare.

Caso 2: $i \leq j$

Escolha um índice k em $[i\,..\,j]$.

Caso 2A: $v[k] = x$

Devolva TRUE e pare.

Caso 2B: $v[k] < x$

Comece uma nova iteração com $k{+}1$ no papel de i.

Caso 2C: $v[k] > x$

Comece uma nova iteração com $k{-}1$ no papel de j.

∎

A ordem em que os casos são enunciados é irrelevante: em cada iteração, qualquer um dos casos aplicáveis pode ser executado. Os casos podem não ser mutuamente exclusivos, e a definição de um caso *não* supõe implicitamente que os demais não se aplicam. São utilizadas ainda expressões como "Escolha um índice k em $[i\,..\,j]$", quando não faz diferença qual o valor escolhido. Portanto, a descrição de um algoritmo pode não ser completamente determinística.

A correção dos algoritmos descritos nesta dissertação baseia-se em demonstrações da validade de invariantes. Estes invariantes são afirmações envolvendo objetos mantidos em cada iteração do algoritmo que são válidas no início de cada iteração. Exemplos de invariantes para o algoritmo descrito acima são:

(i1) i e j são valores em $[0\,..\,n{+}1]$.

(i2) para todo t em $[0\,..\,i{-}1]$, vale que $v[t] < x$.

(i3) para todo t em $[j{+}1\,..\,n+1]$, vale que $v[t] > x$.

Deve-se demonstrar que as relações invariantes valem no início de cada iteração; não fazemos isto para o presente exemplo. Os elementos $v[0]$ e $v[n{+}1]$ foram artificialmente inseridos para simplificar a descrição das relações invariantes acima.

Invariantes nos ajudam a entender e demonstrar a correção de algoritmos. No exemplo em questão vê-se facilmente que quando o algoritmo devolve TRUE, no caso 2A, ele está correto, ou seja, existe k em $[1\,..\,n]$ tal que $v[k] = x$. Ademais, quando $i > j$, do invariante (i2) e (i3), tem-se que $v[k] \neq x$ para todo k em $[0\,..\,i{-}1]$ e $[j{+}1\,..\,n{+}1]$. Como $i > j$, então $v[j] \neq x$ para todo j em $[0\,..\,n{+}1]$. Portanto, no caso 1, o algoritmo corretamente devolve FALSE. Finalmente, o algoritmo para, já que em cada iteração em que não ocorrem os casos 1 e 2A, o caso 2B ou o caso 2C ocorre e, portanto, o valor de i é acrescido de pelo menos 1 ou o valor de j é decrescido de pelo menos 1.

Capítulo 3

Árvores de Steiner

Seja $G = (V_G, E_G)$ um grafo conexo e R um subconjunto de V_G dos chamados vértices **terminais**. Uma **árvore de Steiner** é uma árvore $T = (V_T, E_T)$ subgrafo de G tal que V_T contém todos os vértices terminais ou, em outras palavras, T **conecta** ou **liga** os vértices terminais.

Note que o conceito de árvore de Steiner depende do conjunto R de terminais. Neste texto, ao nos referirmos a uma árvore de Steiner, o conjunto de vértices terminais está frequentemente implícito no contexto.

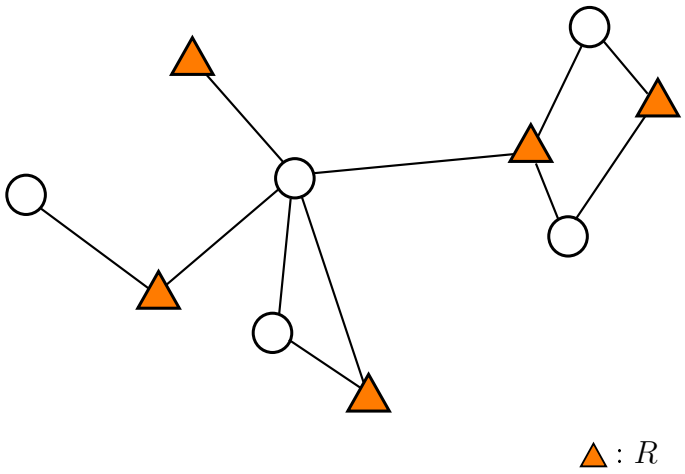


Figura 3.1: *Ilustração de um grafo e um conjunto R de vértices terminais representados por triângulos.*

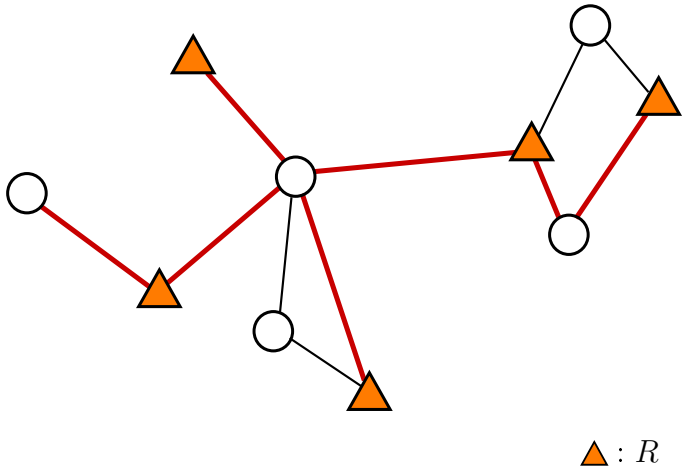


Figura 3.2: *Ilustração de uma árvore de Steiner representada pelas arestas mais espessas.*

Dada uma função custo c de E_G em $\mathbb{Q}_{\geq}$, definimos o **custo da árvore** T como sendo a soma dos custos das arestas que estão em T: $c(T) := \sum_{e \in E_T} c_e$.

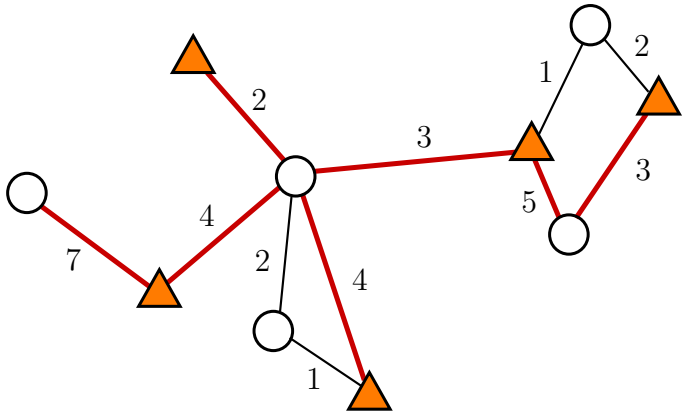


Figura 3.3: *O custo desta árvore de Steiner é* $7+4+2+4+3+5+3=28$.

O problema da árvore de Steiner (*Steiner Tree Problem*) consiste em:

Problema MinST(G, c, R): Dados um grafo conexo G, um custo c_e em $\mathbb{Q}_{\geq}$ para cada e em E_G e um conjunto R de vértices terminais, encontrar uma árvore de Steiner de custo mínimo.

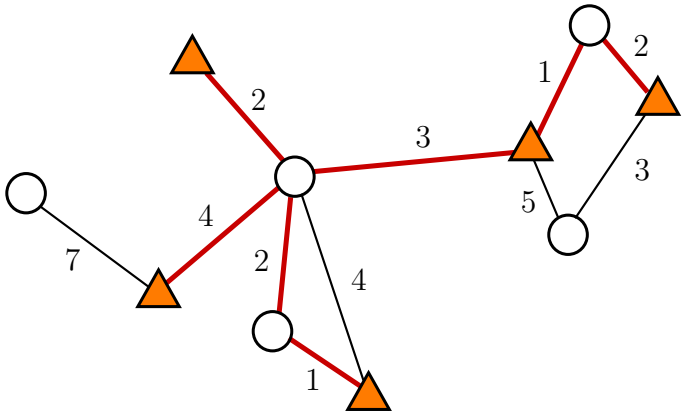


Figura 3.4: *As arestas espessas representam uma árvore de Steiner de custo mínimo. O custo da árvore é* $15 = 4 + 2 + 2 + 1 + 3 + 1 + 2$.

Neste problema de otimização, dada uma árvore T candidata a solução, o valor de T é definido como o custo de T: $c(T)$.

O problema MinST generaliza o problema MinPath (seção 2.2). De fato, o caminho de custo mínimo entre vértices s e t nada mais é do que uma árvore de Steiner de custo mínimo que conecta os terminais $R = \{s, t\}$.

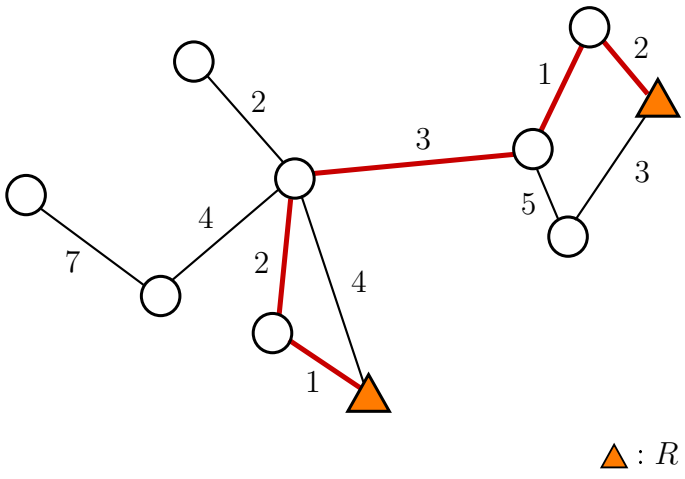


Figura 3.5: *Para* $|R| = 2$ *o* MinST *é equivalente ao* MinPath.

Quando o conjunto R de terminais é V_G o problema MinST é equivalente ao MST (seção 2.2) no qual queremos encontrar uma árvore de custo mínimo que conecte todos os vértices do grafo.

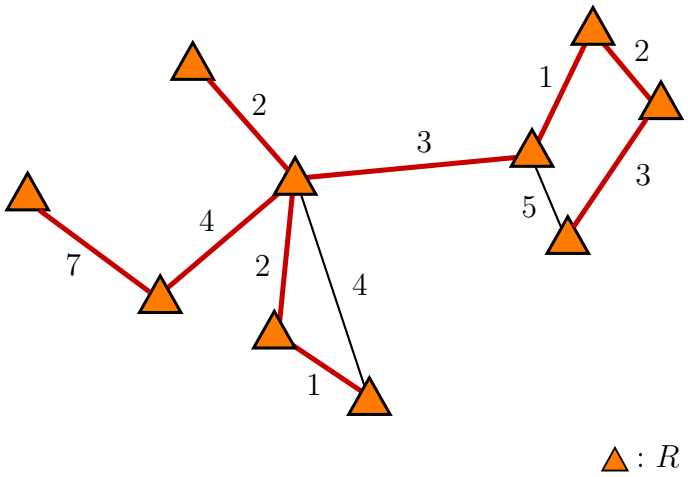


Figura 3.6: *Se todos os vértices do grafo são terminais, o* MinST *é equivalente ao* MST.

Para os problemas MinPath e MST são conhecidos algoritmos eficientes [CLRS01]. Já, em geral, para o MinST não se conhecem algoritmos eficientes, como veremos na próxima seção.

3.1 Complexidade computacional

Resultados de **intratabilidade** mostram que para certos problemas não há, ou não se acredita que haja, algoritmo eficiente para resolvê-lo. Um exemplo desse tipo de resultado pode ser visto nesta seção. É demonstrado a seguir que o problema MinST de encontrar uma árvore de Steiner de custo mínimo é uma tarefa computacionalmente não-trivial [GJ90]. Mais precisamente, é demonstrado que esse problema é NP-difícil. Intuitivamente, o fato desse problema ser NP-difícil significa que à medida que o número de vértices e arestas do grafo cresce, o problema torna-se rapidamente impraticável de ser resolvido por um computador em uma quantidade de tempo razoável. Isto também significa que não há algoritmo eficiente para o problema, a menos que alguns problemas computacionais reconhecidamente difíceis possam ser resolvidos eficientemente [CLRS01, GJ90].

Mostramos que o problema de decisão associado ao MinST é NP-completo, o que nos leva a concluir que o correspondente problema de otimização é NP-difícil. Para isso, apresentamos uma redução do problema da Cobertura Exata (CE) ao MinST, sugerida por Richard Karp em 1972 [Kar72] e descrita por Carlos Eduardo Ferreira em 1989 [Fer89].

O problema de decisão associado ao MinST é definido da seguinte forma:

> **Problema $\textsc{MinST}_d(G, c, R, k)$:** Dados um grafo conexo G, um custo c_e em $\mathbb{Q}_{\geq}$ para cada e em E_G, um conjunto R de vértices terminais e um número k em $\mathbb{Q}$, encontrar (se existe) uma árvore de Steiner cujo custo seja no máximo k.

O problema de decisão associado ao problema de Cobertura Exata é definido da seguinte forma:

Problema $\mathbf{CE}_d(U,\mathcal{F})$: Dados um conjunto $U = \{u_1, u_2, \ldots, u_t\}$ e uma família $\mathcal{F} = \{V_1, V_2, \ldots, V_r\}$ de subconjuntos de U, encontrar (se existe) uma subfamília de $\mathcal{F}$ que seja uma partição de U.

Exemplo: Dados

$$\begin{array}{rcl} U & = & \{1,2,3,4,5\} \text{ e} \\ \mathcal{F} & = & \{\{1,2,3\},\{4,5\},\{1,2,4\},\{1,3,5\},\{2,3\},\{1\}\}, \end{array} \tag{3.1}$$

a subfamília $\{\{1\},\{2,3\},\{4,5\}\}$ é uma **cobertura exata** de U.

Teorema 3.1 *O problema* $\textsc{MinST}_d \in$ NP*-completo.*

Demonstração: Inicialmente, observamos que $\textsc{MinST}_d \in$ NP. De fato, dada uma árvore T de G, podemos verificar em tempo polinomial ($O(|E_G|)$) se T contém um caminho entre cada par de vértices terminais. Basta utilizar um algoritmo de busca em largura ou em profundidade. Também podemos verificar em tempo $O(|E_G|)$ se $c(T) \leq k$. Com isso, fica claro que $\textsc{MinST}_d \in$ NP.

Agora vamos mostrar que CE_d pode ser reduzido a $\textsc{MinST}_d$ em tempo polinomial. Sabe-se que o problema CE_d é NP-completo. Por isso, a redução polinomial é suficiente para demonstrar este teorema.

Dada uma instância $(U,\mathcal{F})$ do CE_d, com

$$U = \{u_1, u_2, \ldots, u_t\} \text{ e } \mathcal{F} = \{V_1, V_2, \ldots V_r\} .$$

construa a seguinte instância (G, c, R, k) do $\textsc{MinST}_d$:

$$\begin{array}{rcl} V_G & := & \{n_0\} \cup \{V_1, V_2, \ldots, V_r\} \cup \{u_1, u_2, \ldots, u_t\} \text{ e} \\ E_G & := & \{n_0 V_i : i = 1, \ldots, r\} \cup \{u_i V_j : u_i \in V_j, i = 1, \ldots, t, j = 1, \ldots, r\} \end{array}$$

$$c_e := \begin{cases} |V_i| & \text{se } e \text{ é do tipo } n_0 V_i \text{ para algum } i = 1, \ldots, r \\ t & \text{caso contrário} \end{cases}$$

$$R := \{u_1, u_2, \ldots, u_t, n_0\}$$

$$k := t^2 + t .$$

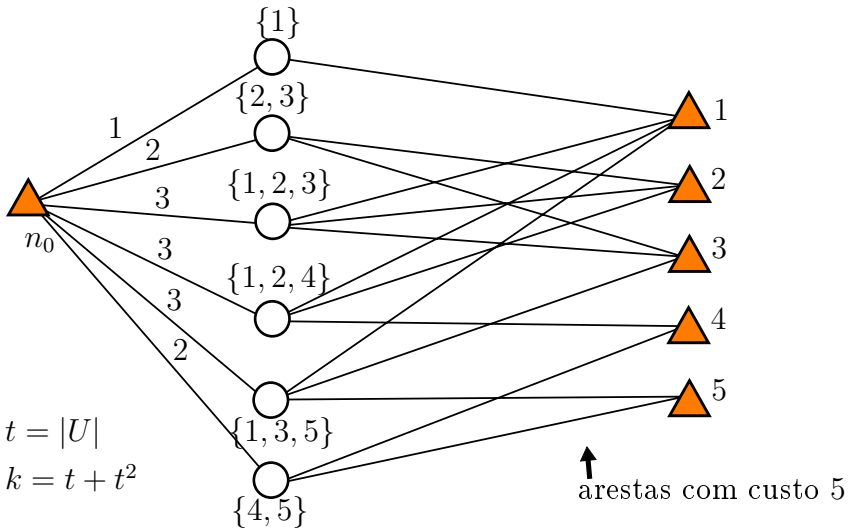


Figura 3.7: *Instância do* $\text{MinST}_d(G, c, R, k)$ *correspondente à instância* $\text{CE}_d(U, \mathcal{F})$ *dada no exemplo 3.1. Os vértices triângulos são terminais.*

Para mostrar a redução polinomial, provaremos que $\text{CE}_d(U, \mathcal{F})$ tem solução se e somente se $\text{MinST}_d(G, c, R, k)$ tem solução.

Primeiro, mostramos que se uma subfamília $\mathcal{L}$ de $\mathcal{F}$ é cobertura exata de U, então o subgrafo $G[I(\mathcal{L})]$ é solução de $\text{MinST}_d(G, c, R, k)$, onde

$$I(\mathcal{L}) = \{e \in E_G : e \text{ incide em algum vértice } V_i, \text{ que pertence a } \mathcal{L}\} .$$

Como $\mathcal{L}$ é cobertura exata de U, existe em $G[I(\mathcal{L})]$ exatamente uma aresta incidente a cada vértice correspondente a um elemento de U. Tais arestas têm custo total t^2. Como cada vértice correspondente a um membro de $\mathcal{L}$ está ligado a n_0, existe um caminho entre n_0 e cada vértice correspondente a um elemento de U, e portanto, $G[I(\mathcal{L})]$ contém um caminho entre cada par de vértices de R. Como $\mathcal{L}$ é cobertura exata de U, segue também que o custo total das arestas incidentes a n_0 em $G[I(\mathcal{L})]$ é t. Logo, $\sum_{e \in E_{G[I(\mathcal{L})]}} c_e = t^2 + t$, e portanto $G[I(\mathcal{L})]$ é solução de $\text{MinST}_d(G, c, R, k)$.

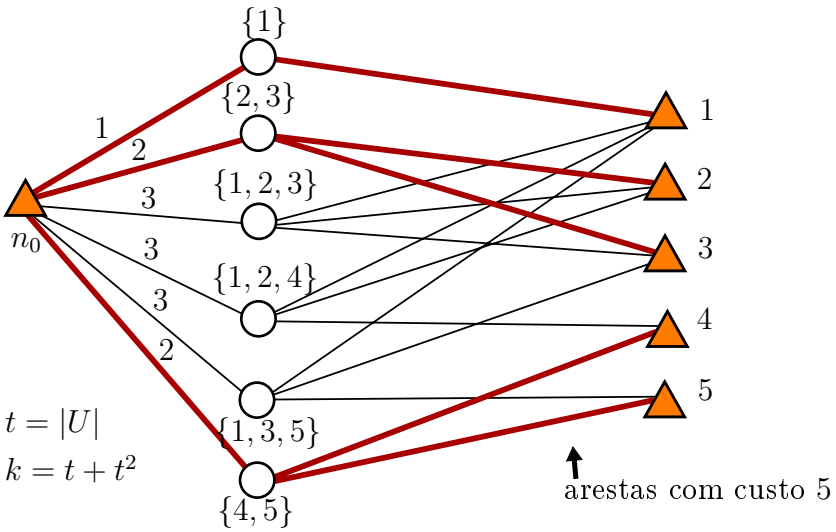


Figura 3.8: *As arestas mais espessas formam o subgrafo $G[I(\mathcal{L})]$ correspondente à cobertura exata $\mathcal{L} = \{\{1\}, \{2,3\}, \{4,5\}\}$ de U.*

Resta mostrar que, se T é uma solução de $\text{MinST}_d(G, c, R, k)$, então tem-se também uma solução de $\text{CE}_d(U, \mathcal{F})$.

Seja $\mathcal{L} := \{V_j : n_0 V j \in E_T\}$. Como n_0 e os vértices correspondentes a elementos de U são vértices terminais, existe em T um caminho entre n_0 e cada um desses vértices. Pela estrutura do grafo G, podemos concluir que $\mathcal{L}$ é uma cobertura de U e portanto $\sum_{V_j \in \mathcal{L}} |V_j| \geq t$.

Por outro lado, notemos que em cada vértice correspondente a um elemento de U deve incidir pelo menos uma aresta de T, pois tais vértices são terminais. Como tais arestas têm custo t e $|U| = t$, o custo total destas arestas é pelo menos t^2. Como $\sum_{e \in E_T} c_e \leq t^2 + t$ não podemos ter em T mais do que t arestas incidentes nos vértices correspondentes aos elementos de U, uma vez que n_0 é terminal. Com isso, o custo total dessas arestas é exatamente t^2.

Como

$$\sum_{e \in E_T} c_e \leq t^2 + t \ ,$$

então

$$\sum_{e \in \delta(\{n_0\})} c_e \leq t \ .$$

Assim,

$$t \ \geq \ \sum_{e \in \delta_T(\{n_0\})} c_e \ = \ \sum_{V_j \in \mathcal{L}} |V_j| \ \geq \ t \ .$$

Logo, $\mathcal{L}$ é cobertura exata de U, e portanto é uma solução de $\text{CE}_d(U, \mathcal{F})$.

■

3.2 Árvores de Steiner de árvores

Antes de explorar o problema MinST(G, c, R) pretendemos, nesta seção, considerar o caso particular em que o grafo de entrada G é uma árvore. Apesar deste caso ser elementar, gostaríamos de enfatizar aqui que muitos algoritmos para problemas de Steiner têm duas etapas distintas. Na primeira etapa, tipicamente a mais envolvente, o problema mais geral é reduzido para um no qual o grafo de entrada é uma árvore. Na segunda etapa, resolve-se de maneira ótima o problema restrito à árvore contruída na primeira etapa. O fator de aproximação do algoritmo resultante é, portanto, devido à primeira etapa.

Denominamos o algoritmo que resolve o problema MinST(G, c, R) para instância em que o grafo G é uma árvore de MinST-Poda. Para o algoritmo o custo c das arestas é irrelevante, já que c_e está em $\mathbb{Q}_{\geq}$ para cada e em E_G. Entretanto, não é difícil estender o algoritmo MinST-Poda para devolver árvores de Steiner de custo mínimo mesmo para grafos em que c_e é possivelmente negativo. Apesar de ser irrelevante, optamos por manter o parâmetro c por questão de uniformidade.

O algoritmo recebe uma árvore G e um subconjunto R de vértices terminais e basicamente remove todas as folhas que não são vértices terminais. A descrição do algoritmo supõe que R tem pelo menos 2 vértices. O nome "Poda" é devido a esse processo de "cortar folhas" da árvore.

> **Algoritmo MinST-Poda(G, c, R):** Recebe uma árvore G, um custo c_e em $\mathbb{Q}_{\geq}$ para cada aresta e em E_G e um conjunto R de vértices terminais com $|R| \geq 2$. Devolve uma árvore de Steiner T de custo mínimo.

O algoritmo é iterativo. No início de cada iteração temos uma subárvore T de G que contém todos os vértices terminais. No início da primeira iteração $T = G$.

Caso 1: Toda folha de T é um vértice terminal

Devolva T e pare.

Caso 2: Existe uma folha de T que não é vértice terminal

Escolha uma folha v de T que não é vértice terminal.

Comece uma nova iteração com $T-v$ no papel de T. ∎

A relação invariante fundamental mantida pelo algoritmo MinST-Poda é a seguinte. No início de cada iteração do algoritmo vale que:

(i1) T é uma árvore que contém todos os vértices terminais.

No início de cada iteração (i1) vale trivialmente. Como no início da última iteração todas as folhas de T estão em R então é evidente que a árvore devolvida é uma árvore de Steiner de custo mínimo.

3.3 Conjuntos ativos e coleções laminares

Se G é um grafo e R é um subconjunto de V_G, dizemos que um subconjunto S de V_G é **ativo** de (G, R) se

$$R \cap S \neq \emptyset \quad \text{e} \quad R \setminus S \neq \emptyset \,.$$

Então S é ativo se existe pelo menos um vértice terminal em S e um fora dele. A coleção de todos os subconjuntos ativos é denotada por $\mathcal{A}$. Aqui, mais uma vez, o conjunto R de vértices terminais está implícito.

É evidente que uma árvore T é de Steiner se e somente se $\delta_T(S) \neq \emptyset$ para todo S em $\mathcal{A}$, já que todo par de vértices terminais está conectado na árvore T.

Para o caso em que R contém apenas um vértice terminal, não há conjuntos ativos e a árvore de Steiner contém este único vértice. Não consideramos este caso na análise do problema e do algoritmo.

Uma coleção $\mathcal{L}$ de subconjuntos de V_G é dita **laminar** se, para quaisquer dois elementos L_1 e L_2 de $\mathcal{L}$, vale que $L_1 \cap L_2 = \emptyset$ ou $L_1 \subseteq L_2$ ou $L_1 \supseteq L_2$: ou os conjuntos são disjuntos, ou um está contido no outro. A coleção de elementos maximais de uma coleção laminar $\mathcal{L}$ é denotada por $\mathcal{L}^*$. Então, $\mathcal{L}^*$ é uma coleção de conjuntos disjuntos (figura 3.9).

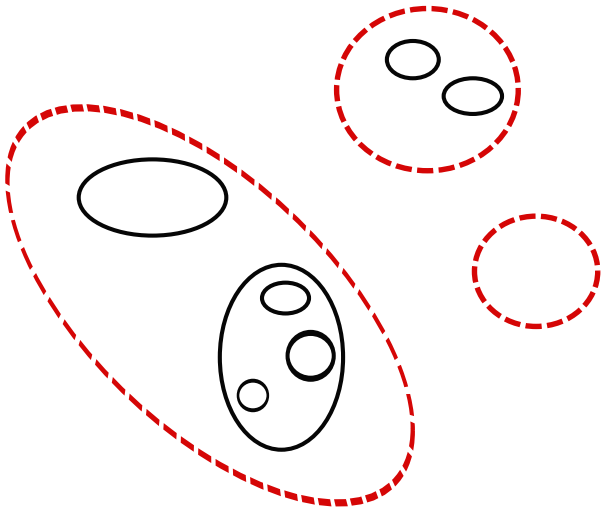


Figura 3.9: *Ilustração de uma coleção laminar $\mathcal{L}$. A coleção $\mathcal{L}^*$ de elementos maximais está representada com linhas tracejadas.*

Se $\mathcal{L}$ é uma coleção laminar de subconjuntos de V_G e X é um subconjunto de V_G, então adotamos a seguinte notação:

$$\mathcal{L}[X] := \{L \in \mathcal{L} : L \subseteq X\} \,.$$

Em palavras, $\mathcal{L}[X]$ é a subcoleção do elementos em $\mathcal{L}$ que estão contidos em X.

3.4 Programa linear primal e dual

Vários dos algoritmos que apresentamos neste texto se apoiam no **método primal-dual**. O algoritmo de aproximação MinST-GW para o MinST descrito neste capítulo é um exemplo. Nesta seção descrevemos o par de programas primal e dual nos quais o algoritmo MinST-GW se inspira. Começamos descrevendo uma relaxação linear para o MinST, em seguida deduzimos o correspondente problema dual.

O seguinte programa linear é uma relaxação de MinST(G, c, R): encontrar um vetor x indexado pelas arestas de G que

$$\begin{array}{rrcll} \text{minimize} & cx & & & \\ \text{sob as restrições} & x(\delta_G(S)) & \geq & 1 & \text{para cada } S \text{ em } \mathcal{A}\ , \\ & x & \geq & 0\ , & \end{array} \tag{3.2}$$

onde $x(\delta_G(S)) := \sum_{e \in \delta_G(S)} x_e$ e $\mathcal{A}$ é a coleção de subconjuntos ativos de (G, R).

Dada uma árvore de Steiner T, é evidente que o vetor característico de E_T é um candidato a solução de (3.2). Portanto, se x^* é uma solução de (3.2) então

$$cx^* \leq \text{opt}(G, c, R),$$

onde $\text{opt}(G, c, R)$ denota o valor da solução do problema MinST(G, c, R).

Seja A a matriz indexada por $\mathcal{A} \times E_G$ tal que, para cada conjunto S em $\mathcal{A}$ e para cada aresta e em E_G, temos que

$$A_{S,e} = \begin{cases} 1 & \text{se } e \in \delta_G(S) \\ 0 & \text{caso contrário.} \end{cases}$$

Com isso podemos reescrever o programa primal da seguinte maneira:

$$\begin{array}{rrcl} \text{minimize} & cx & & \\ \text{sob as restrições} & Ax & \geq & 1\ , \\ & x & \geq & 0\ . \end{array} \tag{3.3}$$

Por se tratar de um problema de minimização, é interessante buscar um limitante inferior para o seu valor ótimo. Utilizando a primeira restrição de (3.3), para qualquer vetor $y \geq 0$ indexado por $\mathcal{A}$, temos que

$$y(Ax) \geq y1 = \sum_{S \in \mathcal{A}} y_S = y(\mathcal{A})\ .$$

Se existir garantia de que vale $c \geq yA$, então temos o seguinte limitante inferior para o valor do primal:

$$\begin{array}{rll} cx \geq & (yA)x & \\ = & y(Ax) & \geq y(\mathcal{A})\ . \end{array} \tag{3.4}$$

Logo, $cx \geq y(\mathcal{A})$. Além disso, como queremos obter o melhor limitante possível, buscamos maximizar $y(\mathcal{A})$. Organizando estas condições temos:

$$\begin{array}{rrcl} \text{maximize} & y(\mathcal{A}) & & \\ \text{sob as restrições} & yA & \leq & c\ , \\ & y & \geq & 0\ . \end{array} \tag{3.5}$$

Mais detalhadamente, o dual do programa linear (3.2) consiste em encontrar um vetor y indexado pela coleção $\mathcal{A}$ de subconjuntos ativos de V_G que

$$\begin{array}{rlcll} \text{maximize} & y(\mathcal{A}) & & & \\ \text{sob as restrições} & y(\mathcal{A}(e)) & \leq & c_e & \text{para cada } e \text{ em } E_G\ , \\ & y & \geq & 0\ , & \end{array} \tag{3.6}$$

onde $\mathcal{A}(e) := \{S \in \mathcal{A} : e \in \delta_G(S)\}$.

3.5 Delimitação para o valor ótimo

Seja G um grafo, c uma função que associa um custo c_e em $\mathbb{Q}_{\geq}$ para cada aresta e em E_G e R um conjunto de vértices terminais. Dizemos que um vetor y **respeita** c se y é indexado pela coleção $\mathcal{A}$ de conjuntos ativos de (G, R) e é um candidato a solução do dual (3.6), ou seja, se y é um vetor não-negativo tal que

$$y(\mathcal{A}(e)) \leq c_e$$

para cada aresta e. Dizemos ainda que uma aresta f está **justa por** y se vale que

$$y(\mathcal{A}(f)) = c_f\ .$$

Se x^* é uma solução do primal, vale que $cx^* \leq \text{opt}(G, c, R)$, pois (3.2) é uma relaxação linear do problema MinST(G, c, R). Se y é um candidato a solução do dual, então $y(\mathcal{A}) \leq cx^*$, segundo o lema da dualidade da programação linear (seção 2.3). Portanto,

$$y(\mathcal{A}) \leq \text{opt}(G, c, R)\ . \tag{3.7}$$

Esta delimitação inferior para $\text{opt}(G, c, R)$ é fundamental para o cálculo do fator de aproximação do algoritmo MinST-GW descrito a seguir.

3.6 Algoritmo MinST-GW

O algoritmo MinST-GW para o problema MinST(G, c, R) que veremos nesta seção foi proposto por Michel Xavier Goemans e David Paul Williamson [GW95].

O algoritmo MinST-GW é composto de duas etapas. A primeira etapa é realizada pelo algoritmo MinST-Expansão que, utilizando o arcabouço do método primal-dual baseado no programa dual (3.2), encontra uma árvore de Steiner T_0 e um vetor dual viável y tais que toda aresta em T_0 está justa por y. Na segunda etapa é aplicado o algoritmo MinST-Poda sobre a árvore de Steiner T_0 e é devolvida uma árvore de Steiner T tal que

$$c(T) \leq 2\, y(\mathcal{A}) \leq 2\, \mathrm{opt}(G, c, R),$$

onde $\mathcal{A}$ é a coleção de conjuntos ativos de (G, R) e a segunda desigualdade é devida à delimitação (3.7). Isto mostra que o algoritmo MinST-GW, que está resumidamente descrito logo a seguir, é uma 2-aproximação polinomial. Este fato é completamente demonstrado na seção 3.7.

> **Algoritmo MinST-GW(G, c, R):** Recebe um grafo conexo G, um custo c_e em $\mathbb{Q}_{\geq}$ para cada aresta e em E_G e um conjunto R de vértices terminais com $|R| \geq 2$. Devolve uma árvore de Steiner T e um vetor y indexado por $\mathcal{A}$ que respeita c tais que $c(T) \leq 2y(\mathcal{A})$.

Passo 1: Sejam T_0 e y a árvore e o vetor obtidos pela execução do algoritmo MinST-Expansão(G, c, R).

Passo 2: Seja T a árvore obtida pela execução do algoritmo MinST-Poda(T_0, c, R). Devolva T e y e pare. ∎

Como vimos anteriormente o algoritmo MinST-Poda resolve o problema MinST(T_0, c, R). A etapa mais envolvente é a realizada pelo algoritmo MinST-Expansão a qual passamos a descrever.

O algoritmo MinST-Expansão é iterativo. No início de cada iteração tem-se uma floresta geradora F de G e um vetor y indexado por $\mathcal{A}$ que respeita c.

No início da primeira iteração, F não contém arestas, apenas os vértices em V_G e y é o vetor nulo. Em cada iteração, dizemos que um componente H de F é um **componente ativo** se V_H está em $\mathcal{A}$, e **inativo** caso contrário. Denotamos por $\mathcal{L}_F$ a coleção (laminar) formada pelos conjuntos de vértices que compuseram um componente de F em algum instante do algoritmo.

Notemos que $\mathcal{L}_F$ depende da floresta F, enquanto $\mathcal{A}$ depende apenas do grafo G e dos vértices em R.

Desta forma, $\mathcal{L}_F^* \cap \mathcal{A}$ é a coleção de conjuntos de vértices dos componentes de F ativos. Cada elemento S de $\mathcal{L}_F^* \cap \mathcal{A}$ viola a restrição $x(\delta_G(S)) \geq 1$ de (3.2), onde x é o vetor característico de F. Isso sugere que devemos acrescentar à F alguma das arestas de $\delta_G(S)$. Qualquer aresta deste tipo tem seus vértices extremos em elementos diferentes de $\mathcal{L}_F^*$. Dizemos que uma tal aresta é **externa** a $\mathcal{L}_F^*$. Devemos, então, escolher uma aresta externa a $\mathcal{L}_F^*$ e acrescentá-la à F. Esta escolha deve, é claro, estar relacionada ao custo das arestas.

Como o programa dual tem o objetivo de maximizar a soma das variáveis y_S, o algoritmo aumenta uniformemente os valores das variáveis y_S com S em $\mathcal{L}_F^* \cap \mathcal{A}$ de modo a manter viabilidade. Esse aumento gradativo de alguns componentes de y é interrompido quando alguma aresta fica justa por y. Essa aresta é então acrescentada à floresta F e uma nova iteração se inicia com a floresta atualizada.

O processo iterativo termina quando F não tem componentes ativos. Então, existe um componente T_0 de F que contém todos os vértices terminais, e cada um dos demais componentes é unitário.

> **Algoritmo MinST-Expansão(G, c, R):** Recebe um grafo conexo G, um custo c_e em $\mathbb{Q}_{\geq}$ para cada aresta e em E_G e um conjunto R de vértices terminais com $|R| \geq 2$. Devolve uma árvore de Steiner T_0 e um vetor y indexado por $\mathcal{A}$ tais que y respeita c e toda aresta em T_0 é justa por y.

O algoritmo é iterativo. Cada iteração começa com:

- um vetor y indexado por $\mathcal{A}$ que respeita c;
- uma floresta geradora F de G; e
- uma coleção laminar $\mathcal{L}_F$ de subconjuntos de V_G.

No início da primeira iteração temos que $y = 0$, $F = \emptyset$, e $\mathcal{L}_F = \{\{v\} : v \in V_G\}$.

Caso 1: $\mathcal{L}_F^* \cap \mathcal{A} = \emptyset$

> Seja T_0 o componente de F que contém todos os vértices terminais.
>
> Devolva T_0 e y e pare.

Caso 2: $\mathcal{L}_F^* \cap \mathcal{A} \neq \emptyset$

> Seja ε o maior número em $\mathbb{Q}_{\geq}$ tal que y' respeita c, onde y' é tal que $y'_S = y_S + \varepsilon$ se $S \in \mathcal{L}_F^* \cap \mathcal{A}$ e $y'_S = y_S$ caso contrário.
>
> Escolha uma aresta f justa por y e externa a $\mathcal{L}_F^*$.
>
> Sejam V_1 e V_2 os extremos de f em $\mathcal{L}_F^*$.
>
> Defina $F' := F + f$ e $\mathcal{L}'_F := \mathcal{L}_F \cup \{V_1 \cup V_2\}$.
>
> Comece uma nova iteração com $F+f$, y' e $\mathcal{L}'_F$ nos papéis de F, y e $\mathcal{L}_F$, respectivamente. ∎

O algoritmo devolve uma árvore de Steiner. De fato, no início de cada iteração de MinST-Expansão, F é uma floresta geradora de G. Ao final do algoritmo MinST-Expansão, F não tem componentes ativos, e portanto, todos os vértices terminais estão conectados. A árvore T_0 é o componente da floresta F que contém todos os vértices terminais, portanto é uma árvore de Steiner. Logo, T devolvida pelo algoritmo MinST-Poda é também uma árvore de Steiner.

A seção 3.8 ilustra a execução deste algoritmo.

Goemans e Williamson [GW95] mostraram que o algoritmo MinST-GW pode ser implementado de tal forma que o seu consumo de tempo seja $O(|V_G|^2 \log |V_G|)$.

Lema 3.2 ([GW95]) *O algoritmo* MinST-GW *admite uma implementação polinomial.* ∎

Um dos aspectos cruciais em uma implementação do algoritmo MinST-GW que seja eficiente é a representação do vetor dual viável y que respeita c. É evidente que uma implementação eficiente não pode manter todos os componentes de y, já que sua dimensão é proporcional a $2^{|V_G|}$. Para contornar este inconveniente é suficiente que sejam armazenados apenas os componentes não-nulos de y, ou seja, os valores de y indexados pelos elementos em $\mathcal{L}_F$. Como $\mathcal{L}_F$ é uma coleção laminar de subconjuntos de V_G temos que o número de elementos em $\mathcal{L}_F$ é não superior a $2|V_G| - 1$. Assim, estaríamos armazenando um número de componentes de y polinomial no tamanho do grafo.

3.7 Análise do algoritmo

Passamos agora a analisar o fator de aproximação do algoritmo MinST-GW. Porém, antes de delimitar o custo $c(T)$ da árvore devolvida pelo algoritmo, é preciso estabelecer uma relação fundamental entre T e a coleção $\mathcal{L}_F^* \cap \mathcal{A}$ dos componentes de F ativos em uma iteração arbitrária do algoritmo.

Lema 3.3 *No início de cada iteração do algoritmo* MinST-Expansão, *vale que*

$$\sum_{S \in \mathcal{L}_F^* \cap \mathcal{A}} |\delta_T(S)| \leq 2|\mathcal{L}_F^* \cap \mathcal{A}| \ ,$$

onde T é a árvore de Steiner que o algoritmo MinST-GW *devolve.*

Demonstração: Digamos que o **grau** em T de um componente S de F é o número de arestas em $\delta_T(S)$, ou seja, o grau em T de S é $|\delta_T(S)|$. Vale que, para qualquer componente S de F,

$$\text{se } |\delta_T(S)| = 1 \text{ então } S \in \mathcal{L}_F^* \cap \mathcal{A} \ . \tag{3.8}$$

A implicação (3.8) afirma que os componentes de F de grau 1 em T são todos ativos. Para provar essa afirmação, tome um componente S de F tal que $|\delta_T(S)|$ contém uma única aresta, digamos uw; ajuste a notação de modo que $u \in S$. Sejam U e W os componentes de $T - uw$ que contêm u e w respectivamente. Como uw é a única aresta de T em $\delta_T(S)$, temos $U \subseteq S$ e $W \subseteq V \setminus S$. Como T é uma árvore minimal, a remoção da aresta uw faz com que a floresta resultante tenha componentes ativos, ou seja, a aresta uw separa vértices terminais. Então,

$$R \subseteq U \cup W, \quad R \cap U \neq \emptyset \quad e \quad R \cap W \neq \emptyset \ .$$

Segue daí que $R \cap S \neq \emptyset$ e $R \setminus S \neq \emptyset$. Assim, S é um componente de F e está em $\mathcal{A}$ e portanto em $\mathcal{L}_F^* \cap \mathcal{A}$. Isso conclui a prova de (3.8).

Seja $\mathcal{Z}_F$ o conjunto de todos os componentes inativos de F cujo grau em T não é nulo, ou seja, todos os componentes inativos S tais que $|\delta_T(S)| \geq 1$. Digamos que dois elementos S e S' de $\mathcal{L}_F^*$ são **adjacentes** se existe uma aresta de T com um extremo em S e outro em S'. Esse conceito de adjacência define um grafo, digamos H, sobre o conjunto de vértices $\mathcal{L}_F^*$. Como T é uma árvore subgrafo de T_0, qualquer circuito em H corresponderia a um circuito em T_0, o que é impossível, já que T_0 é uma árvore. Portanto, H é uma árvore. Segue daí imediatamente que $\sum_{S \in V_H} |\delta_H(S)| = 2|E_H| \leq 2(|V_H| - 1)$. Como $V_H = \mathcal{L}_F^* = (\mathcal{L}_F^* \cap \mathcal{A}) \cup (\mathcal{Z}_F)$, temos que

$$\sum_{S\in\mathcal{L}_F^*} |\delta_T(S)| \leq 2(|\mathcal{L}_F^* \cap \mathcal{A}| + |\mathcal{Z}_F| - 1) \ .$$

Em virtude de (3.8), temos que $|\delta_T(S)| \geq 2$ para cada S em $\mathcal{Z}_F$. Logo $\sum_{S\in\mathcal{Z}_F} |\delta_T(S)| \geq 2|\mathcal{Z}_F|$, e portanto,

$$\begin{aligned}\sum_{S\in\mathcal{L}_F^*\cap\mathcal{A}} |\delta_T(S)| &= \sum_{S\in\mathcal{L}_F^*} |\delta_T(S)| - \sum_{S\in\mathcal{Z}_F} |\delta_T(S)| \\ &\leq 2(|\mathcal{L}_F^* \cap \mathcal{A}| + |\mathcal{Z}_F| - 1) - 2|\mathcal{Z}_F| \\ &\leq 2|\mathcal{L}_F^* \cap \mathcal{A}| \ .\end{aligned}$$

Isso conclui a demonstração do lema 3.3 ■

A interpretação dessa desigualdade é de que o grau médio dos vértices ativos do grafo H não é maior do que 2.

A seguir, mostramos como esse resultado garante o fator de 2-aproximação para a árvore construída pelo algoritmo.

Teorema 3.4 *O algoritmo* MinST-GW *é uma 2-aproximação para o problema* MinST.

Demonstração: Como já foi observado, o subgrafo T que o algoritmo MinST-GW devolve é uma árvore de Steiner. Provaremos agora que, no início de cada iteração de MinST-Expansão, vale a desigualdade

$$\sum_{S\in\mathcal{A}} |\delta_T(S)| y_S \leq 2y(\mathcal{A}) \ . \tag{3.9}$$

É evidente que a desigualdade é válida no início da primeira iteração de MinST-Expansão, quando $y = 0$. Suponha agora que a desigualdade seja válida no início de uma iteração qualquer. Durante a iteração, y_S é acrescido de ε se e somente se $S \in \mathcal{L}_F^* \cap \mathcal{A}$. Portanto, o lado esquerdo de (3.9) é acrescido de

$$\sum_{S\in\mathcal{L}_F^*\cap\mathcal{A}} |\delta_T(S)|\varepsilon$$

enquanto o lado direito é acrescido de $2|\mathcal{L}_F^* \cap \mathcal{A}|\varepsilon$. O lema 3.3 garante que o incremento do lado esquerdo não é maior que do lado direito. Portanto a desigualdade (3.9) vale no início da iteração seguinte de MinST-Expansão. Isso prova (3.9).

No início de cada iteração o vetor y respeita c. Além disso, vale também que toda aresta na árvore T está justa por y:

$$\sum_{S:e\in\delta_T(S)} y_S = c_e \qquad \text{para cada } e \in T \ .$$

Para mostrar que o algoritmo é uma 2-aproximação, resta verificar que $c(T) \leq 2\text{opt}(G, c, R)$. De fato, temos que

$$\begin{aligned} c(T) &= \sum_{e \in T} c_e \\ &= \sum_{e \in T} \sum_{S : e \in \delta_T(S)} y_S && (3.10) \\ &= \sum_{S \in \mathcal{A}} |\delta_T(S)| y_S && (3.11) \\ &\leq 2y(\mathcal{A}) && (3.12) \\ &\leq 2\text{opt}(G, c, R)\ . && (3.13) \end{aligned}$$

A igualdade (3.10) vale porque toda aresta na árvore T está justa por y. A linha (3.11) segue da anterior por meio de reorganização dos somatórios. A desigualdade (3.12) segue de (3.9). Finalmente, a desigualdade (3.13) é consequência da delimitação inferior (3.7).

■

3.8 Ilustração do algoritmo MinST-GW

Nesta seção ilustramos o funcionamento do algoritmo MinST-GW através da execução detalhada de um exemplo.

Como é comum nesse tipo de ilustrações, o grafo utilizado na execução é completo e é dado através do desenho no plano de seus vértices. Inicialmente, somente os vértices são exibidos e as arestas estão implícitas. O custo de cada aresta é a distância euclidiana entre as suas pontas no desenho. As únicas arestas que, à medida que a execução do algoritmo avança, são exibidas por linhas entre as suas pontas são aquelas que estão sendo inseridas na floresta F e correspondem a arestas justas por y. Os vértices do grafo são representados por triângulos e por pequenos discos. Os triângulos são os vértices terminais. A figura 3.10 mostra o grafo no início do algoritmo MinST-Expansão no passo 1 do algoritmo MinST-GW. Através da figura vemos que

$$\begin{aligned} V_G &= \{a, b, c, d, e, f\} \text{ e} \\ R &= \{a, c, d, e\}\,. \end{aligned}$$

Assim, no início da primeira iteração do algoritmo MinST-Expansão temos que

$$\begin{aligned} F &= \emptyset, \\ y &= 0, \\ \mathcal{L}_F &= \{\{a\}, \{b\}, \{c\}, \{d\}, \{e\}, \{f\}\} \quad \text{e} \\ \mathcal{L}_F^* &= \{\{a\}, \{b\}, \{c\}, \{d\}, \{e\}, \{f\}\}\,. \end{aligned}$$

Durante as iterações do algoritmo MinST-Expansão as faixas ou molduras se formam ao redor dos vértices, e, posteriormente, ao redor de aglomerados de círculos, indicando os componentes não-nulos de y que são formados por conjuntos ativos em $\mathcal{L}_F$. A largura de

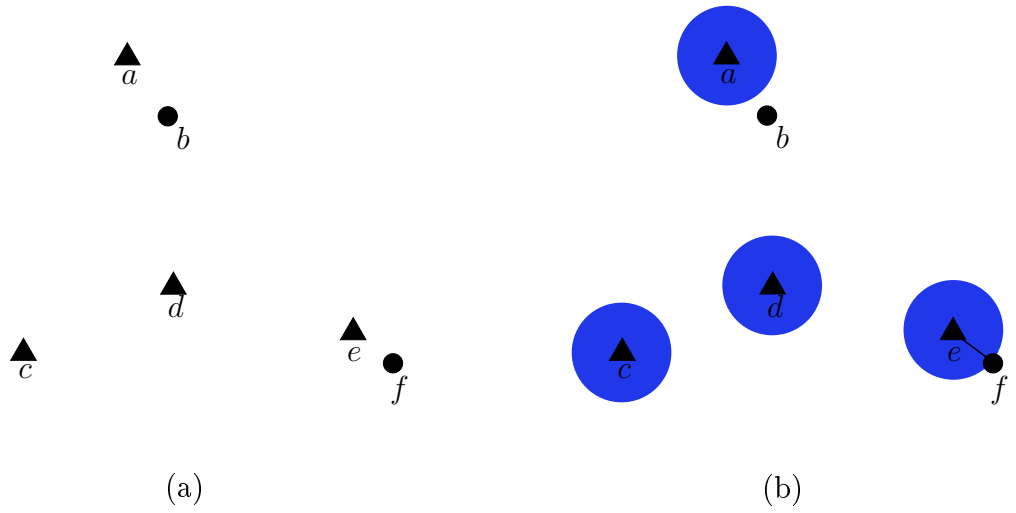


Figura 3.10: (a) *Instância "geométrica" do problema* MinST.
(b) *Situação no final da primeira e início da segunda iteração. Faixas ou molduras indicam os componentes não-nulos de y e a aresta ef torna-se justa por y.*

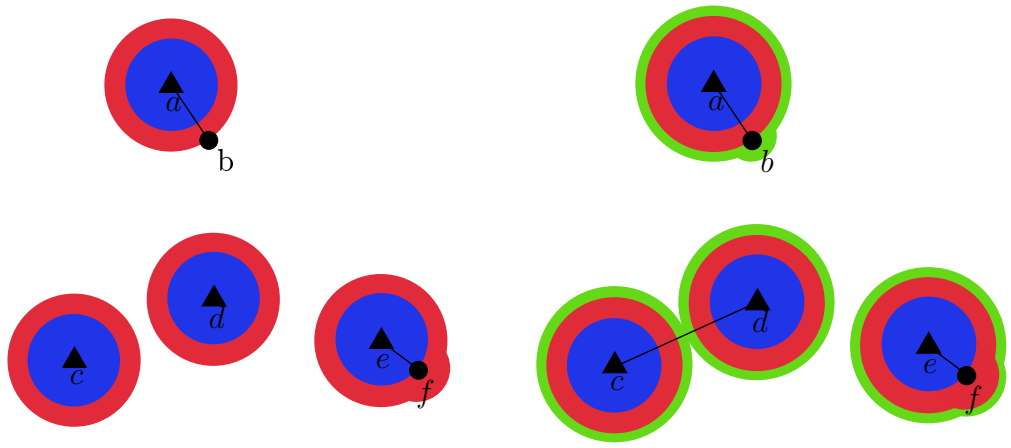
uma moldura representa o valor do componente de y formado pelos vértices no interior da moldura. A soma das larguras das molduras ao redor de um mesmo aglomerado é o valor do componente y formado pelos vértices no conglomerado.

No início da primeira iteração, temos que

$$\mathcal{L}_F^* \cap \mathcal{A} \ = \{\{a\}, \{c\}, \{d\}, \{e\}\},$$

onde $\mathcal{A}$ é a coleção dos conjuntos de vértices ativos. Desta forma, passamos ao caso 2 do algoritmo MinST-Expansão onde os componentes em $\mathcal{L}_F^* \cap \mathcal{A}$ de y serão acrescidos de ε. Na ilustração esses crescimentos uniformes de componentes de y são representados através do crescimento de molduras ao redor dos conjuntos em $\mathcal{L}_F^* \cap \mathcal{A}$. Na figura 3.10(b) vemos o resultado dessa primeira execução do caso 2. O crescimento dos componentes de y e das correspondentes molduras na figura 3.10(b) só pararam pois a aresta ef se tornou justa por y. Isto pode ser visto na figura, pois a distância entre os vértices e e f foi totalmente preenchida pelas molduras que envolvem o vértice e e o vértice f. Assim, a aresta ef é adicionada à floresta F e o conjunto $\{e\} \cup \{f\}$ é inserido na coleção $\mathcal{L}_F$. Com isto, no início da segunda iteração teremos

$$\begin{aligned} F &= \{ef\}, \\ \mathcal{L}_F &= \{\{a\}, \{b\}, \{c\}, \{d\}, \{e\}, \{f\}, \{e, f\}\}, \\ \mathcal{L}_F^* &= \{\{a\}, \{b\}, \{c\}, \{d\}, \{e, f\}\}. \end{aligned}$$



(c) (d)

Figura 3.11: (c) *Configuração ao final da segunda iteração do algoritmo. A aresta ab passa a fazer parte da floresta F.*
(d) *Na terceira iteração a aresta cd é incluída na floresta F. Como as duas pontas de cd pertenciam a conjuntos ativos, desta vez o número de conjuntos ativos em* $\mathcal{L}_F^*$ *diminui de um.*

Na segunda iteração do algoritmo MinST-Expansão,

$$\mathcal{L}_F^* \cap \mathcal{A} = \{\{a\}, \{c\}, \{d\}, \{e, f\}\},$$

e o caso 2 é mais uma vez executado. Depois do incremento uniforme dos componentes de y em $\mathcal{L}_F^* \cap \mathcal{A}$ a aresta ab se torna justa por y e ao final da iteração passamos a ter a configuração

$$\begin{aligned} F &= \{ab, ef\}, \\ \mathcal{L}_F &= \{\{a\}, \{b\}, \{c\}, \{d\}, \{e\}, \{f\}, \{a, b\}, \{e, f\}\}, \text{ e} \\ \mathcal{L}_F^* &= \{\{a, b\}, \{c\}, \{d\}, \{e, f\}\}. \end{aligned}$$

Esta nova execução do caso 2 encontra-se ilustrada na figura 3.11(c).

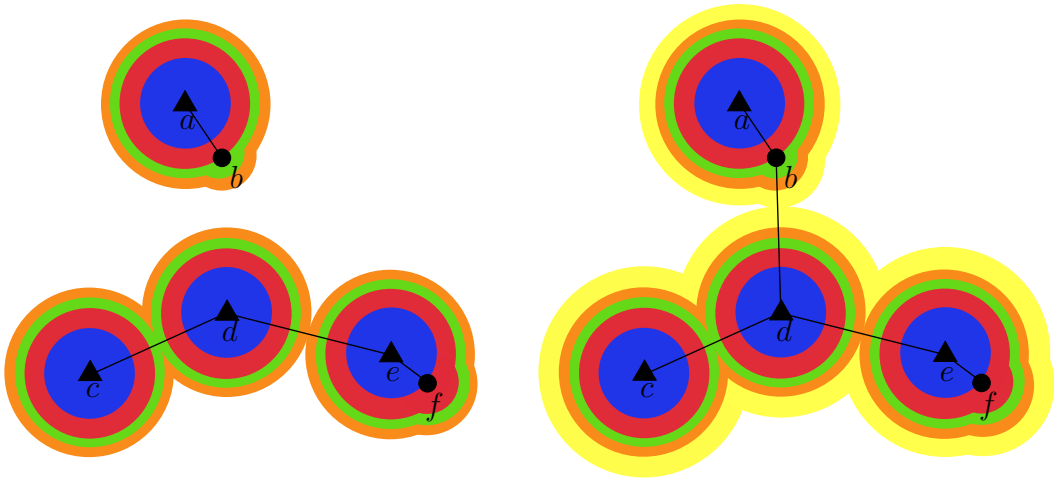
Em seguida, no início da terceira iteração temos que

$$\mathcal{L}_F^* \cap \mathcal{A} = \{\{a, b\}, \{c\}, \{d\}, \{e, f\}\},$$

e o caso 2 é novamente executado. Desta vez, a aresta cd se torna justa por y, ligando dois componentes ativos. Por isso, a cardinalidade da coleção de conjuntos ativos em $\mathcal{L}_F^*$ diminui. Neste momento temos que

$$\begin{aligned} F &= \{ab, cd, ef\}, \\ \mathcal{L}_F &= \{\{a\}, \{b\}, \{c\}, \{d\}, \{e\}, \{f\}, \{a, b\}, \{c, d\}, \{e, f\}\}, \\ \mathcal{L}_F^* &= \{\{a, b\}, \{c, d\}, \{e, f\}\}. \end{aligned}$$

A presente configuração se encontra ilustrada na figura 3.11(d).



(e) (f)

Figura 3.12: (e) *A aresta de fica justa na iteração quatro e mais uma vez o número de conjunto ativos em* $\mathcal{L}_F^* \cap \mathcal{A}$ *diminui.*
(f) *Desta vez, a aresta bd fica justa e é adicionada à floresta* F*, que fica com um único componente que não é ativo.*

Na quarta iteração do algoritmo o caso 2 é outra vez executado pois

$$\mathcal{L}_F^* \cap \mathcal{A} = \{\{a,b\},\{c,d\},\{e,f\}\}.$$

Mais uma vez dois conjuntos ativos são ligados já que a aresta de fica justa por y devido ao incremento dos valores dos componente y em $\mathcal{L}_F^* \cap \mathcal{A}$. Dessa forma, a nova configuração é

$$\begin{aligned} F &= \{ab, cd, de, ef\}, \\ \mathcal{L}_F &= \{\{a\},\{b\},\{c\},\{d\},\{e\},\{f\},\{a,b\},\{c,d\},\{e,f\},\{c,d,e,f\}\}, \\ \mathcal{L}_F^* &= \{\{a,b\},\{c,d,e,f\}\}. \end{aligned}$$

Essa configuração está ilustrada na figura 3.12(e).

Começando a quinta iteração vemos que

$$\mathcal{L}_F^* \cap \mathcal{A} = \{\{a,b\},\{c,d,e,f\}\}.$$

Agora é a vez da aresta bd ficar justa por y no caso 2 e ser inserida na floresta F que fica com um único componente que não é ativo, $\mathcal{L}_F^* \cap \mathcal{A} = \emptyset$. A configuração ao final desta iteração se encontra ilustrada na figura 3.12(f) e corresponde a

$$\begin{aligned} F &= \{ab, bd, cd, de, ef\}, \\ \mathcal{L}_F &= \{\{a\},\{b\},\{c\},\{d\},\{e\},\{f\},\{a,b\},\{c,d\},\{e,f\},\{c,d,e,f\},\{a,b,c,d,e,f\}\}, \\ \mathcal{L}_F^* &= \{\{a,b,c,d,e,f\}\}. \end{aligned}$$

Como já foi mencionado, no início da sexta e última iteração do algoritmo MinST-Expansão temos que nenhum dos conjuntos de vértices formados por componentes de F é um conjunto ativo, em símbolos $\mathcal{L}_F^* \cap \mathcal{A} = \emptyset$. Portanto, o caso 1 é executado e a única árvore T_0 da floresta F é a árvore de Steiner que é devolvida pelo algoritmo junto com o vetor y. Com isto se encerra o algoritmo MinST-Expansão e o passo 1 do algoritmo MinST-GW.

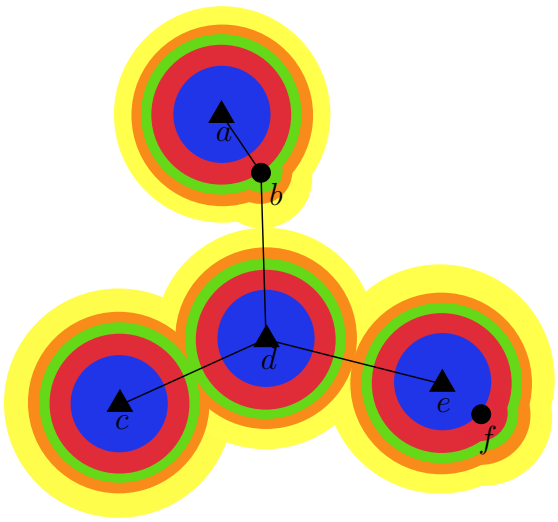


Figura 3.13: *Árvore de Steiner encontrada pelo algoritmo* MinST-GW.

No passo 2 do algoritmo MinST-GW o algoritmo MinST-Poda recebe a árvore T_0 e o conjunto R de vértices terminais. Como a ponta f da aresta ef é uma folha de T_0 que não é vertice terminal, o algoritmo remove ef da árvore e devolve a árvore $T = \{ab, bd, cd, de\}$ resultante para o algoritmo MinST-GW, já que todas as demais folhas são vértices terminais. A árvore T obtida no passo 2 pode ser vista na figura 3.13.

Finalmente, o algoritmo MinST-GW devolve a árvore de Steiner T e o vetor y e para. Como foi visto na seção anterior, o vetor y devolvido respeita o custo c das arestas e é um "certificado de qualidade" da árvore de Steiner, já que com y e T em mãos é possível limitar o quão longe o custo da árvore T está do custo de uma árvore de Steiner de custo mínimo (figura 3.14).

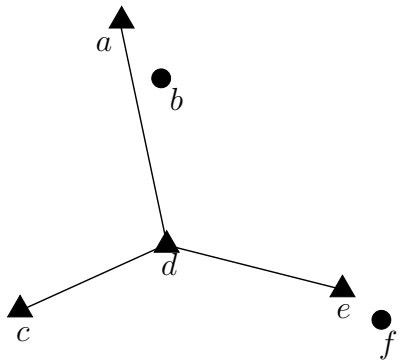


Figura 3.14: *Árvore de Steiner ótima para esta instância.*

No próximo capítulo consideraremos uma generalização natural do MinST e do algoritmo visto neste capítulo.

Capítulo 4

Árvore de Steiner com coleta de prêmios

Seja $G = (V_G, E_G)$ e c uma função custo de E_G em $\mathbb{Q}_{\geq}$. Na árvore de Steiner com coleta de prêmios trocamos os vértices terminais R da árvore de Steiner por uma função **penalidade** π de V_G em $\mathbb{Q}_{\geq}$. Assim, para cada vértice v passamos a ter uma penalidade π_v que funciona como uma certa generalização dos vértices terminais. O interesse, como no MINST é encontrar uma árvore "barata", mas desta vez é acrescentado ao custo das arestas da árvore a penalidade que deve ser "paga" por aqueles vértices que deixamos de conectar e não fazem parte da árvore construída.

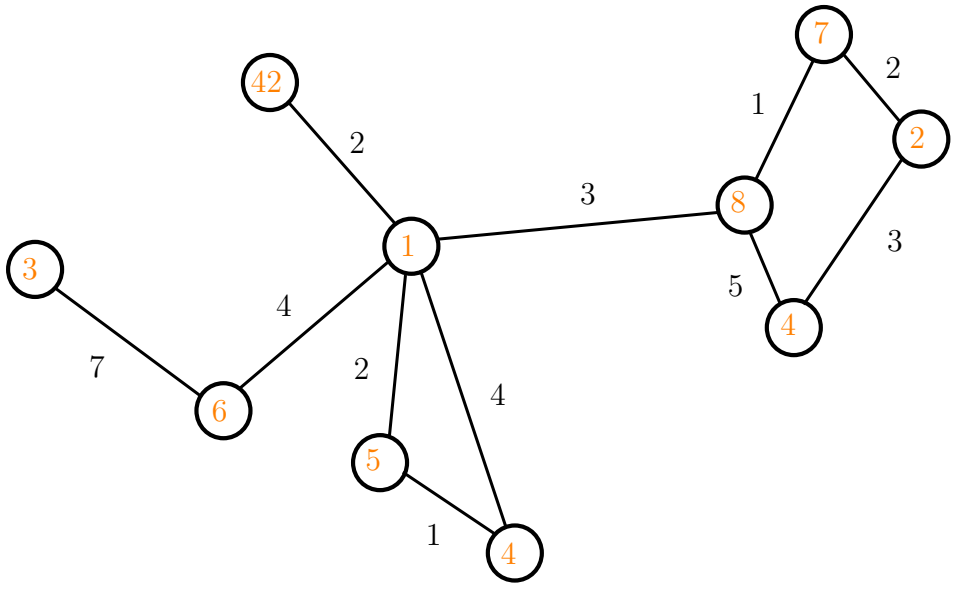


Figura 4.1: *Uma instância do* PCST*: um grafo, custo nas arestas e penalidades nos vértices.*

O **custo da árvore** T de G é $c(T) = c(E_T)$. No problema da árvore de Steiner com coleta de prêmios temos ainda uma **função penalidade** que associa a cada vértice v de G um valor racional não-negativo π_v. Se π é uma função penalidade de V_G em $\mathbb{Q}_{\geq}$, então a **penalidade paga** pelos vértices que não foram conectados por uma árvore T e $\pi(\overline{T}) = \pi(V_G \setminus V_T)$.

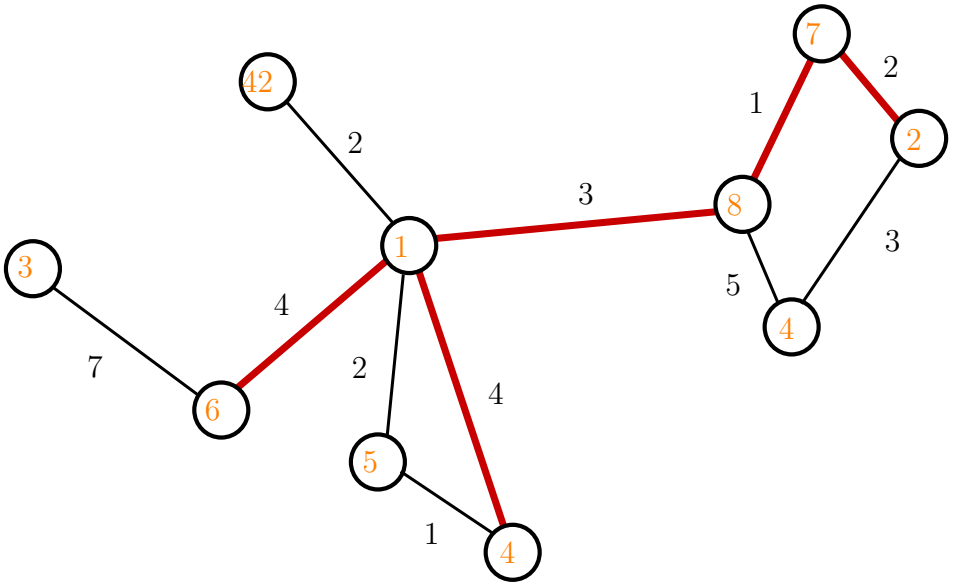


Figura 4.2: *O custo da árvore na figura é* $4+4+3+1+2=13$*, já a penalidade paga por essa árvore é* $3+42+5+4=54$*.*

> **Problema PCST(G, c, π):** Dados um grafo conexo G, um custo c_e em $\mathbb{Q}_{\geq}$ para cada aresta e em E_G e uma penalidade π_v em $\mathbb{Q}_{\geq}$ para cada vértice v em V_G, encontrar uma árvore T que minimize $c(T) + \pi(\overline{T})$.

Neste problema de otimização, dada uma árvore T candidata a solução, o valor de T é definido como $c(T) + \pi(\overline{T})$.

Notemos que, diferentemente do que ocorre com o MinST, no PCST toda árvore é uma candidata a solução. Por esta razão não seria necessário exigir conexidade do grafo dado. Entretanto, preferimos manter a conexidade do grafo da instância para manter a uniformidade e evitar casos irrelevantes e incômodos.

4.1 Complexidade computacional

O problema MinST pode ser visto como um caso especial do PCST. De fato, dada uma instância MinST(G, c, R) podemos transformá-la em uma instância PCST(G, c, π) com

$$\pi_v := \begin{cases} M, & \text{se } v \in R, \\ 0, & \text{caso contrário}, \end{cases}$$

onde M é um valor suficientemente grande como, por exemplo, $c(E_G) + 1$. Com esta definição da função π de penalidade, qualquer solução de PCST(G, c, π) é uma solução de MinST(G, c, R), e vice-versa. Assim, como o problema MinST é NP-difícil (seção 3.1) concluímos que o PCST também é NP-difícil.

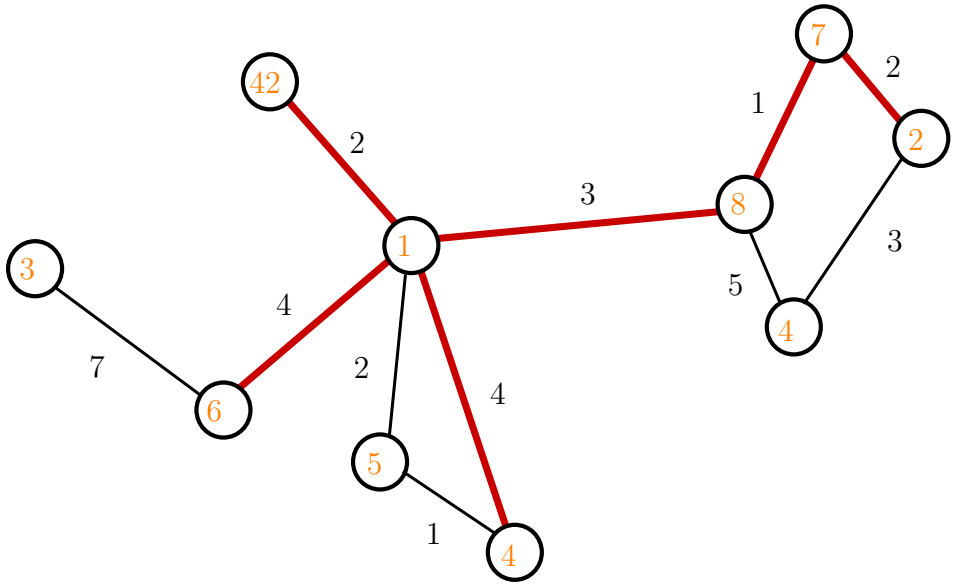


Figura 4.3: *Árvore de Steiner com coleta de prêmios com custo menor:* $c(T) + \pi(\overline{T}) = (4 + 4 + 3 + 1 + 2 + 2) + (3 + 5 + 4) = 28$.

Não é difícil perceber que essa redução **preserva o fator de aproximação**. Mais especificamente, um algoritmo que é uma α-aproximação para o PCST pode ser transformado em uma α-aproximação para o MinST utilizando-se a redução descrita. Isto é devido ao fato de que a função penalidade como definida garante que qualquer candidato a solução para o PCST que seja uma α-aproximação deve conter todos os vértices terminais, já que a penalidade paga pela não conexão de qualquer um desses vértices é essencialmente "infinito".

Uma versão deste problema, denominada **enraizada**, recebe também como parâmetro um vértice especial chamado de **raiz** r:

> **Problema R-PCST**(G, r, c, π)**:** Dados um grafo conexo G, um vértice r em V_G, um custo c_e em $\mathbb{Q}_{\geq}$ para cada aresta e em E_G e uma penalidade π_v em $\mathbb{Q}_{\geq}$ para cada vértice v em V_G, encontrar uma árvore T que contenha r e minimize $c(T) + \pi(\overline{T})$.

Pode-se reduzir uma instância do R-PCST para o PCST atribuindo penalidade grande ao vértice r. Dessa forma, o algoritmo não-enraizado é forçado a incluí-lo na árvore. Por outro lado, para reduzir uma instância do PCST para o R-PCST, basta executar o algoritmo para todas as possibilidades de raiz e, dentre todas as árvores obtidas, escolher aquela com menor custo. Essas reduções entre o PCST e o R-PCST preservam o fator de aproximação.

Assim, do ponto de vista de complexidade computacional, os problemas MinST, PCST e R-PCST são todos NP-difíceis e um algoritmo que é um α-aproximação para algum desses problemas pode ser utilizado como subrotina de uma α-aproximação para os demais. Dessa intratabilidade computacional vem o interesse em algoritmos de aproximação para estes problemas.

4.2 Árvore de Steiner com coleta de prêmios de árvores

Da mesma forma que foi feito no capítulo anterior, nesta seção mostraremos um algoritmo que resolve o PCST restrito a grafos que são árvores. Diferentemente do que ocorre com o MinST, o algoritmo que resolve o PCST restrito a árvores não é tão elementar e faz uso de programação dinâmica.

O algoritmo que veremos para o PCST é uma espécie de generalização do algoritmo MinST-GW para o MinST visto no capítulo anterior. Como o algoritmo MinST-GW, o algoritmo de aproximação para o PCST tem duas etapas. Na primeira etapa o problema mais geral é reduzido para um PCST no qual o grafo de entrada é uma árvore. Na segunda etapa, resolve-se o problema restrito à árvore contruída na primeira etapa. Curiosamente, não fará diferença, do ponto de vista de aproximação, resolver esse subproblema de maneira ótima, apesar disso ser possível utilizando-se o algoritmo JMP-Poda que passamos a descrever logo a seguir.

Em 2000, David Stifler Johnson, Maria Minkoff e Steven Phillips [JMP00], apresentaram um algoritmo que resolve o R-PCST restrito a árvores, com tempo de execução linear, através de programação dinâmica. A assimetria causada pela existência de um vértice especial raiz que deve fazer parte da solução do problema parece ser bastante conveniente. O algoritmo que descrevemos se apoia na **propriedade da subestrutura ótima** [CLRS01] que apresentamos a seguir.

No restante desta seção utilizamos a seguinte notação. Se um grafo H é uma árvore com raiz r, então é conveniente enxergar H como uma arborescência com raiz r. Desta forma, se u é um vértice de H, escrevemos H_u para nos referirmos à árvore em H com raiz u correspondente à subarborescência com raiz u (figura 4.4(a)). Além disso, também é conveniente considerar que a raiz r de H induz arborescências em florestas contidas em H. Desta forma, se F é uma floresta que é subgrafo de H, então F_u é a árvore em F com raiz u e que é subárvore de H_u (figura 4.4(b)).

Além disto, se (H, r, c, π) é uma instância do problema R-PCST então escrevemos (H_u, u, c, π) para nos referirmos à instância do R-PCST restrita à subárvore de H com raiz u, sem nos preocuparmos em restringir as funções c e π às arestas e aos vértices de H_u, respectivamente. Também escrevemos $\text{opt}(H, r, c, \pi)$ para nos referirmos ao valor de uma solução do problema R-PCST(H, r, c, π).

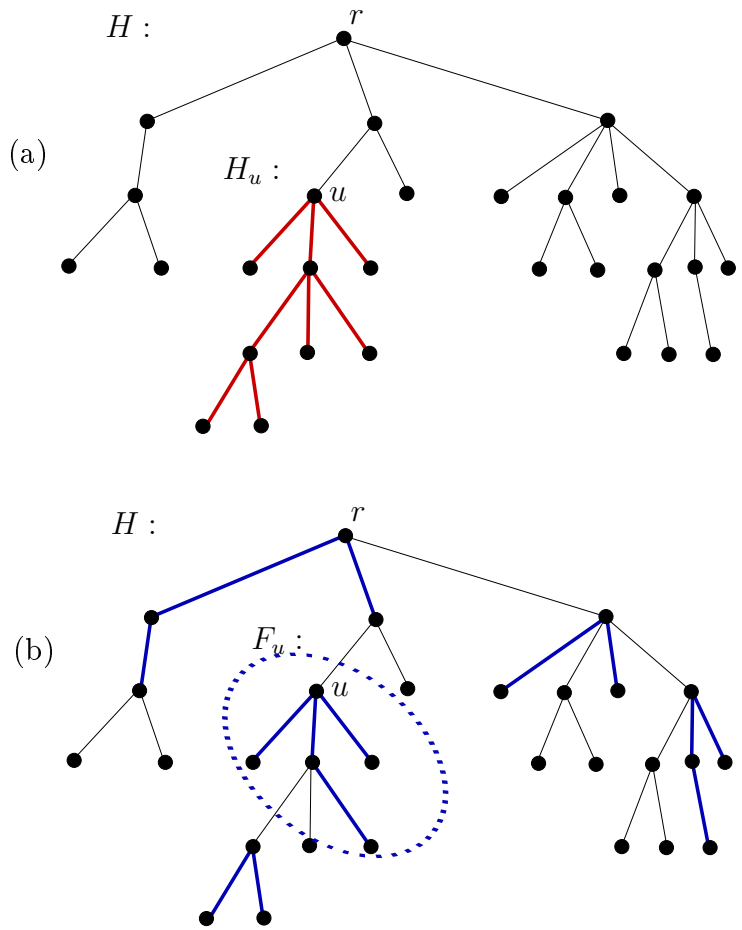


Figura 4.4: (a) *Se u é um vértice do grafo H, denotamos por H_u a arborescência com raiz em u (arestas mais espessas na figura).*
(b) *Se F é uma floresta que é um subgrafo de H (arestas mais espessas), F_u é a árvore de F com raiz em u: destacada pela circunferência pontilhada na figura.*

Teorema 4.1 (da subestrutura ótima) *Sejam (G, r, c, π) uma instância do problema* R-PCST *em que G é uma árvore e ru uma aresta de G. Se*

$$c_{ru} + \text{opt}(G_u, u, c, \pi) < \pi(V_{G_u}) \ , \tag{4.1}$$

então toda solução T de R-PCST(G, r, c, π) *tem ru como aresta e T_u é uma solução de* R-PCST(G_u, u, c, π). *Reciprocamente, se (4.1) não vale, então existe uma solução de* R-PCST(G, r, c, π) *que não tem ru como aresta. A figura 4.5 mostra um esquema deste teorema.*

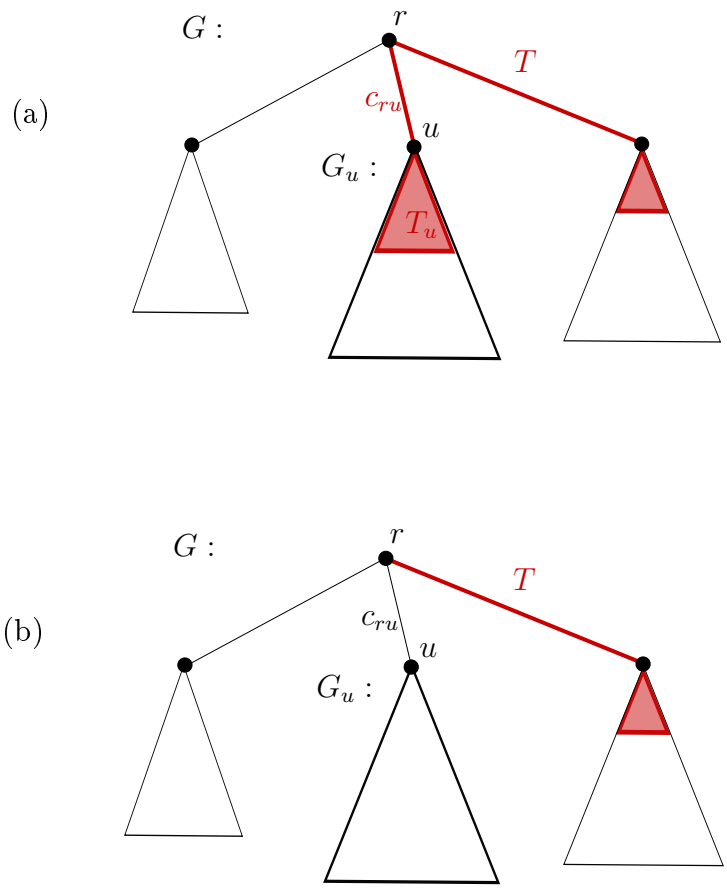


Figura 4.5: (a) *Os triângulos da figura representam as arborescências cujas raízes são adjacentes a r. O triângulo mais espesso representa a arborescência G_u. Se (4.1) vale, então toda solução T tem ru como aresta e T_u é solução da instância restrita à árvore G_u.*
(b) *Se (4.1) não vale, então existe uma solução T que não tem ru como aresta.*

Demonstração: Seja ru uma aresta para a qual (4.1) vale e seja T uma solução de R-PCST(G, r, c, π). Seja U uma solução de R-PCST(G_u, u, c, π).

Primeiro mostraremos que ru é uma aresta de T. Suponha que ru não é aresta de T e portanto T não contém vértices em G_u. Seja T' a árvore resultante da ligação das árvores T e U pela aresta ru.

Temos que

$$\begin{aligned}
c(T) + \pi(\overline{T}) &= c(T) + \pi(V_G \setminus V_T) \\
&= c(T) + \pi(V_G \setminus V_T) - \pi(V_{G_u}) + \pi(V_{G_u}) \\
&> c(T) + \pi(V_G \setminus V_T) - \pi(V_{G_u}) + c_{ru} + \text{opt}(G_u, u, c, \pi) \\
&= c(T) + c_{ru} + \text{opt}(G_u, u, c, \pi) - \pi(V_{G_u}) + \pi(V_G \setminus V_T) \\
&= c(T) + c_{ru} + c(U) + \pi(V_{G_u} \setminus V_U) - \pi(V_{G_u}) + \pi(V_G \setminus V_T) \\
&= c(T') - \pi(V_U) + \pi(V_G \setminus V_T) \\
&= c(T') + \pi(V_G \setminus V_{T'}) \\
&= c(T') + \pi(\overline{T'}) \ ,
\end{aligned}$$

onde a única desigualdade é devida a (4.1). Isto contradiz a hipótese de T ser uma solução de R-PCST(G, r, c, π) já que T' é uma solução mais "barata" que T.

Sabemos que ru é uma aresta de T. Assim, resta mostrar que T_u é solução de R-PCST(G_u, u, c, π). Seja R a subárvore de $T - ru$ que contém r. Logo $T = R + ru + T_u$. Seja $T' := R + ru + U$ e vale que

$$\begin{aligned}
c(T) + \pi(\overline{T}) &\leq c(T') + \pi(\overline{T'}) \\
&= c(R) + c_{ru} + c(U) + \pi(V_G \setminus (V_R \cup V_U)) \\
&= c(R) + c_{ru} + c(U) + \pi(V_G \setminus (V_R \cup V_{G_u})) + \pi(V_{G_u} \setminus V_U) \\
&= c(R) + c_{ru} + \pi(V_G \setminus (V_R \cup V_{G_u})) + c(U) + \pi(V_{G_u} \setminus V_U) \\
&\leq c(R) + c_{ru} + \pi(V_G \setminus (V_R \cup V_{G_u})) + c(T_u) + \pi(V_{G_u} \setminus V_{T_u}) \\
&= c(R) + c_{ru} + c(T_u) + \pi(V_G \setminus (V_R \cup V_{G_u})) + \pi(V_{G_u} \setminus V_{T_u}) \\
&= c(R) + c_{ru} + c(T_u) + \pi(V_G \setminus (V_R \cup V_{T_u})) \\
&= c(T) + \pi(V_G \setminus V_T) \\
&= c(T) + \pi(\overline{T}) \ ,
\end{aligned}$$

onde a primeira desigualdade é devido à hipótese de que T é solução de R-PCST(G, r, c, π) e a segunda é devido ao fato de U ser uma solução de R-PCST(G_u, u, c, π). Como o primeiro e último termos das desigualdades acima são os mesmos, então as duas desigualdades são igualdades e portanto T_u é solução de R-PCST(G_u, u, c, π).

Suponhamos agora que (4.1) não vale e seja T uma solução de R-PCST(G, r, c, π). Se ru não é aresta de T então não há o que demonstrar. Logo, podemos supor que ru é uma aresta de T. Seja T' a subárvore de $T - ru$ que contém r. Temos que

$$\begin{aligned}
c(T) + \pi(\overline{T}) &= c(T') + c_{ru} + c(T_u) + \pi(V_G \setminus (V_{T'} \cup V_{T_u})) \\
&= c(T') + \pi(V_G \setminus (V_{T'} \cup V_{G_u})) + c_{ru} + c(T_u) + \pi(V_{G_u} \setminus V_{T_u}) \\
&\geq c(T') + \pi(V_G \setminus (V_{T'} \cup V_{G_u})) + c_{ru} + \text{opt}(G_u, u, c, \pi) \\
&\geq c(T') + \pi(V_G \setminus (V_{T'} \cup V_{G_u})) + \pi(V_{G_u}) \\
&= c(T') + \pi(\overline{T'}) \ ,
\end{aligned}$$

onde a primeira desigualdade é devido ao fato de T_u ser um candidato a solução de R-PCST(G_u, u, c, π) e a segunda desigualdade vem da hipótese de que (4.1) não vale. Portanto, T' é uma solução de R-PCST(G, r, c, π) que não contém a aresta ru. Com isto concluímos a demosntração do teorema. ■

O teorema da subestrutura ótima 4.1 nos fornece um método recursivo para decidirmos quais arestas fazem parte de uma solução do problema R-PCST(G, r, c, π) em que G é uma árvore. O algoritmo recursivo JMP-PODA a seguir é uma aplicação imediata do teorema da subestrutura ótima.

Algoritmo JMP-PODA(G, r, c, π): Recebe uma árvore G, um vértice raiz r, um custo c_e em $\mathbb{Q}_{\geq}$ para cada aresta e em E_G e uma penalidade π_v em $\mathbb{Q}_{\geq}$ para cada vértice v em V_G. Devolve uma árvore T_r que contém r e que minimiza $c(T_r) + \pi(\overline{T_r})$.

Caso 1: $E_G = \emptyset$

Devolva a árvore formada apenas pelo vértice r e pare.

Caso 2: $E_G \neq \emptyset$

Seja U o conjunto dos vértice adjacentes a r.

Para cada vértice u em U, seja T_u a árvore devolvida pelo execução de JMP-PODA(G_u, u, c, π) e $\text{opt}_u = c(T_u) + \pi(V_{G_u} \setminus V_{T_u})$.

Para cada vértice u em U, seja $T'_u = T_u + ru$ se $c_{ru} + \text{opt}_u < \pi(V_{G_u})$; caso contrário seja T'_u a árvore formada apenas pelo vértice r.

Seja T_r a árvore que tem $\cup_{u \in U} V_{T'_u}$ como conjunto de vértices e $\cup_{u \in U} E_{T'_u}$ como conjunto de arestas.

Devolva T_r e pare. ∎

O algoritmo JMP-PODA realiza, essencialmente, uma busca em profundidade no grafo G. Logo, este algoritmo admite um implementação que consome tempo $O(|V_G| + |E_G|)$ para resolver o R-PCST restrito a árvores.

A seguir apresentamos uma versão iterativa do algoritmo JMP-PODA.

Algoritmo JMP-PODA-I(G, r, c, π): Recebe uma árvore G, um vértice raiz r, um custo c_e em $\mathbb{Q}_{\geq}$ para cada aresta e em E_G e uma penalidade π_v em $\mathbb{Q}_{\geq}$ para cada vértice v em V_G. Devolve uma árvore T_r que contém r e que minimiza $c(T_r) + \pi(\overline{T_r})$.

O algoritmo é iterativo. No início de cada iteração temos três florestas geradoras H, F e T de G. No início da primeira iteração temos que $E_H = E_G$, $E_F = E_T = \emptyset$.

Cada iteração consiste em:

Caso 1: $E_H = \emptyset$

Devolva T_r e pare.

Caso 2: $E_H \neq \emptyset$

Seja u uma folha de H, $u \neq r$.

Seja v o vértice em H adjacente a u.

Seja H' e F' as florestas geradoras de G com $E_{H'} = E_H - \{vu\}$ e $E_{F'} = E_F \cup \{vu\}$.

Seja $\text{opt}_u = c(T_u) + \pi(V_{G_u} \setminus V_{T_u})$.

Seja T' a floresta geradora com $E_{T'} = E_T \cup \{vu\}$ se $c_{vu} + \text{opt}_u < \pi(V_{G_u})$; caso contrário $E_{T'} = E_T$.

Comece uma nova iteração com H', F' e T' nos papéis de H, F e T, respectivamente. ∎

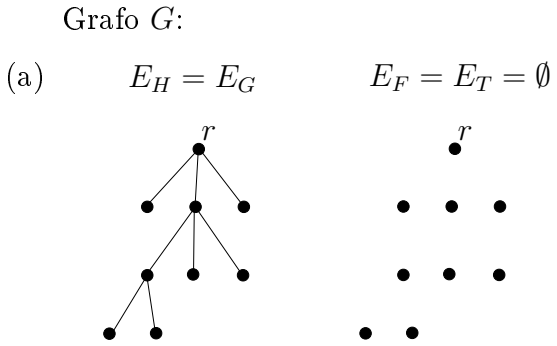


Figura 4.6: (a)*Execução do algoritmo* JMP-PODA-I(G, r, c, π).

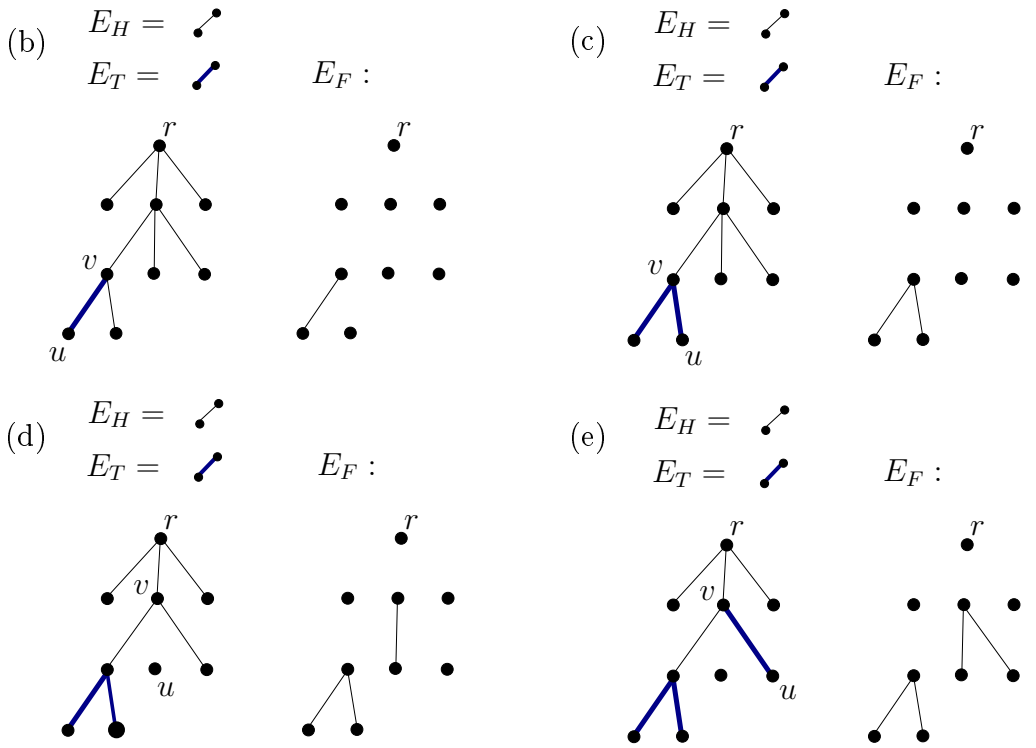


Figura 4.7: (b) *O vértice folha u de H é selecionado na primeira iteração do algoritmo. A aresta uv é removida de H e adicionada à floresta F. Suponha que vale a pena pagar o custo da aresta ao invés da penalidade do vértice u. Por isso a aresta uv é adicionada a floresta T.*
(c) *Na segunda iteração do algoritmo, a aresta uv também é adicionada à floresta T.*
(d) *Nesta iteração, suponha que o algoritmo decide pagar pela penalidade do vértice u ao invés de adicionar a aresta uv à floresta T. Por isso, a aresta uv simplesmente é removida de H e adicionada à F.*
(e) *A aresta uv é adicionada à floresta T.*

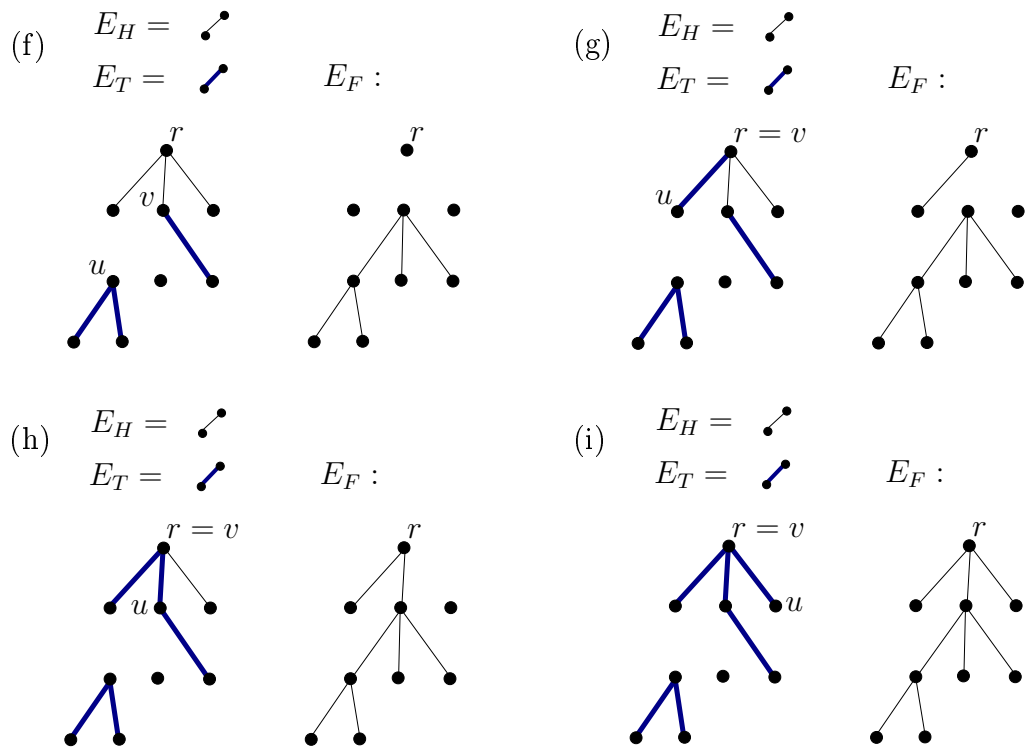


Figura 4.8: (f) *Nesta iteração, semelhante à decisão tomada na figura 4.7(d), o algoritmo escolhe pagar pelas penalidades da arborescência G_u ao invés de pagar pela aresta uv. Então, a aresta uv é removida de H e adicionada à F.*
(g), (h) *A aresta uv é adicionada à floresta T.*
(i) *A aresta uv também é adicionada à floresta T. Neste instante, $E_H = \emptyset$ e $F = G$.*

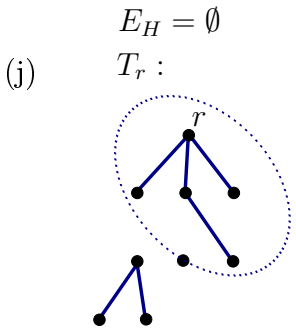


Figura 4.9: (j) *O algoritmo termina, pois $E_H = \emptyset$, devolvendo a árvore T_r.*

As relações invariantes fundamentais mantidas pelo algoritmo JMP-PODA-I são as seguintes. No início de cada iteração do algoritmo vale que:

(i0) $E_{H_r} = E_H$;

(i1) $E_H = E_G \setminus E_F$; e

(i2) T_u é solução de R-PCST(F_u, u, c, π) para todo vértice u em V_G.

A correção do algoritmo JMP-PODA-I segue facilmente dos invariantes (i1) e (i2). De fato, o algoritmo para já que em cada execução do caso 2 uma aresta é removida de H. Assim, como no início da última iteração temos que $E_H = \emptyset$, então, devido a (i1), temos que $E_G = E_F$. Dessa forma, ao final do algoritmo $F_r = G_r = G$ e por (i2) temos que a árvore T_r devolvida é solução de R-PCST(F_r, r, c, π) = R-PCST(G, r, c, π).

Demonstração de (i0): No início da primeira iteração temos que $E_H = E_G = E_{H_r}$ pois G é uma árvore. Suponhamos que no início de uma iteração que ocorre o caso 2 temos que $E_{H_r} = E_H$ e portanto H_r é uma árvore. Precisamos verificar que (i0) vale com H' no papel de H ao final da iteração. No caso 2 a floresta geradora H' de G é definida com tendo o conjunto de arestas $E_{H'} = E_H \setminus \{vu\}$ onde u é uma folha de H ou, equivalentemente, uma folha de H_r. Logo, ao final da iteração temos que $E_{H'_r} = E_{H'}$. ■

Demonstração de (i1): A validade da relação invariante (i1) é evidente. Ela vale no início da primeira iteração pois inicialmente $E_H = E_G$ e $E_F = \emptyset$ e em cada execução do caso 2 uma aresta (que é adjacente a uma folha de H) é removida de H e a mesma aresta é inserida em F. ■

Demonstração de (i2): No início da primeira iteração (i2) vale trivialmente pois $E_F = E_T = \emptyset$ e portanto T e F são florestas geradoras formadas apenas por vértices isolados.

Considere uma iteração qualquer em que ocorre o caso 2 e suponha que (i2) vale no início da iteração. Temos que verificar que (i2) vale no final dessa iteração com T' no papel de T e F' no papel de F. Temos que $E_{F'} = E_F \cup \{vu\}$ e $E_{T'} = E_T \cup \{vu\}$ ou $E_{T'} = E_T$. Como u é uma folha de H temos que, devido (i0) u é uma folha da árvore H_r e devido a (i1) v é o vértice raiz da árvore de F' que o contém. Isto significa que $F'_w = F_w$ e $T'_w = T_w$ para todo $w \neq v$. Desta forma, resta apenas verificarmos que ao final da iteração T'_v é solução de R-PCST(F'_v, v, c, π).

Como no início da iteração T_u é solução de R-PCST(F_u, u, c, π), então o valor opt_u definido no caso 2 é igual a $\text{opt}(F_u, u, c, \pi)$. Desta forma, o teste $c_{vu} + \text{opt}_u < \pi(V_{G_u})$ para decidir se a aresta vu será ou não incluída a T para formar T' nada mais é que a aplicação do teorema da subestrutura ótima para decidir se vu é aresta de R-PCST(G_v, v, c, π) e portanto T'_v é solução de R-PCST(F'_v, v, c, π). ■

Uma consequência imediata dessas relações invariantes é o seguinte teorema.

Teorema 4.2 (Estrutura da solução) *Se (G, r, c, π) é uma instância do problema* R-PCST *em que G é uma árvore, então existe uma floresta T tal que T_u é solução de* R-PCST(G_u, u, c, π) *para todo vértice u de G.*

Demonstração: Devido à relação invariante (i1) temos que, ao final do algoritmo JMP-PODA-I, $G = F$. Portanto, a relação invariante (i2), ao final do algoritmo, afirma que T é uma floresta tal que T_u é solução de R-PCST(F_u, u, c, π) = R-PCST(G_u, u, c, π). ■

As figuras 4.7 e 4.8 mostram a execução do algoritmo JMP-PODA-I(G, r, c, π) onde o grafo G é uma árvore, ilustrada em 4.6. Ao final do algoritmo, T (arestas espessas em 4.9) é uma floresta tal que T_u é solução de R-PCST(G_u, u, c, π) para todo vértice u de G, especialmente, para $u = r$, T_r é solução de R-PCST(G, r, c, π).

Para resolver o PCST restrito a árvores, podemos executar JMP-PODA para todas as possibilidades de raiz, devolvendo ao final, dentre as árvores encontradas durante o processo, a árvore T que minimiza $c(T) + \pi(\overline{T})$. O consumo de tempo desse algoritmo é, portanto, $|V_G|$ vezes o consumo de tempo da busca em profundidade, ou seja, o consumo de tempo é $O(|V_G|(|V_G| + |E_G|))$. Alternativamente, combinando o algoritmo JMP-PODA-I com a observação contida no teorema 4.3 a seguir podemos resolver o PCST(G, c, π) quando G é uma árvore com o mesmo consumo de tempo de apenas uma busca em profundidade. Para isto, basta alterarmos o caso 1 do algoritmo JMP-PODA-I para

Caso 1: $E_H = \emptyset$

Seja v o vértice de V_G para o qual

$$c(T_v) + \pi(V_G \setminus V_{T_v}) = \min\{c(T_w) + \pi(V_G \setminus V_{T_w}) : w \in V_G\}.$$

Devolva a árvore T_v e pare.

Teorema 4.3 *Seja G uma árvore com raiz r e (G, c, π) uma instância do problema* PCST. *Se T é uma floresta geradora de G tal que T_u é solução de* R-PCST(G_u, u, c, π) *para todo vértice u de G, então T_v é solução de* PCST(G, c, π) *para algum vértice v.*

Demonstração: Seja T^* uma solução do problema PCST(G, c, π) e seja v o vértice de T^* mais próximo de r. Para mostrarmos que T_v é solução de PCST(G, c, π) basta verificarmos que

$$c(T_v) + \pi(\overline{V_{T_v}}) \leq c(T^*) + \pi(\overline{V_{T^*}}).$$

Temos que

$$\begin{aligned} c(T_v) + \pi(\overline{V_{T_v}}) &= c(T_v) + \pi(V_G \setminus V_{T_v}) \\ &= c(T_v) + \pi(V_{G_v} \setminus V_{T_v}) + \pi(V_G \setminus V_{G_v}) \\ &\leq c(T^*) + \pi(V_{G_v} \setminus V_{T^*}) + \pi(V_G \setminus V_{G_v}) \\ &= c(T^*) + \pi(V_G \setminus V_{T^*}) \\ &= c(T^*) + \pi(\overline{V_{T^*}}), \end{aligned}$$

onde a desigualdade é devido ao fato de que T_v e T^* são uma solução e um candidato a solução de R-PCST(G_v, v, c, π), respectivamente. ∎

O teorema a seguir resume o que vimos na presente seção.

Teorema 4.4 *O problema* PCST(G, c, π) *restrito a árvores pode ser resolvido em tempo linear.* ∎

4.3 Arestas justas e conjuntos saturados

Para qualquer parte $\mathcal{X}$ de 2^{V_G} e qualquer aresta e, denotamos por $\mathcal{X}(e)$ a coleção dos elementos de $\mathcal{X}$ que têm e no seu corte, ou seja,

$$\mathcal{X}(e) := \{X \in \mathcal{X} : e \in \delta_G(X)\} \ .$$

Seja $\mathcal{A} := 2^{V_G}$ e seja y uma função de $\mathcal{A}$ em $\mathbb{Q}_{\geq}$. Dizemos que y **respeita** (em relação a $\mathcal{A}$) uma função custo c definida sobre E_G se

$$y(\mathcal{A}(e)) \leq c_e \tag{4.2}$$

para cada e em E_G, sendo $y(\mathcal{A}(e)) = \sum_{S \in \mathcal{A}(e)} y_S$. Uma aresta e está **justa por** y se (4.2) vale com igualdade.

Dizemos ainda que o vetor y **respeita** (em relação a $\mathcal{A}$) uma função penalidade π definida sobre V_G se

$$\sum_{S \subseteq X} y_S + \sum_{S \supseteq \overline{X}} y_S \leq \pi(X) \qquad \text{para cada } X \text{ em } \mathcal{A} \tag{4.3}$$

$$\text{e} \quad \sum_{S \subseteq \overline{X}} y_S + \sum_{S \supseteq X} y_S \leq \pi(\overline{X}) \qquad \text{para cada } X \text{ em } \mathcal{A} \ . \tag{4.4}$$

Um conjunto de vértices $X \in \mathcal{A}$ está **saturado por** y se (4.3) vale com igualdade. Se (4.4) vale com igualdade, dizemos que $\overline{X}$ está **saturado por** y. Usualmente, dizemos que a soma dos y_S para todo subconjunto S de X não deve exceder a soma das penalidades de todos os elementos em X. Em (4.3) adicionou-se a parcela da soma dos y_S dos conjuntos S que contêm o complemento de X. A desigualdade (4.4) é análoga a (4.3) para o complemento de cada subconjunto de vértices X em $\mathcal{A}$.

Uma aresta é **interna a** uma partição $\mathcal{P}$ de V_G se as suas extremidades pertencem ao mesmo elemento de $\mathcal{P}$. Arestas cujas extremidades pertencem a dois elementos diferentes de $\mathcal{P}$ são ditas **externas a** $\mathcal{P}$.

Para qualquer coleção laminar $\mathcal{L}$ (seção 3.3) de subconjuntos de V_G e uma árvore T de G, diz-se que T **tem uma ponte em** $\mathcal{L}$ se existe S em $\mathcal{L}$ tal que $|\delta_T(S)| = 1$.

4.4 Programa linear primal e dual

O algoritmo que veremos na próxima seção é uma aplicação do **método primal-dual**. Nesta seção descrevemos o par de programas primal e dual nos quais o algoritmo PCST-GW se inspira. Começamos descrevendo uma relaxação linear para o PCST(G, c, π), em seguida apresentamos o correspondente problema dual. Denotamos por $\mathcal{A}$ a coleção formada por todos os subconjuntos de vértices de G [1].

O seguinte programa linear é uma relaxação do PCST(G, c, π): encontrar um vetor x indexado por E_G e um vetor z indexado por $\mathcal{A}$ que

[1] No capítulo 3 denotamos por $\mathcal{A}$ a coleção dos subconjuntos de vértices ativos do MinST(G, c, R). Como, de certa forma, no PCST(G, c, π) todos os vértices são terminais, neste capítulo $\mathcal{A}$ é a coleção formada por todos os subconjuntos de vértices de G.

$$\begin{array}{rrcll} \text{minimize} & cx + \sum_{A\in\mathcal{A}} \pi(A) z_A & & & \\ \text{sob as restrições} & x(\delta_G(S)) + \sum_{A\supseteq S} z_A + \sum_{A\supseteq \overline{S}} z_A & \geq & 1 & \text{para cada } S \text{ em } \mathcal{A}\,, \\ & x & \geq & 0\,, & \\ & z & \geq & 0\,. & \end{array} \tag{4.5}$$

Para ver que o programa (4.5) é de fato uma relaxação para o $\text{PCST}(G, c, \pi)$ tomemos uma árvore T candidata a solução do problema. Seja x o seguinte vetor indexado por E_G

$$x_e := \begin{cases} 1, & \text{se } e \in E_T, \\ 0, & \text{caso contrário}\,, \end{cases}$$

e seja z o seguinte vetor indexado por $\mathcal{A}$

$$z_A := \begin{cases} 1, & \text{se } A = V_G \setminus V_T, \\ 0, & \text{caso contrário}\,. \end{cases}$$

Da maneira como foram definidos, x e z são vetores não-negativos e para todo conjunto S em $\mathcal{A}$ vale uma das seguintes afirmações:

- há uma aresta de T com uma ponta em S e outra fora de S, assim $x(\delta_G(S))$ é maior ou igual a 1;
- S está contido em $V_G \setminus V_T$ e nesse caso $\sum_{A\supseteq S} z_A$ é 1;
- $\overline{S}$ está contido em $V_G \setminus V_T$ e nesse caso $\sum_{A\supseteq \overline{S}} z_A$ é 1.

Logo, o par (x, z) é um candidato a solução do programa (4.5) de custo

$$\begin{aligned} cx + \sum_{A\in\mathcal{A}} \pi(A) z_A &= c(E_T) + \pi(V_G \setminus V_T) \\ &= c(T) + \pi(\overline{T}). \end{aligned}$$

Portanto, o programa linear (4.5) é de fato uma relaxação do $\text{PCST}(G, c, \pi)$.

O correspondente programa linear dual do programa (4.5) é: encontrar um vetor y indexado por $\mathcal{A}$ que

$$\begin{array}{rrcll} \text{maximize} & y(\mathcal{A}) & & & \\ \text{sob as restrições} & y(\mathcal{A}(e)) & \leq & c_e & \text{para cada } e \text{ em } E_G\,, \\ & \sum_{S\subseteq A} y_S + \sum_{S\supseteq \overline{A}} y_S & \leq & \pi(A) & \text{para cada } A \text{ em } \mathcal{A}\,, \\ & y & \geq & 0\,. & \end{array} \tag{4.6}$$

Esse programa linear dual é uma extensão do programa (3.6) visto na seção 3.4 para o MinST. Em palavras, um vetor y indexado por $\mathcal{A}$ é um candidato a solução do programa dual se y respeita as funções custo c e penalidade π em relação a $\mathcal{A}$.

4.5 Delimitação para o valor ótimo

O método de aproximação primal-dual é iterativo e no início de cada iteração temos um vetor dual viável y. No caso do PCST(G, c, π), cada iteração consiste na procura por uma árvore T cujo valor $c(T) + \pi(\overline{T})$ está "perto" do valor de uma solução. A medida de proximidade da solução é o fator de aproximação do algoritmo que, para ser determinado, necessita de um limitante inferior para a solução. O programa linear dual da seção anterior fornece esse limitante através do seguinte lema (da dualidade) especializado para o PCST.

Lema 4.5 (da dualidade) *Para toda árvore T de G e todo vetor y indexado por uma coleção laminar $\mathcal{L}$ de subconjuntos de V_G que respeita c e π em relação a $\mathcal{L}$ vale que*

$$y(\mathcal{L}) \leq c(T) + \pi(\overline{T}) \ .$$

Demonstração: Seja S um conjunto minimal de $\mathcal{L}$ que contém V_T. Se um conjunto como tal não existe, seja $S := V_G$. Consideremos a partição de $\mathcal{L}$ em três conjuntos:

$$\begin{aligned}
\mathcal{B} &:= \{L \in \mathcal{L} : \delta_T(L) \neq \emptyset\} \ , \\
\mathcal{C} &:= \{L \in \mathcal{L} : L \subseteq \overline{S} \text{ ou } L \supseteq S\} \ , \text{ e} \\
\mathcal{D} &:= \{L \in \mathcal{L} : L \subseteq S \setminus V_T\} \ .
\end{aligned}$$

Além disso, seja $\mathcal{D}^*$ a coleção dos elementos maximais em $\mathcal{D}$. Temos que

$$\begin{aligned}
y(\mathcal{B}) &= \sum_{L \in \mathcal{B}} y_L \leq \sum_{L \in \mathcal{B}} |\delta_T(L)| y_L = \sum_{e \in E_T} y(\mathcal{L}(e)) \\
&\leq \sum_{e \in E_T} c_e = c(T) \ , \\
y(\mathcal{C}) &= \sum_{L \in \mathcal{C}} y_L = \sum_{L \subseteq \overline{S}} y_L + \sum_{L \supseteq S} y_L \\
&\leq \pi(\overline{S}) \ , \\
y(\mathcal{D}) &= \sum_{L \in \mathcal{D}} y_L = \sum_{L \in \mathcal{D}^*} \sum_{X \subseteq L} y_X \leq \sum_{L \in \mathcal{D}^*} \pi(L) \\
&\leq \pi(S \setminus V_T) \ .
\end{aligned}$$

Portanto, utilizando estas três desigualdades obtemos que

$$\begin{aligned}
y(\mathcal{L}) &= y(\mathcal{B}) + y(\mathcal{C}) + y(\mathcal{D}) \\
&\leq c(T) + \pi(\overline{S}) + \pi(S \setminus V_T) \\
&= c(T) + \pi(\overline{T}) \ .
\end{aligned}$$

Com isto concluímos a demonstração do lema. ∎

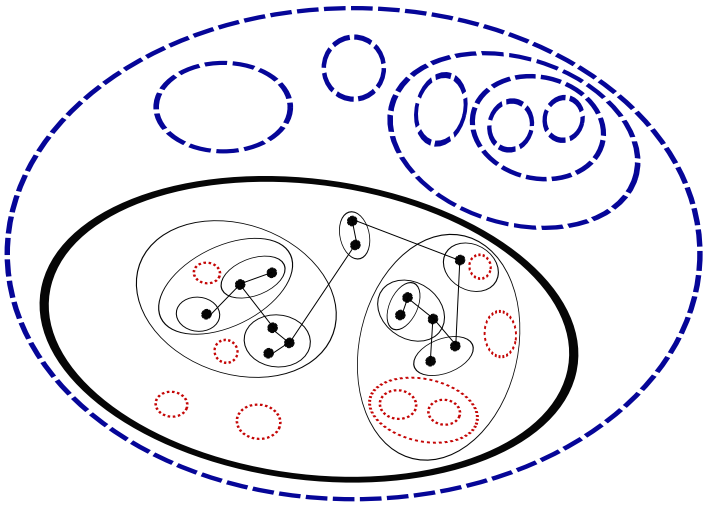


Figura 4.10: *Uma árvore T e uma coleção laminar. O conjunto com linha mais espessa é S. Conjuntos em linhas pontilhadas formam a coleção $\mathcal{D}$, as linhas tracejadas junto com o conjunto S formam $\mathcal{C}$ e os demais conjuntos formam a coleção $\mathcal{B}$.*

Uma consequência do lema 4.5 está logo a seguir.

Corolário 4.6 *Considere uma instância (G, c, π) do* PCST. *Dada uma coleção laminar $\mathcal{L}$ de subconjuntos de V_G e um vetor y sobre $\mathbb{Q}_{\geq}$ indexado por $\mathcal{L}$ que respeita c e π em relação a $\mathcal{L}$, vale que*

$$y(\mathcal{L}) \leq \text{opt}(G, c, \pi) \ ,$$

onde $\text{opt}(G, c, \pi)$ *é o valor de uma solução ótima para* $\text{PCST}(G, c, \pi)$.

Demonstração: Seja O uma árvore de G que é solução da instância $\text{PCST}(G, c, \pi)$. Pelo lema 4.5, vale que $y(\mathcal{L}) \leq c(O) + \pi(\overline{O})$

Como $c(O) + \pi(\overline{O}) = \text{opt}(G, c, \pi)$, concluímos que

$$y(\mathcal{L}) \leq \text{opt}(G, c, \pi) \ .$$

■

O limite inferior dado pelo colorário 4.6 é o já bem-conhecido argumento de que qualquer candidato a solução do programa linear dual (4.6) fornece um limite inferior para o valor de uma solução para o $\text{PCST}(G, c, \pi)$, como mencionamos no início desta seção. Entretanto, devemos notar que o vetor y do colorário é um vetor indexado por uma coleção laminar $\mathcal{L}$ de subconjuntos de V_G, enquanto os candidatos a solução do programa (4.6) são vetores indexados pela coleção $\mathcal{A}$ de todos os subconjunto de V_G. Não é difícil verificar que o vetor y que satisfaz as condições do colorário 4.6 corresponde a um candidado a solução do programa (4.6) como mostramos a seguir.

Proposição 4.7 *Considere uma instância (G, c, π) do* PCST. *Seja y um vetor sobre $\mathbb{Q}_{\geq}$ indexado pela coleção $\mathcal{A}$ de todos os subconjuntos de V_G. Suponha que os componentes não-nulos de y formam uma coleção laminar $\mathcal{L}$. Se y respeita c e π em relação a $\mathcal{L}$, então y respeita c e π em relação a $\mathcal{A}$.*

Demonstração: Para verificar que y respeita c em relação $\mathcal{A}$ basta notarmos que para toda aresta e vale que

$$\begin{aligned} y(\mathcal{A}(e)) &= y(\mathcal{L}(e)) + y(\mathcal{A}(e) \setminus \mathcal{L}(e)) \\ &= y(\mathcal{L}(e)) \\ &\leq c_e \ , \end{aligned} \tag{4.7}$$

onde a segunda igualdade é devida ao fato dos componentes não-nulos de y estarem em $\mathcal{L}$ e a desigualdade vem da hipótese de que y respeita c em relação a $\mathcal{L}$.

Passamos agora e verificar que y respeita π em relação a $\mathcal{A}$, isto é, y satisfaz (4.3) e (4.4). A demonstração é idêntica à do lema 4.5.

Seja A um subconjunto de vértices de G, com $A \neq \emptyset$ e $A \neq V_G$. Verificaremos primeiro que y satisfaz (4.3) com $\mathcal{A}$ no papel de $\mathcal{X}$:

$$\sum_{X \subseteq A} y_X + \sum_{X \supseteq \overline{A}} y_X \leq \pi(A) \ .$$

Seja S um conjunto minimal em $\mathcal{L}$ tal que S contém $\overline{A}$ ou $S = V_G$ e definamos as coleções

$$\begin{aligned} \mathcal{B}' &:= \{L \in \mathcal{L} : L \subseteq A \text{ ou } L \supseteq \overline{A}\} \text{ e} \\ \mathcal{C} &:= \{L \in \mathcal{L} : L \subseteq \overline{S} \text{ ou } L \supseteq S\} \text{ e} \\ \mathcal{D} &:= \{L \in \mathcal{L} : L \subseteq S \setminus \overline{A}\} \ . \end{aligned}$$

Como y respeita π em relação a $\mathcal{L}$ e S está em $\mathcal{L}$ temos que

$$\begin{aligned} \sum_{L \in \mathcal{C}} y_L &= \sum_{L \subseteq \overline{S}} y_L + \sum_{L \supseteq S} y_L \leq \pi(\overline{S}) \ , \text{ e} \\ \sum_{L \in \mathcal{D}} y_L &= \sum_{L \subseteq S \setminus \overline{A}} y_L = \sum_{L \in \mathcal{D}^*} \sum_{X \subseteq L} y_X \\ &\leq \sum_{L \in \mathcal{D}^*} \pi(L) = \pi(S \setminus \overline{A}) \ , \end{aligned}$$

onde por $\mathcal{D}^*$ denotamos a coleção dos elementos maximais de $\mathcal{D}$.

Como $\mathcal{L}$ é uma coleção laminar, então para todo L em $\mathcal{L}$ temos que

- se $L \subseteq A$ então $L \subseteq \overline{S}$ ou $L \subseteq S \setminus \overline{A}$ e, assim sendo, L está em $\mathcal{C}$ ou L está em $\mathcal{D}$;
- se $L \supseteq \overline{A}$ então $L \supseteq S$ e, portanto, L está em $\mathcal{C}$.

Desta forma concluímos que $\mathcal{C}$ e $\mathcal{D}$ formam uma partição de $\mathcal{B}'$ e que portanto vale que

$$\begin{aligned} \sum_{X \subseteq A} y_X + \sum_{X \supseteq \overline{A}} y_X &= \sum_{L \in \mathcal{B}'} y_L \\ &= \sum_{L \in \mathcal{C}} y_L + \sum_{L \in \mathcal{D}} y_L \\ &\leq \pi(\overline{S}) + \pi(S \setminus \overline{A}) = \pi(A) \ . \end{aligned}$$

A demonstração de que y satisfaz (4.4) com $\mathcal{A}$ no papel de $\mathcal{X}$ é simétrica. ■

A proposição 4.7 é importante para relacionar o comportamento do algoritmo (apresentado na seção a seguir) com o conceito do programa dual (apresentado na seção anterior) no seguinte sentido: o algoritmo mantém viabilidade dual com o vetor y respeitando custos e penalidades em relação à coleção laminar $\mathcal{L}_F$. Ao passo que o programa dual e o conceito de aresta justa e conjunto saturado são definidos em relação a $\mathcal{A}$. Esta proposição garante que o conceito "respeitar y" do algoritmo e da definição do programa dual são equivalentes.

4.6 Algoritmo PCST-GW

Nesta seção descrevemos o algoritmo PCST-GW que é uma aproximação polinomial para o PCST(G, c, π). Este algoritmo é uma modificação do algoritmo de Goemans e Williamson [GW95] para o R-PCST. Assim, ele é uma certa versão "não-enraizada" do algoritmo de Goemans e Williamson que se baseia no programa linear descrito na seção 4.4 e executa o arcabouço primal-dual apenas uma vez. A descrição que apresentamos é devida a Paulo Feofiloff, Cristina Gomes Fernandes, Carlos Eduardo Ferreira e José Coelho de Pina [FFFdP07].

O algoritmo PCST-GW é, de certa forma, uma extensão do algoritmo MinST-GW (seção 3.6).

Como antes, nesta seção denotamos por (G, c, π) uma instância do PCST e por $\mathcal{A}$ a coleção de todos os subconjuntos de vértices de G. O algoritmo consiste de duas etapas. A primeira etapa é realizada pelo algoritmo PCST-Expansão que recebe G, c e π e devolve quatro objetos:

- uma árvore T_o de G;
- um vetor y indexado por $\mathcal{A}$;
- uma coleção laminar $\mathcal{L}_F$ de subconjuntos de V_G; e
- uma subcoleção $\mathcal{Z}$ de $\mathcal{L}_F$.

Esses objetos são tais que

- y respeita c e π;
- toda aresta em T_0 é justa por y;
- $\mathcal{L}_F$ é o conjunto dos índices dos componentes não-nulos de y; e
- todo conjunto em $\mathcal{Z}$ é saturado por y.

Na segunda etapa é aplicado o algoritmo PCST-Poda sobre a árvore T_0 e a coleção $\mathcal{Z}$. O algoritmo PCST-Poda devolve uma subárvore T de T_0 que não possui pontes em $\mathcal{Z}$. A árvore T é devolvida pelo algoritmo PCST-GW junto com o vetor y. Na próxima seção demonstramos que a árvore T e o vetor y devolvidos são tais que

$$c(T) + \pi(\overline{T}) \leq (2 - \frac{2}{n})\, y(\mathcal{A}) = (2 - \frac{2}{n})\, y(\mathcal{L}_F) \leq (2 - \frac{2}{n})\, \mathrm{opt}(G, c, \pi)\ , \qquad (4.8)$$

onde a demonstração da primeira desigualdade é a missão central da próxima seção, a igualdade é devida à afirmação de que $\mathcal{L}_F$ é o conjunto de componentes não-nulos de y e a segunda desigualdade é devida ao corolário 4.6, já que o algoritmo promete devolver um vetor y que respeita c e π e cujos componentes não-nulos estão na coleção laminar $\mathcal{L}_F$. Supondo demonstradas as relações acima, concluímos que o algoritmo PCST-GW é uma $(2-\frac{2}{n})$-aproximação polinomial. A descrição do algoritmo PCST-GW está logo a seguir.

Algoritmo PCST-GW(G, c, π)**:** Recebe um grafo conexo G, um custo c_e em $\mathbb{Q}_{\geq}$ para cada aresta e em E_G e uma penalidade π_v em $\mathbb{Q}_{\geq}$ para cada vértice v em V_G. Devolve uma árvore T, uma coleção laminar $\mathcal{L}_F$ de subconjuntos de V_G e um vetor y indexado por $\mathcal{A}$ que respeita c e π e tais que $c(T)+\pi(\overline{T}) \leq (2-2/n)\, y(\mathcal{A})$.

Passo 1: Sejam T_0, y, $\mathcal{L}_F$ e $\mathcal{Z}$ a árvore, o vetor e as coleções laminares de subconjuntos de vértices obtidos pela execução do algoritmo PCST-Expansão(G, c, π).

Passo 2: Seja T a árvore obtida pela execução do algoritmo PCST-Poda$(T_0, \mathcal{Z})$.

Devolva T e y e pare. ∎

O algoritmo PCST-Expansão, descrito a seguir, é o componente central do arcabouço primal-dual do algoritmo PCST-GW. Este algoritmo pode ser visto como uma generalização do algoritmo MinST-Expansão (seção 3.6). De fato, com valores apropriados para π a execução do algoritmo PCST-Expansão(G, c, π) tem o mesmo comportamento e produz o mesmo resultado da execução do algoritmo MinST-Expansão(G, c, R). Para isto basta, como fizemos na seção 4.1, definirmos

$$\pi_v := \begin{cases} M, & \text{se } v \in R, \\ 0, & \text{caso contrário}. \end{cases}$$

Como todo algoritmo primal-dual, o PCST-Expansão utiliza de maneira fundamental as variáveis "duais" presentes no programa linear (4.6) para obter um candidato a solução do problema "primal", que no caso do PCST é uma árvore.[2] Assim, a principal relação invariante do algoritmo PCST-Expansão é a viabilidade dual. Mais especificamente, o algoritmo mantém ao longo de suas iterações

um vetor y indexado por $\mathcal{A}$ que respeita c e π .

Candidatos a solução do algoritmo são também mantidos pelo algoritmo PCST-Expansão representados por uma floresta geradora F de G formada por arestas justas em relação a y.

A fim de maximizar o valor da função objetivo $y(\mathcal{A})$ do programa dual (4.6), o algoritmo, de maneira idêntica ao algoritmo MinST-Expansão faz uso dos componentes ativos de F e incrementa uniformemente os valores de y correspondentes a esses componentes e uma floresta F vai sendo construída à medida que arestas ficam justas por y. Aqui, o conceito de componente ativo é também uma extensão do utilizado pelo algoritmo MinST-Expansão e depende do vetor y.

[2] Esse candidato a solução é posteriormente lapidado pelo algoritmo PCST-Poda como pode ser visto na descrição do algoritmo PCST-GW.

Dizemos que um conjunto de vértices A é **ativo** (em relação a y) se A não é saturado por y, em caso contrário dizemos que A é **inativo**. Denotamos por $\mathcal{L}_F$ a coleção (laminar) dos subconjuntos de vértices que constituiram um dos componentes de F no início de alguma iteração do algoritmo PCST-Expansão. Notemos que $\mathcal{L}_F$ depende da ordem que as arestas foram acrsecentadas à floresta F. Em cada iteração, $\mathcal{L}_F^*$ representa a coleção dos conjuntos maximais em $\mathcal{L}_F$. Em outras palavras, $\mathcal{L}_F^*$ é a coleção dos conjuntos de vértices de componentes de F.

Cada iteração do algoritmo começa com um vetor y viável dual e uma floresta geradora F de G. No início de cada iteração $\mathcal{L}_F^* \setminus \mathcal{Z}$ é a coleção de conjuntos de vértices dos componentes ativos de F. São estes os componentes de y incrementados em cada iteração. O processo iterativo termina quando F possui apenas um componente ativo, mas poderia, como faz o algoritmo MinST-Expansão, parar quando não há conjuntos ativos.

> **Algoritmo PCST-Expansão(G, c, π):** Recebe um grafo conexo G, um custo c_e em $\mathbb{Q}_{\geq}$ para cada aresta e em E_G e uma penalidade π_v em $\mathbb{Q}_{\geq}$ para cada vértice v em V_G. Devolve uma árvore T_0 de G, uma coleção laminar $\mathcal{L}_F$ de subconjuntos de V_G e um vetor y indexado por $\mathcal{A}$, uma subcoleção $\mathcal{Z}$ de $\mathcal{L}_F$ tais que y respeita c e π, toda aresta em T_0 é justa por y, $\mathcal{L}_F$ é o conjunto dos índices dos componentes não-nulos de y, e todo conjunto em $\mathcal{Z}$ é saturado por y.

O algoritmo é iterativo. Cada iteração começa com:

- um vetor y indexado por $\mathcal{A}$ que respeita c e π;
- uma floresta geradora F de G;
- uma coleção laminar $\mathcal{L}_F \subseteq \mathcal{A}$ tal que $\mathcal{L}_F^*$ é o conjunto de vértices dos componentes de F;
- uma subcoleção $\mathcal{Z}$ de $\mathcal{L}_F$ de conjuntos inativos; e
- uma subcoleção $\mathcal{M}$ de $\mathcal{L}_F$.

No início da primeira iteração temos que $y = 0$, F é tal que $V_F = V_G$ e $E_F = \emptyset$ (floresta geradora sem arestas), $\mathcal{L}_F = \{\{v\} : v \in V_G\}$ e $\mathcal{Z} = \mathcal{M} = \emptyset$.

Caso 1: $|\mathcal{L}_F^* \setminus \mathcal{Z}| = 1$ ou $\mathcal{M} \neq \emptyset$

Seja T_0 a árvore de F tal que $\mathcal{L}_F^* \setminus \mathcal{Z} = \{V_{T_0}\}$ ou $\mathcal{M} = \{V_{T_0}\}$.

Devolva T_0, y, $\mathcal{L}_F$ e $\mathcal{Z}$ e pare.

Caso 2: $|\mathcal{L}_F^* \setminus \mathcal{Z}| > 1$

Seja ε o maior número em $\mathbb{Q}_{\geq}$ tal que y' respeita c e π, onde y' é tal que $y'_S = y_S + \varepsilon$ se $S \in \mathcal{L}_F^* \setminus \mathcal{Z}$ e $y'_S = y_S$ caso contrário.

Caso 2A: Uma aresta f externa a $\mathcal{L}_F^*$ está justa por y'

Sejam V_1 e V_2 os conjuntos em $\mathcal{L}_F^*$ que contêm os extremos da aresta f.

Defina $F' := F + \{f\}$ e $\mathcal{L}'_F := \mathcal{L}_F \cup \{V_1 \cup V_2\}$.

Comece uma nova iteração com F', y' e $\mathcal{L}'_F$ nos papéis de F, y e $\mathcal{L}_F$.

Caso 2B: Um conjunto S em $\mathcal{L}_F^* \setminus \mathcal{Z}$ está saturado por y'

Defina $\mathcal{Z}' := \mathcal{Z} \cup \{S\}$.

Comece uma nova iteração com y' e $\mathcal{Z}'$ nos papéis de y e $\mathcal{Z}$.

Caso 2C: Para algum M em $\mathcal{L}_F^*$ o conjunto $\overline{M}$ está saturado por y'

Defina $\mathcal{M}' := \{M\}$.

Comece uma nova iteração com y' e $\mathcal{M}'$ nos papéis de y e $\mathcal{M}$. ∎

Na segunda etapa do algoritmo PCST-GW poderíamos utilizar o algoritmo JMP-PODA-I modificado que foi mencionado na seção 4.2, entretanto esta alteração não resultaria em um algoritmo com um fator de aproximação comprovadamente menor [FFFdP07]. A missão do algoritmo PCST-PODA é simplesmente devolver uma subárvore da árvore T_0 que não possui pontes na coleção $\mathcal{Z}$ dos conjuntos saturados por y em $\mathcal{L}_F$. O fato de a árvore devolvida pelo algoritmo PCST-GW não possuir pontes em conjuntos saturados por y é utilizado na demonstração do fator de aproximação do algoritmo.

Algoritmo PCST-PODA$(T_0, \mathcal{Z})$: Recebe uma árvore T_0 e uma coleção $\mathcal{Z}$ de conjuntos de vértices. Devolve uma árvore T de T_0 tal que T não tem nenhuma ponte em $\mathcal{Z}$.

Caso 1: T_0 não tem pontes em $\mathcal{Z}$

Seja $T = T_0$.

Devolva T e pare.

Caso 2: T_0 tem alguma ponte em $\mathcal{Z}$

Seja S em $\mathcal{Z}$ tal que $|\delta_{T_0}(S)| = 1$.

Comece uma nova iteração com $T_0 - S$ no papel de T_0. ∎

Goemans e Williamson [GW95] mostraram que o algoritmo PCST-GW pode ser implementado de tal forma que o seu consumo de tempo seja $O(|V_G|^2 \log |V_G|)$. Uma tal implementação foi descrita em detalhe por Feofiloff et al. em [FFFPJ02].

O principal detalhe de uma implementação eficiente é a manutenção de dois valores para cada conjunto L em $\mathcal{L}_F$ para armazenar a folga das desigualdades (4.3) e (4.4):

$$\Delta_1(L) := \pi(L) - \sum_{S \subseteq L} y_S + \sum_{S \supseteq \overline{L}} y_S$$

e

$$\Delta_2(L) := \pi(\overline{L}) - \sum_{S \subseteq \overline{L}} y_S + \sum_{S \supseteq L} y_S \ .$$

Assim como no algoritmo MINST-GW, é suficiente que sejam armazenados apenas os componentes não-nulos de y e os valores $\Delta_1(L)$ e $\Delta_2(L)$ para conjuntos L em $\mathcal{L}_F$. O tamanho de $\mathcal{L}_F$ é polinomial em $|V_G|$. O lema a seguir resume a presente discussão.

Lema 4.8 ([GW95]) *O algoritmo* PCST-GW *admite uma implementação polinomial.* ∎

4.7 Análise do algoritmo

O algoritmo PCST-Expansão é o componente central do algoritmo PCST-GW. Assim, a demonstração do fator de aproximação do algoritmo PCST-GW é fundamentada na verificação de uma série de relações invariantes do algoritmo PCST-Expansão. Cada iteração do algoritmo PCST-Expansão começa com um vetor y indexado por $\mathcal{A}$ que respeita c e π, uma floresta geradora F de G, uma coleção laminar $\mathcal{L}_F$ de subconjuntos de V_G, uma subcoleção $\mathcal{Z}$ de $\mathcal{L}_F$ de conjuntos inativos e uma subcoleção $\mathcal{M}$ de $\mathcal{L}_F$ com no máximo um elemento. No início de cada iteração valem as seguintes relações invariantes:

(i) $y_S = 0$ para todo S em $\mathcal{A} \setminus \mathcal{L}_F$;

(i0) todas as arestas de F são internas a $\mathcal{L}_F^*$;

(i1) F é conexa em cada elemento de $\mathcal{L}_F$;

(i2) y respeita c e π em relação a $\mathcal{L}_F$;

(i3) cada aresta de F é justa por y;

(i4) cada elemento de $\mathcal{Z}$ é saturado por y;

(i5) $\mathcal{M}$ tem no máximo um elemento e se $\mathcal{M} \neq \emptyset$ e $M \in \mathcal{M}$, então $M \in \mathcal{L}_F^*$ e $\overline{M}$ está saturado por y;

(i6) para qualquer árvore T de G, se T é conexa em cada elemento de $\mathcal{L}_F$ e não tem pontes em $\mathcal{Z}$, então vale que

$$\sum_{e \in E_T} y(\mathcal{L}_F(e)) + \sum_{S \subseteq \overline{V_T}} y_S + \sum_{S \supseteq V_T} y_S \leq \left(2 - \frac{2}{n}\right) y(\mathcal{L}_F) \ .$$

Uma outra relação invariante, muito semelhante a (i6), também é mantida pelo algoritmo. Esta relação, que batizamos de (i6'), é utilizada mais adiante, na seção 4.9.

(i6') para qualquer árvore T de G, se T é conexa em cada elemento de $\mathcal{L}_F$ e não tem pontes em $\mathcal{Z}$, então vale que

$$\sum_{e \in E_T} y(\mathcal{L}_F(e)) + 2 \sum_{S \subseteq \overline{V_T}} y_S + 2 \sum_{S \supseteq V_T} y_S \leq 2\, y(\mathcal{L}_F) \ .$$

Começamos com a verificação das relações invariantes (i0) a (i5).

Demonstração de (i) a (i5): É fácil verificar que cada uma dessas relações vale no começo da primeira iteração:

- F é tal que $V_F = V_G$ e $E_F = \emptyset$ e $\mathcal{L}_F = \{\{v\} : v \in V_G\}$, logo as relações (i0), (i1) e (i3) são satifeitas;
- $y = 0$ e como c e π são não-negativos, portanto (i) e (i2) são satisfeitas;
- $\mathcal{Z} = \mathcal{M} = \emptyset$ e assim (i4) e (i5) também são satisfeitas.

Consideremos agora uma iteração que não seja a última. Portanto estamos no caso 2 do algoritmo PCST-Expansão. A definição de ε e y feita no início do caso 2 garante que (i) e (i2) são satisfeitas no início da próxima iteração com y' no papel de y. Temos agora três possibilidades. Se o caso 2A ocorre, então f é uma aresta externa a $\mathcal{L}_F^*$ justa por y', $F' = F+f$ e $\mathcal{L}'_F = \mathcal{L}_F \cup \{V_1 \cup V_2\}$, onde V_1 e V_2 são elementos de $\mathcal{L}_F^*$ que contêm os extremos de f. Logo, no início da próxima iteração (i0), (i1) e (i3) são satisfeitas por F', y' e $\mathcal{L}'_F$ nos papéis de F, y e $\mathcal{L}_F$. Também é fácil verificar que, se o caso 2B ocorre, então a relação invariante (i4) é válida no início da próxima iteração com $\mathcal{Z}'$ no papel $\mathcal{Z}$ e y' no papel de y. Já, se o caso 2C ocorre, então a relação invariante (i5) é válida no início da próxima iteração com $\mathcal{M}'$ no papel de $\mathcal{M}$ e y' no papel de y. ■

Agora passamos à demonstração da relação invariante (i6). Esta é a relação mais envolvente e central na demonstração do fator de aproximação do algoritmo PCST-GW. De certa forma a relação invariante (i6) não é muito usual, pois envolve um objeto, uma árvore T genérica, que não está presente em cada iteração do algoritmo, pelo menos de maneira explícita.

Demonstração de (i6): No início da primeira iteração temos que y é o vetor nulo e, portanto, para toda árvore T temos que ambos os lados da desigualdade são iguais a zero e a relação invariante (i6) é satisfeita.

Agora suponhamos que a relação invariante (i6) vale no início de uma iteração que não seja a última. Assim, estamos no caso 2 do algoritmo.

Suponha que no caso 2 ocorre o caso 2A, ou seja, uma aresta f externa a $\mathcal{L}_F^*$ está justa por y'. Seja T uma árvore de G conexa em cada elemento de $\mathcal{L}'_F = \mathcal{L}_F \cup \{V_1 \cup V_2\}$, sem pontes em $\mathcal{Z}$. Devemos mostrar que (i6) vale com $\mathcal{L}'_F$ e y' nos papéis de $\mathcal{L}_F$ e y.

Como $\{V_1 \cup V_2\}$ não está em $\mathcal{L}_F$ temos que $y'_{V_1 \cup V_2} = 0$ e os índices dos componentes não-nulos de y' são os mesmos de y e estão todos em $\mathcal{L}_F$. Logo, basta verificar que no fim da iteração corrente

$$\sum_{e \in E_T} y'(\mathcal{L}_F(e)) + \sum_{S \subseteq \overline{V_T}} y'_S + \sum_{S \supseteq V_T} y'_S \leq \left(2 - \frac{2}{n}\right) y'(\mathcal{L}_F) \ . \tag{4.9}$$

Se o valor de ε calculado no início do caso 2 é zero, então (4.9) é verdade, pois $y = y'$ e (i6) vale no início da iteração. Logo, podemos supor que $\varepsilon > 0$. Seja $\mathcal{A}tivos := \mathcal{L}_F^* \setminus \mathcal{Z}$ e seja $z := y' - y$. Como (i6) vale no início da iteração, então para verificar (4.9) basta verificarmos que

$$\sum_{e \in E_T} z(\mathcal{L}_F(e)) + \sum_{S \subseteq \overline{V_T}} z_S + \sum_{S \supseteq V_T} z_S \leq \left(2 - \frac{2}{n}\right) z(\mathcal{L}_F) \ . \tag{4.10}$$

Por sua vez, $y' \geq y$ e y' difere de y apenas em componentes com índices em $\mathcal{A}tivos$. Desta forma, por definição, z é um vetor não-negativo que em que todos os componentes não-nulos

têm índices em $\mathcal{A}tivos$. Assim, dividindo ambos os lados de (4.10) por ε obtemos

$$\sum_{e\in E_T} |\mathcal{A}tivos(e)| + |\{S \in \mathcal{A}tivos : S \subseteq \overline{V_T}\}| + |\{S \in \mathcal{A}tivos : S \supseteq V_T\}| \leq \left(2-\frac{2}{n}\right)|\mathcal{A}tivos| \ . \tag{4.11}$$

Passamos agora à verificação da desigualdade anterior, cuja validade implica na desigualdade (4.9).

Seja $\mathcal{N} := \{S \in \mathcal{L}_F^* : \delta_T(S) = \emptyset\}$. Como a árvore T é conexa em cada elemento de $\mathcal{L}_F$, então

$$\mathcal{N} \cap \mathcal{A}tivos = \{S \in \mathcal{A}tivos : S \subseteq \overline{V_T}\} \cup \{S \in \mathcal{A}tivos : S \supseteq V_T\}.$$

Além disso, como $\sum_{e\in E_T} |\mathcal{A}tivos(e)| = \sum_{S\in\mathcal{A}tivos} |\delta_T(S)|$ e $(2-\frac{2}{n})|\mathcal{A}tivos| \geq 2|\mathcal{A}tivos| - 2$, então (4.11) seguirá de

$$\sum_{S\in\mathcal{A}tivos} |\delta_T(S)| + |\mathcal{N} \cap \mathcal{A}tivos| \leq 2|\mathcal{A}tivos| - 2 \ . \tag{4.12}$$

Se $\mathcal{A}tivos \subseteq \mathcal{N}$, então (4.12) vale pois

$$\sum_{S\in\mathcal{A}tivos} |\delta_T(S)| + |\mathcal{N} \cap \mathcal{A}tivos| = |\mathcal{A}tivos| \leq 2|\mathcal{A}tivos| - 2, \tag{4.13}$$

já que $|\mathcal{A}tivos| > 1$ dentro do caso 2 de PCST-Expansão.

Agora suponha que $\mathcal{A}tivos \not\subseteq \mathcal{N}$ e considere o grafo $H := (\mathcal{L}_F^*, E')$, onde E' é o conjunto de arestas de T externas a $\mathcal{L}_F^*$. Como T é conexo em cada elemento de $\mathcal{L}_F$, este grafo é uma floresta. Ele tem um componente não trivial e $|\mathcal{N}|$ componentes triviais. Então $|E'| = |\mathcal{L}_F^*| - 1 - |\mathcal{N}|$ e

$$\begin{aligned}
\sum_{S\in\mathcal{A}tivos} |\delta_T(S)| &= \sum_{S\in\mathcal{L}_F^*} |\delta_T(S)| - \sum_{S\in\mathcal{L}_F^*\cap\mathcal{Z}} |\delta_T(S)| \\
&= 2|E'| - \sum_{S\in\mathcal{L}_F^*\cap\mathcal{Z}} |\delta_T(S)| \\
&= 2|\mathcal{L}_F^*| - 2 - 2|\mathcal{N}| - \sum_{S\in\mathcal{L}_F^*\cap\mathcal{Z}} |\delta_T(S)| \\
&\leq 2|\mathcal{L}_F^*| - 2 - 2|\mathcal{N}| - 2|(\mathcal{L}_F^* \cap \mathcal{Z}) \setminus \mathcal{N}| \\
&= 2|\mathcal{L}_F^*| - 2 - 2|\mathcal{N}| - 2|\mathcal{L}_F^* \cap \mathcal{Z}| + 2|\mathcal{N} \cap \mathcal{Z}| \\
&= 2|\mathcal{L}_F^*| - 2|\mathcal{L}_F^* \cap \mathcal{Z}| - 2 - 2|\mathcal{N}| + 2|\mathcal{N} \cap \mathcal{Z}| \\
&= 2|\mathcal{L}_F^* \setminus \mathcal{Z}| - 2 - 2|\mathcal{N} \setminus \mathcal{Z}| \\
&= 2|\mathcal{A}tivos| - 2 - 2|\mathcal{N} \cap \mathcal{A}tivos| \ ,
\end{aligned} \tag{4.14}$$

onde a desigualdade (4.14) vale pois T não tem pontes em $\mathcal{Z}$.

Pode-se aplicar uma demonstração muito semelhante a esta para os casos (2B) e (2C) do algoritmo. ■

Demonstração de (i6'): A demonstração de (i6') é idêntica a de (i6), o que não é nenhuma surpresa tendo em vista a semelhança das duas relações. Descrevemos aqui apenas as adaptações que devem ser feitas à demonstração da relação (i6) a fim de obtermos a relação (i6'). Em primeiro lugar, na demonstração de (i6) todos os valores $2 - 2/n$ devem ser substituídos por 2. Em vez de demonstrarmos (4.12), devemos verificar que

$$\sum_{S \in \mathcal{A}tivos} |\delta_T(S)| + 2|\mathcal{N} \cap \mathcal{A}tivos| \leq 2|\mathcal{A}tivos| \ . \tag{4.15}$$

Assim, em vez de (4.13) temos que se $\mathcal{A}tivos \subseteq \mathcal{N}$, então (4.15) vale pois

$$\sum_{S \in \mathcal{A}tivos} |\delta_T(S)| + 2\,|\mathcal{N} \cap \mathcal{A}tivos| = |\mathcal{A}tivos| \leq 2|\mathcal{A}tivos| \,. \tag{4.16}$$

A desigualdade que é demonstrada em seguida, a saber,

$$\sum_{S \in \mathcal{A}tivos} |\delta_T(S)| = 2|\mathcal{A}tivos| - 2 - 2|\mathcal{N} \cap \mathcal{A}tivos| \ ,$$

já nos apresenta o resultado desejado e não precisa ser adaptada. ■

Tendo provado as relações invariantes de (i0) até (i6) e (i6'), temos ferramentas suficientes para apresentarmos o fator de aproximação do algoritmo PCST-GW.

Teorema 4.9 (do fator de aproximação [FFFdP07]) *Se T é uma árvore devolvida pelo algoritmo* PCST-GW*(G, c, π), então*

$$c(T) + \pi(\overline{T}) \leq (2 - 2/n)\,\mathrm{opt}(G, c, \pi) \ \ e \tag{4.17}$$

$$c(T) + 2\,\pi(\overline{T}) \leq 2\,\mathrm{opt}(G, c, \pi) \ . \tag{4.18}$$

Demonstração: Seja T_0 a árvore devolvida por PCST-Expansão(G, c, π) e seja $T \subseteq T_0$ a árvore devolvida por PCST-GW(G, c, π). Segue da relação invariante (i3) que toda aresta de T está justa por y, logo

$$c(T) = \ \sum_{e \in E_T} c_e \ = \sum_{e \in E_T} y(\mathcal{L}_F(e)) \ . \tag{4.19}$$

No começo da última iteração de PCST-Expansão, seja $\mathcal{X}$ a coleção de conjuntos de vértices definida da seguinte maneira:

$$\mathcal{X} := \begin{cases} \{\overline{M}\} & \text{se o algoritmo passou pelo caso 2C ,} \\ \mathcal{L}_F^* \cap \mathcal{Z} & \text{caso contrário .} \end{cases}$$

Em palavras, $\mathcal{X}$ é uma coleção de conjuntos disjuntos em que todo elemento está saturado

por y, segundo os invariantes (i4) e (i5). Portanto,

$$\begin{aligned}
\pi(\overline{T}) &= \sum_{v \in \overline{V_T}} \pi_v \\
&= \sum_{X \in \mathcal{X}} \pi(X) \\
&= \sum_{X \in \mathcal{X}} \left(\sum_{S \subseteq X} y_S + \sum_{S \supseteq \overline{X}} y_S \right) \\
&= \sum_{X \in \mathcal{X}} \sum_{S \subseteq X} y_S + \sum_{X \in \mathcal{X}} \sum_{S \supseteq \overline{X}} y_S \\
&\leq \sum_{S \subseteq \overline{V_T}} y_S + \sum_{X \in \mathcal{X}} \sum_{S \supseteq \overline{X}} y_S \qquad (4.20) \\
&\leq \sum_{S \subseteq \overline{V_T}} y_S + \sum_{S \supseteq V_T} y_S \ . \qquad (4.21)
\end{aligned}$$

A desigualdade (4.20) vale pois todo elemento de $\mathcal{X}$ é disjunto de V_T. Para explicar a desigualdade (4.21), observe que $V_T \subseteq \overline{X}$ e por isso

$$\{S \in \mathcal{L}_F : S \supseteq \overline{X}\} \subseteq \{S \in \mathcal{L}_F : S \supseteq T\} \qquad (4.22)$$

para todo X em $\mathcal{X}$.

Além disso,

$$\{S \in \mathcal{L}_F : S \supseteq \overline{X}\} \cap \{S \in \mathcal{L}_F : S \supseteq \overline{X'}\} = \emptyset \qquad (4.23)$$

para quaisquer dois elementos distintos X e X' de $\mathcal{X}$, já que $\mathcal{X}$ é uma coleção disjunta de conjuntos.

De (4.19) e (4.21) e da relação invariante (i6) segue que

$$\begin{aligned}
c(T) + \pi(\overline{T}) &= \sum_{e \in E_T} y(\mathcal{L}_F(e)) + \pi(\overline{T}) \\
&\leq \sum_{e \in E_T} y(\mathcal{L}_F(e)) + \sum_{S \subseteq \overline{V_T}} y_S + \sum_{S \supseteq V_T} y_S \\
&\leq \left(2 - \frac{2}{n}\right) y(\mathcal{L}_F) \ .
\end{aligned}$$

Da desigualdade anterior e do corolário (4.6), concluímos que

$$c(T) + \pi(\overline{T}) \leq (2 - \frac{2}{n})\mathrm{opt}(G, c, \pi) \ .$$

Isto demonstra (4.17). Analogamente, De (4.19) e (4.21) e da relação invariante (i6') segue

que

$$\begin{aligned} c(T) + 2\pi(\overline{T}) &= \sum_{e \in E_T} y(\mathcal{L}_F(e)) + 2\,\pi(\overline{T}) \\ &\leq \sum_{e \in E_T} y(\mathcal{L}_F(e)) + 2 \sum_{S \subseteq \overline{V_T}} y_S + 2 \sum_{S \supseteq V_T} y_S \\ &\leq 2\, y(\mathcal{L}_F)\ . \end{aligned}$$

Finalmente, da desigualdade anterior e do corolário (4.6) temos que

$$c(T) + 2\,\pi(\overline{T}) \leq 2\,\mathrm{opt}(G, c, \pi)\ .$$

Com isto encerramos a demonstração do teorema. ■

A figura 4.11 é um exemplo (descrito em [FFFdP07]) de que o fator de aproximação do teorema 4.9 é justo. O grafo consiste de um circuito com n vértices. Uma das arestas tem custo $2{+}z$, sendo z um número positivo arbitrariamente pequeno. Esta aresta com custo $2{+}z$ é chamada de *especial*. As demais arestas têm custo 2. Os vértices pontas da aresta especial têm penalidade 10 e os demais vértices têm penalidade 1. O algoritmo aumenta o valor das variáveis duais y_S para cada conjunto unitário S até que todas as arestas ficam justas, com exceção da aresta especial. Em uma ordem arbitrária, elas são incluídas na floresta F e o algoritmo para, devolvendo a árvore composta pelas arestas mais espessas na figura 4.11(a), que tem custo $2(n{-}1)$ e não paga penalidades. Entretanto, a árvore ótima tem custo $2{+}z$ e paga $n{-}2$ de penalidades. Ela consiste apenas da aresta especial, destacada na figura 4.11(b). A razão $\frac{2(n-1)}{2+z+n-2}$ tende a $2{-}\frac{2}{n}$ quando z tende a zero.

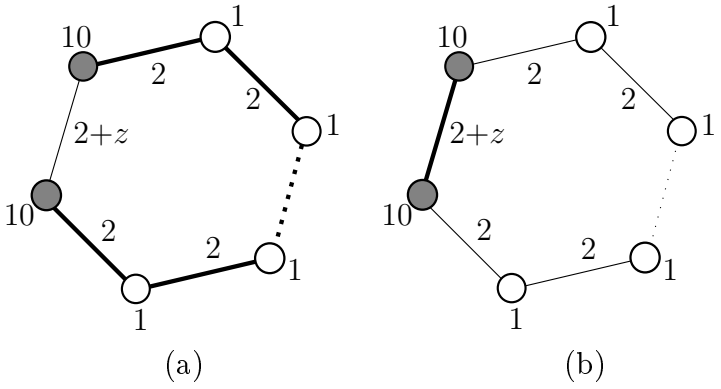


Figura 4.11: (a) *Arestas mais espessas indicam a árvore devolvida pelo algoritmo, com custo* $2(n{-}1)$.
(b) *A única aresta espessa indica a árvore ótima, que tem custo mais penalidades no valor de* $n{+}z$.

4.8 Ilustração do algoritmo PCST-GW

Nesta seção ilustramos o funcionamento do algoritmo PCST-GW.

De maneira idêntica ao que foi feito na seção 3.8 quando vimos uma ilustração do algoritmo MinST-GW, o grafo considerado na ilustração é completo e é dado através de um

desenho plano de seus vértices. A figura 4.12(a) ilustra o grafo dado. Os vértices são representados por circunferências. O número dentro de uma circunferência que representa um vértice indica a penalidade do vértice. O nome de cada vértice é a letra que aparece próxima aos vértices. O custo de cada aresta é representado pela distância euclidiana entre os vértices que são extremos da aresta. Assim, através da figura vemos que o grafo G e função penalidade π considerados no passo 1 do algoritmo PCST-GW são tais que

$$\begin{aligned} &V_G = \{a, b, c, d, e\}, \\ &\pi_a = 3, \pi_b = 8, \pi_c = 7, \pi_d = 9 \text{ e } \pi_e = 9\,. \end{aligned}$$

No início da primeira iteração do algoritmo PCST-Expansão temos que

$$\begin{aligned} y &= 0, \\ F &= \emptyset, \\ \mathcal{L}_F &= \{\{a\}, \{b\}, \{c\}, \{d\}, \{e\}\}, \text{ e} \\ \mathcal{Z} &= \mathcal{M} = \emptyset\,. \end{aligned} \tag{4.24}$$

Como na execução do algortimo MinST-GW, durante as iterações do algoritmo PCST-Expansão as faixas ou molduras se formam ao redor dos vértices, e, posteriormente, ao redor de aglomerados de círculos, indicando os componentes não-nulos de y que são formados por conjuntos ativos em $\mathcal{L}_F$. A largura de uma moldura representa o valor do componente de y formado pelos vértices no interior da moldura. A soma das larguras das molduras ao redor de um mesmo aglomerado é o valor do componente y formado pelos vértices no conglomerado.

Na primeira iteração do algoritmo PCST-Expansão, temos que

$$\mathcal{L}_F^* \setminus \mathcal{Z} \;\; = \{\{a\}, \{b\}, \{c\}, \{d\}, \{e\}\}\,,$$

que é a coleção dos componentes de F ativos. Desta forma, passamos ao caso 2 do algoritmo PCST-Expansão onde os componentes em $\mathcal{L}_F^* \setminus \mathcal{Z}$ de y são acrescidos de ε. Vamos supor que as distâncias entre os vértices são tais que as molduras criadas em cada iteração têm largura de 1 unidade.

Na figura 4.12(b) vemos o resultado dessa primeira execução do caso 2. O crescimento dos componentes de y e das correspondentes molduras na figura 4.12(b) só pararam pois a aresta bc se tornou justa por y. Isto pode ser visto na figura, pois a distância entre os vértices b e c foi totalmente preenchida pelas molduras que envolvem o vértice b e o vértice c. Assim, no caso 2A, a aresta bc é adicionada à floresta F e o conjunto $\{b\} \cup \{c\}$ é inserido na coleção $\mathcal{L}_F$. Com isto, no início da segunda iteração teremos

$$\begin{aligned} F &= \{bc\}, \\ \mathcal{L}_F &= \{\{a\}, \{b\}, \{c\}, \{d\}, \{e\}, \{b, c\}\}, \\ \mathcal{L}_F^* &= \{\{a\}, \{b, c\}, \{d\}, \{e\}\} \text{ e} \\ \mathcal{Z} &= \mathcal{M} = \emptyset\,. \end{aligned}$$

Na segunda iteração do algoritmo PCST-Expansão,

$$\mathcal{L}_F^* \setminus \mathcal{Z} = \{\{a\}, \{b, c\}, \{d\}, \{e\}\}\,,$$

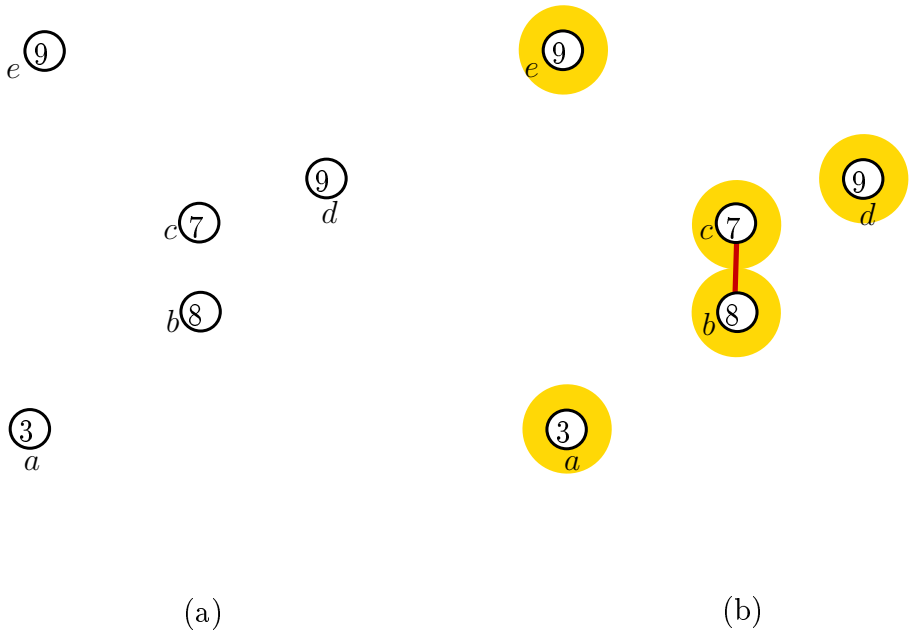
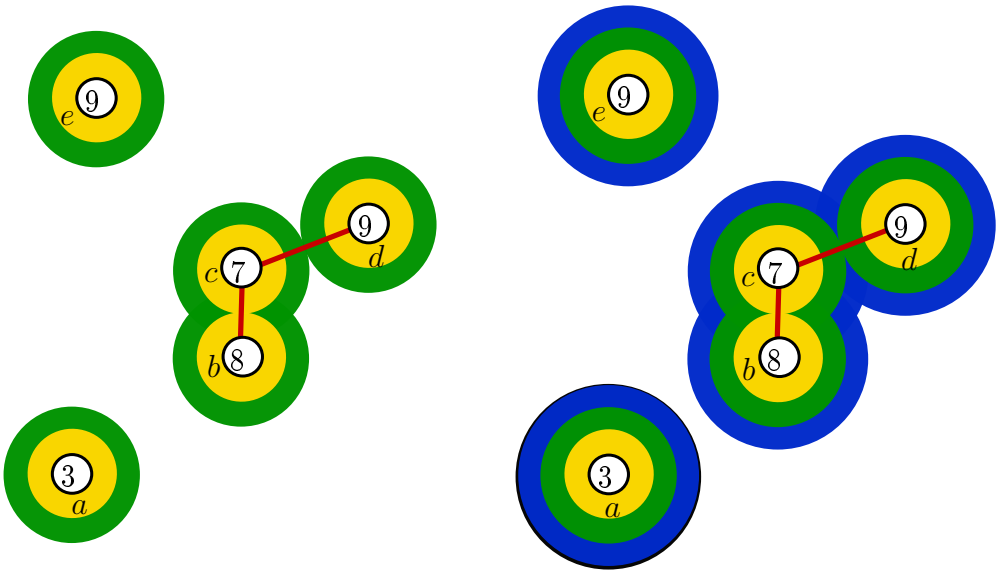


Figura 4.12: (a) *Instância "geométrica" do problema* PCST.
(b) *Situação no final da primeira e início da segunda iteração. Faixas ou molduras indicam os componentes não-nulos de y e a aresta cb torna-se justa por y.*

e o caso 2 será executado novamente. Depois do "incremento uniforme" dos componentes de y em $\mathcal{L}_F^* \setminus \mathcal{Z}$ a aresta cd é a próxima a se tornar justa por y, pois a distância entre os vértices c e d é 4. O caso 2A é mais uma vez executado e ao final da iteração passamos a ter

$$\begin{aligned} F &= \{bc, cd\}, \\ \mathcal{L}_F &= \{\{a\}, \{b\}, \{c\}, \{d\}, \{e\}, \{b, c\}, \{b, c, d\}\}, \text{ e} \\ \mathcal{L}_F^* &= \{\{a\}, \{b, c, d\}, \{e\}\}. \end{aligned}$$

A situação depois desta nova execução do caso 2A encontra-se ilustrada na figura 3.11(c).



(c) (d)

Figura 4.13: (c) *Configuração ao final da segunda iteração do algoritmo* PCST-Expansão. *A aresta cd passa a fazer parte da floresta F.*
(d) *Na terceira iteração o conjunto $\{a\}$ é saturado por y e passa a fazer parte do conjunto $\mathcal{Z}$.*

Em seguida, no início da terceira iteração temos que

$$\begin{aligned} F &= \{bc, cd\}, \\ \mathcal{L}_F &= \{\{a\}, \{b\}, \{c\}, \{d\}, \{e\}, \{b, c\}, \{b, c, d\}\}, \\ \mathcal{L}_F^* &= \{\{a\}, \{b, c, d\}, \{e\}\} \text{ e} \\ \mathcal{Z} &= \mathcal{M} = \emptyset\,. \end{aligned}$$

e portanto

$$\mathcal{L}_F^* \setminus \mathcal{Z} = \{\{a\}, \{b, c, d\}, \{e\}\}\,.$$

Assim, o caso 2 será executado mais uma vez. Desta vez, ao incrementarmos uniformente os componentes de y em $\mathcal{L}_F^* \setminus \mathcal{Z}$ o conjunto $\{a\}$ será saturado por y' pois teremos que

$$\sum_{S \subseteq \{a\}} y'_S + \sum_{S \supseteq \overline{\{a\}}} y'_S = \pi(\{a\})$$

$$y_{\{a\}} = 3 = \pi(\{a\}).$$

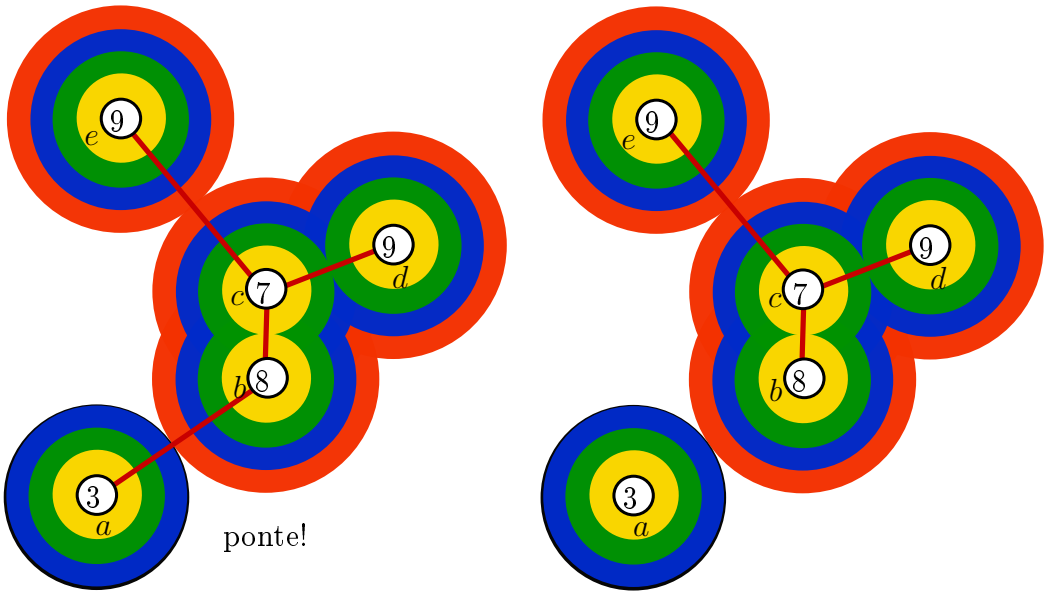
Assim, no caso 2B o conjunto $\{a\}$ será incluído em $\mathcal{Z}$. Intuitivamente, $\{a\}$ passa a fazer parte de $\mathcal{Z}$ pois para conectar a aos demais vértices por uma aresta custaria à função objetivo mais do que 3 unidades, enquanto que a penalidade por não conectar a aos demais vértices é de apenas 3.

Nesta iteração, a cardinalidade da coleção de conjuntos ativos em $\mathcal{L}_F^*$ diminui. Ao final

da iteração temos que

$$\begin{aligned} F &= \{bc, cd, df\}, \\ \mathcal{L}_F &= \{\{a\}, \{b\}, \{c\}, \{d\}, \{e\}, \{b, c\}, \{b, c, d\}\}, \\ \mathcal{L}_F^* &= \{\{a\}, \{b, c, d\}, \{e\}\}, \\ \mathcal{Z} &= \{a\} \text{ e } \mathcal{M} = \emptyset. \end{aligned}$$

A presente configuração se encontra ilustrada na figura 4.13(d).



(e) (f)

Figura 4.14: (e) *A aresta ab e ef ficam justas simultaneamente na quarta iteração. A árvore* T_0 *ilustrada é devolvida pelo algoritmo* PCST-Expansão *na sua sexta e última iteração.*
(f) *A árvore que é devolvida pelo algoritmo* PCST-GW *após a execução de* PCST-Poda.

Na quarta iteração do algoritmo o caso 2 é outra vez executado pois

$$\mathcal{L}_F^* \setminus \mathcal{Z} = \{\{b, c, d\}, \{e\}\}.$$

Desta feita temos que duas novas arestas tornam-se justas, a aresta ab e a aresta ce. Assim, o caso 2A do algoritmo será executado com uma dessas arestas no papel da aresta f do algoritmo PCST-Expansão. Digamos que a aresta ab foi selecionada. Desta forma, ao final da quarta e início da quinta iterações teremos que

$$\begin{aligned} F &= \{ab, bc, cd, df\}, \\ \mathcal{L}_F &= \{\{a\}, \{b\}, \{c\}, \{d\}, \{e\}, \{b, c\}, \{b, c, d\}, \{a, b, c, d\}\}, \\ \mathcal{L}_F^* &= \{\{a, b, c, d\}, \{e\}\}, \\ \mathcal{Z} &= \{a\} \text{ e } \mathcal{M} = \emptyset. \end{aligned}$$

No início da quinta iteração,

$$\mathcal{L}_F^* \setminus \mathcal{Z} = \{\{a, b, c, d\}, \{e\}\}\,.$$

Assim, o caso 2 será executado, mas desta vez como a aresta ce já está justa teremos que o valor de ε calculado no início da iteração será zero. O caso 2A será então executado de tal forma que ao final da iteração teremos

$$\begin{aligned} F &= \{ab, cd, de, ef\},\\ \mathcal{L}_F &= \{\{a\}, \{b\}, \{c\}, \{d\}, \{e\}, \{f\}, \{a, b\}, \{c, d\}, \{b, c, d\}, \{a, b, c, d\}, \{a, b, c, d, e\}\},\\ \mathcal{L}_F^* &= \{\{a, b, c, d, e\}\}\\ \mathcal{Z} &= \{a\} \text{ e } \mathcal{M} = \emptyset\,. \end{aligned}$$

A configuração após a execução da quarta e quinta iterações está ilustrada na figura 4.14(e).

No início da sexta iteração

$$\mathcal{L}_F^* \setminus \mathcal{Z} = \{\{a, b, c, d, e\}\}\,.$$

Portanto, esta é a última iteração do algoritmo PCST-Expansão que devolve a árvore $T_0 = F$, o vetor y e as coleções $\mathcal{L}_F$ e $\mathcal{Z}$.

Voltamos agora ao algoritmo PCST-GW onde o passo 1 acabou de ser executado. No passo 2, o algoritmo PCST-Poda remove a aresta ab de T_0, pois esta é uma ponte em $\mathcal{Z}$. A árvore T resultante será devolvida pelo algoritmo PCST-GW conjuntamente com o vetor y. A árvore T é exibida na figura 4.14(f).

Com isto terminamos a ilustração da execução do algoritmo PCST-GW.

4.9 Árvore de Steiner enraizada com coleta de prêmios

Nesta seção consideramos a versão enraizada do problema de Steiner com coleta de prêmios descrita na seção 4.1:

> **Problema R-PCST(G, r, c, π):** Dados um grafo conexo G, um vértice r em V_G, um custo c_e em $\mathbb{Q}_{\geq}$ para cada aresta e em E_G e uma penalidade π_v em $\mathbb{Q}_{\geq}$ para cada vértice v em V_G, encontrar uma árvore T que contenha r e minimize $c(T) + \pi(\overline{T})$.

A especialização do algoritmo PCST-GW para o problema R-PCST que descrevemos nesta seção é usada no próximo capítulo como uma subrotina do algoritmo com o menor fator de aproximação conhecido para o R-PCST. Apesar de, como vimos na seção 4.1, os problemas PCST e R-PCST serem equivalentes do ponto de vista de algoritmos de aproximação, parece que considerar a versão enraizada é frequentemente mais conveniente. Este fenômeno já foi presenciado na seção 4.2 onde consideramos o PCST restrito a árvores. Resumidamente, o objetivo desta seção é fazer a transição entre o PCST deste capítulo e o R-PCST do próximo capítulo.

Programa linear primal e dual

Os programas lineares primal e dual para o problema R-PCST são muito semelhantes ao do problema PCST visto na seção 4.4. Vimos que o seguinte programa linear é uma relaxação do PCST(G, c, π): encontrar um vetor x indexado por E_G e um vetor z indexado por $\mathcal{A}$ que

$$\begin{array}{rrcll} \text{minimize} & cx + \sum_{A\in\mathcal{A}} \pi(A) z_A & & & \\ \text{sob as restrições} & x(\delta_G(S)) + \sum_{A\supseteq S} z_A + \sum_{A\supseteq \overline{S}} z_A & \geq & 1 & \text{para cada } S \text{ em } \mathcal{A}\ , \\ & x & \geq & 0\ , & \\ & z & \geq & 0\ . & \end{array} \tag{4.25}$$

Passamos agora a considerar o programa linear (4.25) do ponto de vista do R-PCST. Já que o vértice raiz r deve fazer parte de uma árvore candidata a solução do R-PCST então é natural considerarmos que a penalidade π_r da raiz é infinita. Assim, para utilizarmos a relaxação linear (4.25) em um arcabouço primal-dual a fim de obtermos um candidato a solução do R-PCST com um "bom" fator de aproximação podemos supor que

$$z_A = 0 \text{ para cada } A \text{ em } \mathcal{A} \text{ que contém } r,$$

pois em caso contrário teríamos que o valor do segundo termo $\sum_{A\in\mathcal{A}} \pi(A) z_A$ da função objetivo de (4.25) seria infinito.

A discussão anterior indica que, do ponto de vista do problema R-PCST, podemos reduzir o número de componentes do vetor z do programa linear (4.25). Assim, em vez de considerarmos a coleção $\mathcal{A}$ dos subconjuntos dos vértices ativos como sendo formada por todos os subconjuntos de vértices de G, passamos a considerar $\mathcal{A}$ como sendo formada pelos subconjuntos dos vértices ativos que não contêm r, ou seja

$$\mathcal{A} := \{A \subset V_G : r \notin A\}.$$

Convém enfatizar aqui que o mesmo símbolo $\mathcal{A}$ está sendo usado para denotar coleções diferentes de subconjuntos de vértices de V_G. O conceito de conjunto **ativo**, e portanto a coleção $\mathcal{A}$, depende do problema sob consideração, que neste texto pode ser o MinST (seção 3.3), o PCST (seção 4.6) ou o R-PCST (na presente seção e no próximo capítulo). A decisão de usar o mesmo símbolo para objetos distintos tem duas razões. A primeira razão é que acreditamos que o contexto é suficiente para eliminar qualquer ambiguidade, apesar do risco de ser inconveniente. A segunda razão, a mais importante do ponto de vista deste texto, é salientar semelhança dos problemas MinST, PCST e R-PCST e que, essencialmente, o mesmo algoritmo encontra candidatos a solução para todos esses problema com fator de aproximação próximo de 2.

Resumindo a discussão anterior, restrito ao R-PCST o programa linear (4.25) pode ser simplificado. Após as simplificações obtemos o seguinte programa linear que é uma relaxação do R-PCST(G, c, π): encontrar um vetor x indexado por E_G e um vetor z indexado por $\mathcal{A}$

que

$$\begin{array}{rrcll} \text{minimize} & cx + \sum_{A\in\mathcal{A}} \pi(A) z_A & & & \\ \text{sob as restrições} & x(\delta_G(S)) + \sum_{A\supseteq S} z_A & \geq & 1 & \text{para cada } S \text{ em } \mathcal{A}\,, \\ & x & \geq & 0\,, & \\ & z & \geq & 0\,. & \end{array} \tag{4.26}$$

Para certificarmos que o programa linear (4.25) é uma relaxação, basta procedermos de maneira idêntica ao que fizemos na seção 4.4. Se T é uma árvore candidata a solução do problema R-PCST(G, c, π) basta definirmos o seguinte vetor x indexado por E_G

$$x_e := \begin{cases} 1, & \text{se } e \in E_T\,, \text{ e} \\ 0, & \text{caso contrário}\,, \end{cases}$$

e o seguinte z vetor indexado por $\mathcal{A}$

$$z_A := \begin{cases} 1, & \text{se } A = V_G \setminus V_T\,, \text{ e} \\ 0, & \text{caso contrário}\,. \end{cases}$$

É facil verificar que o par (x, z) é um candidato a solução do programa linear (4.26) tal que

$$cx + \sum_{A\in\mathcal{A}} \pi(A) z_A = c(T) + \pi(\overline{T})\,. \tag{4.27}$$

O correspondente programa linear dual do programa (4.5) é: encontrar um vetor y indexado por $\mathcal{A}$ que

$$\begin{array}{rrcll} \text{maximize} & y(\mathcal{A}) & & & \\ \text{sob as restrições} & y(\mathcal{A}(e)) & \leq & c_e & \text{para cada } e \text{ em } E_G\,, \\ & \sum_{S\subseteq A} y_S & \leq & \pi(A) & \text{para cada } A \text{ em } \mathcal{A}\,, \\ & y & \geq & 0\,. & \end{array} \tag{4.28}$$

O programa linear (4.28) sugere que no contexto do problema R-PCST outros conceitos devam ser adaptados, como fizemos para o de subconjuntos ativos. São estes os conceitos de um vetor y indexado por $\mathcal{A}$ respeitar π e o de subconjunto de vértices saturado por y.

Seja $\mathcal{X}$ uma subcoleção de $\mathcal{A}$. No contexto do problema R-PCST dizemos que um vetor y indexado por $\mathcal{X}$ **respeita** (em relação a $\mathcal{X}$) uma função penalidade π definida sobre os vértices V_G se

$$\sum_{S\subseteq X} y_S \leq \pi(X) \qquad \text{para cada } X \text{ em } \mathcal{X}\,. \tag{4.29}$$

Um subconjunto de vértices X está **saturado por** y se (4.29) vale com igualdade.

Com estas adaptações para o R-PCST dos conceitos de

- coleção $\mathcal{A}$ de conjuntos ativos,
- candidato a solução do dual y respeitar uma função penalidade, e de

- um subconjunto de vértices estar saturado por y,

os lema 4.5 da dualidade e a delimitação para o valor do ótimo do corolário 4.6 valem com o mesmo enunciado e demonstrações semelhantes.

Algoritmo R-PCST-GW

Nesta seção apresentamos o algoritmo para o problema do R-PCST. Ao invés de descrever o algoritmo proposto por Goemans e Williamson em [GW95], construímos apenas uma chamada aos algoritmos PCST-Expansão e PCST-Poda descritos anteriormente colocando penalidade suficientemente grande para o vértice raiz para garantir que ele esteja na árvore final devolvida pelo algoritmo. Esses algoritmos serão utilizados como subrotinas do algoritmo do próximo capítulo.

> **Algoritmo R-PCST-GW(G, r, c, π):** Recebe um grafo conexo G, um vértice r em V_G, um custo c_e em $\mathbb{Q}_{\geq}$ para cada aresta e em E_G e uma penalidade π_v em $\mathbb{Q}_{\geq}$ para cada vértice v em V_G. Devolve uma árvore T de G que contém r e tal que $c(T) + 2\pi(\overline{T}) \leq 2\, y(\mathcal{A})$.

Passo 1: Sejam $T_0, y, \mathcal{L}_F$ e $\mathcal{Z}$ a árvore, o vetor e as coleções laminares de subconjuntos de vértices obtidos pela execução do algoritmo R-PCST-Expansão(G, r, c, π).

Passo 2: Seja T a árvore obtida pela execução do algoritmo PCST-Poda$(T_0, \mathcal{Z})$.

Devolva T e y e pare. ∎

O algoritmo R-PCST-Expansão é apenas uma "casca" que faz uma chamada apropriada para PCST-Expansão (seção 4.6), que de fato faz todo o trabalho.

> **Algoritmo R-PCST-Expansão(G, r, c, π):** Recebe um grafo conexo G, um vértice r em V_G, um custo c_e em $\mathbb{Q}_{\geq}$ para cada aresta e em E_G e uma penalidade π_v em $\mathbb{Q}_{\geq}$ para cada vértice v em V_G. Devolve uma árvore T_0 de G que contém r, um vetor y de variáveis duais, uma coleção laminar $\mathcal{L}_F$ de conjuntos de vértices que compuseram a floresta F e uma coleção laminar $\mathcal{Z}$ de conjuntos de vértices que foram desativados.

Passo 1: Seja π' uma função penalidade tal que $\pi'_r = \infty$ e $\pi'_v = \pi_v$ para $v \neq r$.

Passo 2: Seja T_0, y, $\mathcal{L}_F$ e $\mathcal{Z}$ a árvore, o vetor de variáveis duais e a coleção de conjuntos de vértices devolvidos pelo algoritmo PCST-Expansão(G, c, π').
Devolva T_0, y, $\mathcal{L}_F$ e $\mathcal{Z}$. ∎

A demonstração do fator de aproximação do algoritmo R-PCST-GW é análoga ao do teorema 4.9 do fator de aproximação e se apóia nas relações invariantes da seção 4.7.

Teorema 4.10 (do fator de aproximação para o R-PCST-GW) *Se T é uma árvore devolvida pelo algoritmo* R-PCST-GW(G, r, c, π)*, então*

$$c(T) + 2\pi(\overline{T}) \leq 2\,\mathrm{opt}(G, r, c, \pi) \ ,$$

onde $\mathrm{opt}(G, r, c, \pi)$ *e o custo de uma solução de* R-PCST(G, r, c, π).

Demonstração: Sejam $T_0, y, \mathcal{L}_F$ e $\mathcal{Z}$ os objetos devolvida pela execução R-PCST-Expansão(G, r, c, π). Portanto, $T_0, y, \mathcal{L}_F$ e $\mathcal{Z}$ foram construídos pela execução de PCST-Expansão(G, c, π'), onde π' é a função penalidade modificada pelo passo 1 do algoritmo R-PCST-Expansão. Como argumentado no início desta seção, o fato de $\pi_r = \infty$ implica que

- r está em T_0;
- os elementos de $\mathcal{L}_F$ e $\mathcal{Z}$ não contêm r;
- os componentes não nulos de y estão em $\mathcal{L}_F$.

Assim, a subárvore T de T_0 devolvida por R-PCST-GW(G, r, c, π) contém r, é conexa em cada elemento de $\mathcal{L}_F$ e não tem pontes em $\mathcal{Z}$. Desta forma, segue do teorema 4.9 do fator de aproximação que

$$c(T) + 2\pi(\overline{T}) \leq 2\,\mathrm{opt}(G, r, c, \pi) \ .$$

■

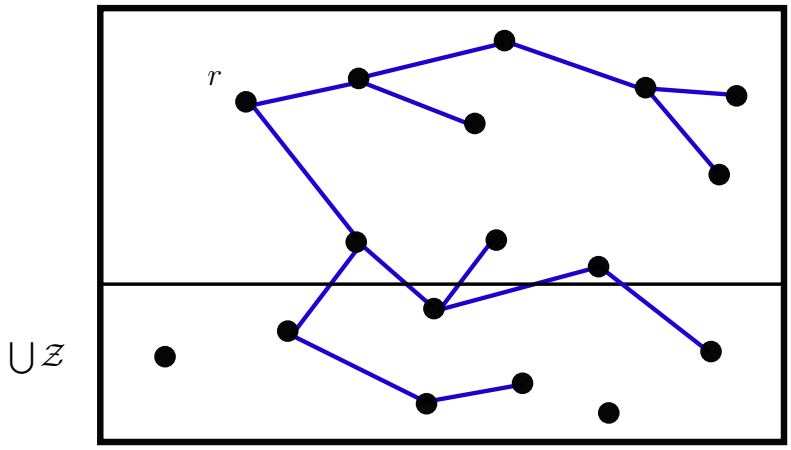


Figura 4.15: *Ilustração da árvore devolvida pelo algoritmo* R-PCST-Expansão.

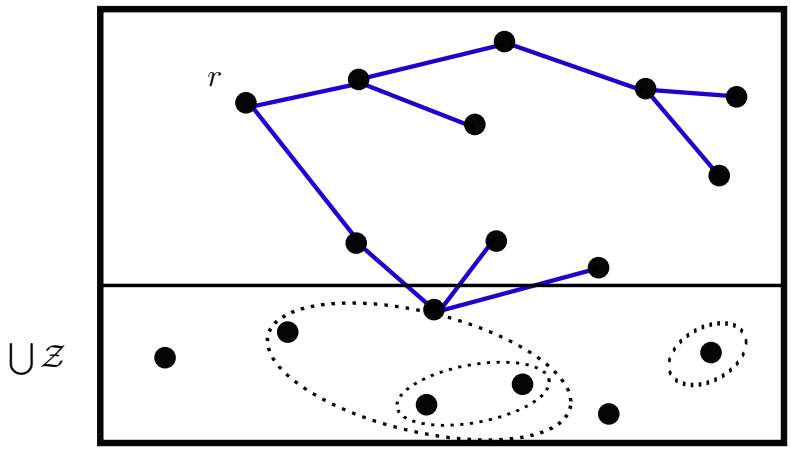


Figura 4.16: *Ilustração da árvore devolvida pelo algoritmo* R-PCST-GW*, depois da execução de* PCST-Poda*. As arestas que eram pontes em $\mathcal{Z}$ foram removidas.*

Este teorema é a base da construção do algoritmo com melhor fator de aproximação para R-PCST apresentado no próximo capítulo.

Capítulo 5

Algoritmo de ABHK

Este capítulo é baseado no estudo feito pelos autores Aaron Archer, Mohammad Hossein Bateni, Mohammad Taghi Hajiaghayi e Howard Jeffrey Karloff (ABHK) e publicado no artigo *Improved approximation algorithms for prize-collecting Steiner tree and TSP* [ABHK11] em 2011, no qual é descrita uma técnica aplicada a problemas de coleta de prêmios que resulta na melhoria do fator de aproximação dos seus respectivos algoritmos.

Apresentamos esta técnica aplicada à versão enraizada do problema da árvore de Steiner com coleta de prêmios. A relaxação linear de R-PCST tem um intervalo de integralidade 2, assim como no caso do MinST. Simplificadamente, **intervalo de integralidade** é o máximo fator entre a solução de um programa inteiro e a sua relaxação. Por isso, esta relaxação não poderia por si só fornecer um limite para construir um algoritmo com fator de aproximação menor do que 2. Este intervalo de integralidade tem sido visto como uma barreira para o aperfeiçoamento de algoritmos para este problema.

ABHK apresentam a descrição e análise de uma técnica aplicada a algoritmos para problemas de coleta de prêmios que resulta em $(2-\varepsilon)$-aproximação. Apesar do valor de ε obtido neste estudo ser pequeno (menor que 0,04), trata-se de um grande avanço conceitual, porque eles apresentam uma técnica única de melhoria de algoritmos de coleta de prêmios que permite contornar a barreira do intervalo de integralidade. Na próxima seção apresentamos uma intuição do algoritmo para R-PCST.

5.1 Ideia do algoritmo

Como descrito por Fabian Ariel Chudak, Tim Roughgarden e David Paul Williamson [CRW04], R-PCST é uma relaxação lagrangeana do problema da k-árvore que é descrito como:

> **Problema k-MST(G, r, c, k):** Dados um grafo conexo G, um vértice r em V_G, um custo c_e em $\mathbb{Q}_{\geq}$ para cada aresta e em E_G e um número inteiro positivo k, encontrar uma árvore T com k vértices, que contenha r e que minimize $c(T)$.

A relaxação linear deste problema pode ser formulada da seguinte maneira:

$$\begin{array}{rrcll} \text{minimize} & cx & & & \\ \text{sob as restrições} & x(\delta_G(S)) + \sum_{A \supseteq S} z_A & \geq & 1 & \text{para cada } S \text{ em } \mathcal{A}\ , \\ & \sum_{S \in \mathcal{A}} |S| z_S & \leq & |V_G| - k\ , & \\ & x & \geq & 0\ , & \\ & z & \geq & 0\ . & \end{array} \tag{5.1}$$

onde, relembrando a notação, $\mathcal{A} = V_G \setminus \{r\}$.

Estes autores observaram uma grande semelhança entre os programas lineares dos problemas K-MST e R-PCST. Simplificadamente, as penalidades π do problema R-PCST podem ser relacionadas à parcela adicionada na função objetivo da relaxação lagrangeana de K-MST aplicada sobre a restrição (5.1) que assegura que T tenha exatamente k vértices. A relaxação lagrangeana do K-MST pode ser descrita da seguinte forma:

$$\begin{array}{rrcll} \text{minimize} & cx + \lambda\left(\sum_{S \in \mathcal{A}} |S| z_S - (|V_G| - k)\right) & & & \\ \text{sob as restrições} & x(\delta_G(S)) + \sum_{A \supseteq S} z_A & \geq & 1 & \text{para cada } S \text{ em } \mathcal{A}\ , \\ & x & \geq & 0\ , & \\ & z & \geq & 0\ . & \end{array} \tag{5.2}$$

Para uma constante λ, esta relaxação equivale ao problema R-PCST com $\pi_v = \lambda$ para todo $v \in V_G$ a menos do termo $-\lambda(|V_G| - k)$ na função objetivo.

Suponha que R-PCST-GW(G, r, c, π) com $\pi_v = \lambda$ para todo vértice v em V_G devolva a árvore T que contém exatamente k vértices. Então T é uma 2-aproximação de K-MST.

Além disso, eles também notaram que o algoritmo R-PCST-GW tem fator de aproximação menor que 2. Este resultado é apresentado no teorema 4.10 e explorado pelo algoritmo R-PCST-ABHK.

O teorema 4.10 apresentado no final da seção 4.9 afirma que a árvore devolvida por R-PCST-GW para o problema R-PCST satisfaz:

$$c(T) + 2\pi(\overline{T}) \leq 2\,\mathrm{opt}(G, r, c, \pi)\ ,$$

onde $\mathrm{opt}(G, r, c, \pi)$ é o valor da solução de R-PCST(G, r, c, π).

É importante notar que este fato apresenta-se mais forte do que seria necessário para que R-PCST-GW seja uma 2-aproximação: há um fator 2 multiplicando $\pi(\overline{T})$, onde o fator 1 seria suficiente. A interpretação deste fato é de que o algoritmo essencialmente obtém o fator de aproximação 2 no custo das arestas e 1 nas penalidades dos vértices.

O teorema 5.1 apresentado mais adiante mostra que:

$$c(T) + \pi(\overline{T}) \leq 2\,\mathrm{opt}(G, r, c, \pi) - \pi(\overline{O})\ ,$$

onde O denota uma árvore que é solução do problema R-PCST(G, r, c, π).

Essa diferença implica que, se uma fração constante ε do valor ótimo corresponde a penalidades, isto é, se vale

$$\pi(\overline{O}) \geq \varepsilon \operatorname{opt}(G, r, c, \pi) \ , \tag{5.3}$$

então a árvore T devolvida por R-PCST-GW(G, r, c, π) é uma $(2-\varepsilon)$-aproximação.

Entretanto, quando esta proporção (5.3) entre o total de penalidades e o valor ótimo não é válida, uma outra abordagem é adotada: buscar uma segunda candidata a solução que tenha garantia de ser melhor do que uma 2-aproximação quando $\pi(\overline{O}) < \varepsilon \operatorname{opt}(G, r, c, \pi)$. O algoritmo resultante da combinação dessas duas políticas devolve a árvore com menor custo dentre as duas candidatas a solução.

Pode-se generalizar o comportamento de qualquer algoritmo para o problema R-PCST como a tomada de duas decisões em sequência, mesmo que muitos deles não trabalhem explicitamente desta maneira: decidir qual subconjunto S de V_G deve pertencer à árvore final, em seguida, construir uma árvore T tal que $V_T = S$.

Existem 3 pontos nos quais o algoritmo pode falhar sobre uma dada instância R-PCST(G, r, c, π):

1. O conjunto de vértices escolhidos para ficar de fora da árvore contribui com muita penalidade em relação ao valor de $\operatorname{opt}(G, r, c, \pi)$, isto é, $\pi(\overline{S})$ é muito grande;
2. O algoritmo mantém a penalidade $\pi(\overline{S})$ baixa, mas a árvore paga caro pelas arestas para conectar os vértices em S. Visto que existem algoritmos eficientes para o problema da árvore geradora mínima (MST), esta segunda armadilha é facilmente evitada simplesmente escolhendo uma árvore geradora mínima T de S.
3. A terceira falha pode acontecer quando o algoritmo escolhe um conjunto S cuja MST é muito cara.

Obviamente, a maioria dos algoritmos não seleciona S e T em sequência, e sim, constrói uma árvore T que define implicitamente $S = V_T$.

ABHK adotam a seguinte estratégia para gerar a segunda candidata a solução: evitam a primeira armadilha usando R-PCST-Expansão para selecionar um conjunto preliminar S de vértices. Para evitar a terceira armadilha, usam um algoritmo para o problema MinST para possivelmente aumentar o conjunto S, enquanto produzem uma árvore T de custo reduzido e que contenha o novo conjunto S, mesmo que T não seja uma MST de S.

Suponha que seja possível identificar um conjunto de vértices D tal que $\pi(D)$ é uma pequena fração de $\operatorname{opt}(G, r, c, \pi)$, e podemos provar que há uma árvore que contém pelo menos os vértices em $\overline{D} = V_G \backslash D$ e custa apenas um pouco mais do que $\operatorname{opt}(G, r, c, \pi)$. Agora suponha que executamos um algoritmo ρ-aproximação para o MinST, com $\rho < 2$, sobre a instância em que os vértices $\overline{D}$ são terminais. Então, a árvore resultante custará não muito mais do que $\rho \operatorname{opt}$ e, já que ela contém no mínimo $\overline{D}$, paga no máximo $\pi(D)$ de penalidade. Logo, esta árvore é uma $(2-\varepsilon)$-aproximação, se existirem limitantes suficientemente fortes sobre o conjunto D.

A chave da compreensão do algoritmo é que, quando $\pi(\overline{O})$ é pequeno em relação ao valor ótimo, é possível identificar este conjunto D, de vértices não-terminais, que resulta em

uma segunda candidata a solução com valor controlado. O algoritmo R-PCST-Expansão é executado para computar D, e em seguida passa $\overline{D}$ como vértices terminais para algum algoritmo para o MinST. O algoritmo para o MinST é usado como uma caixa-preta: para obter um fator de $2-\varepsilon$ para o R-PCST, qualquer ρ-aproximação com $\rho < 2$ é suficiente. É importante enfatizar que para selecionar os vértices terminais, não usamos simplesmente o conjunto de vértices V_{T_0} que R-PCST-Expansão escolheu para incluir na árvore, e sim um subconjunto cuidadosamente escolhido.

As próximas seções apresentam a descrição do algoritmo e a análise do seu fator de aproximação. Em seguida, descrevemos detalhadamente a construção das duas candidatas a solução geradas pelo algoritmo que batizamos de R-PCST-ABHK.

5.2 Algoritmo R-PCST-ABHK

Nesta seção descrevemos o algoritmo R-PCST-ABHK que usa como subrotina os algoritmos R-PCST-Expansão e R-PCST-GW apresentados na seção 4.6 combinados com um algoritmo para o problema MinST que tenha fator de aproximação $\rho < 2$. Resumidamente, R-PCST-ABHK modifica as penalidades da instância original (G, r, c, π) e produz duas árvores. Ao final, devolve a de menor custo dentre elas.

O algoritmo é parametrizado por uma constante $\beta > 1$. Sua descrição é simples, mas sua análise é bastante complexa. A primeira árvore construída pelo algoritmo é T^{GW}, a saída de R-PCST-GW sobre a instância $(G, r, c, \frac{1}{2}\pi)$. Para a segunda candidata a solução, identificamos um conjunto de vértices D_β por meio de R-PCST-Expansão sobre a instância $(G, r, c, \beta\pi)$. Seja MinST_ρ um algoritmo para o problema MinST com fator de aproximação $\rho < 2$. A árvore T^{ST} é construída executando $\text{MinST}_\rho(G, c, \overline{D_\beta})$, com o conjunto $\overline{D_\beta}$ no papel de vértices terminais.

> **Algoritmo R-PCST-ABHK(G, r, c, π, β):** Recebe um grafo conexo G, um vértice r em V_G, um custo c_e em $\mathbb{Q}_{\geq}$ para cada aresta e em E_G e uma penalidade π_v em $\mathbb{Q}_{\geq}$ para cada vértice v em V_G. Além disso, recebe um parâmetro $\beta > 1$. Devolve uma árvore T tal que $r \in V_T$ e $c(T) + \pi(\overline{T}) \leq (2-\varepsilon)\ \text{opt}(G, r, c, \pi)$ para uma constante ε, sendo ε um número tal que $0 < \varepsilon < 1$, que depende de β.

Passo 1: Seja T^{GW} a árvore obtida na execução do algoritmo R-PCST-GW$(G, r, c, \frac{1}{2}\pi)$.

Passo 2: Seja T^{ST} a árvore obtida da seguinte maneira:

- **Passo 2A.** Sejam T_0, y, $\mathcal{L}_F$ e $\mathcal{Z}$ a árvore, o vetor de variáveis duais e as coleções laminares de conjuntos de vértices devolvidos pela execução do algoritmo R-PCST-Expansão$(G, r, c, \beta\pi)$.
- **Passo 2B.** Seja $D_\beta = \bigcup_{S \in \mathcal{Z}} S$ o conjunto de vértices que pertenceram a algum componente desativado durante o R-PCST-Expansão$(G, r, c, \beta\pi)$.
- **Passo 2C.** Seja T^{ST} a árvore devolvida pelo algoritmo $\text{MinST}_\rho(G, c, \overline{D_\beta})$.

Passo 3: Dentre T^{GW} e T^{ST}, devolva a árvore T que minimize $c(T) + \pi(\overline{T})$.

Ao final, o algoritmo R-PCST-ABHK, de fato, devolve uma árvore pois tanto T^{GW}, saída do algoritmo R-PCST-GW, quanto T^{ST}, saída do algoritmo MinST$_\rho$ são árvores que contêm r. A próxima seção contém a demonstração de que a árvore T satisfaz $c(T)+\pi(\overline{T}) \leq (2-\varepsilon)\ \mathrm{opt}(G,r,c,\pi)$, para alguma constante positiva ε.

5.3 Fator de aproximação

Para facilitar a notação, usaremos simplesmente opt para indicar $\mathrm{opt}(G,r,c,\pi)$: o custo da árvore ótima do problema R-PCST(G,r,c,π). Seja δ a fração do valor ótimo que corresponde a penalidades:

$$\delta = \frac{\pi(\overline{O})}{\mathrm{opt}} \ .$$

Teorema 5.1 (Limitante de T^{GW}) *Se T^{GW} é a árvore obtida no passo 1 do algoritmo* R-PCST-ABHK(G,r,c,π,β)*, então vale que:*

$$c(T^{GW}) + \pi(\overline{T^{GW}}) \leq (2-\delta)\,\mathrm{opt} \ .$$

Seja ρ o fator de aproximação do algoritmo para o problema MinST executado no passo 2C do algoritmo R-PCST-ABHK.

Teorema 5.2 (Limitante de T^{ST}) *Se T^{ST} é a árvore obtida no passo 2 do algoritmo* R-PCST-ABHK(G,r,c,π,β)*, então vale que:*

$$c(T^{ST}) + \pi(\overline{T^{ST}}) \leq \left(\rho(1+(2\beta-1)\delta) \quad + \quad \frac{1-\delta}{\beta} + \delta\right)\mathrm{opt} \ .$$

A demonstração destes teoremas são apresentadas nas próximas seções. A seguir, utilizaremos esses teoremas para a demonstração do fator de aproximação do algoritmo.

Teorema 5.3 (Fator de aproximação de R-PCST-ABHK) *Se $\beta = \frac{2}{2-\rho}$ então o algoritmo* R-PCST-ABHK(G,r,c,π,β) *tem fator de aproximação $2-(\frac{2-\rho}{2+\rho})^2$.*

Demonstração: Seja $B^{GW} = 2-\delta$ e $B^{ST} = \rho\,(1+(2\beta-1)\delta) + \frac{1-\delta}{\beta} + \delta$ os fatores de aproximação das soluções T^{GW} e T^{ST} respectivamente, como enunciado nos teoremas 5.1 e 5.2. Vamos mostrar que $B^{GW}, B^{ST} \leq 2-(\frac{2-\rho}{2+\rho})^2$. Se $\delta \geq (\frac{2-\rho}{2+\rho})^2$, então $B^{GW} \leq 2-(\frac{2-\rho}{2+\rho})^2$ como desejado. Portanto, podemos supor que $\delta < (\frac{2-\rho}{2+\rho})^2$ e vamos provar que $B^{ST} \leq 2-(\frac{2-\rho}{2+\rho})^2$.

$$
\begin{aligned}
B^{ST} &= \rho\,(1+(2\beta-1)\delta)+\frac{1-\delta}{\beta}+\delta \\
&= \delta\left(1-\frac{1}{\beta}+\rho(2\beta-1)\right)+\rho+\frac{1}{\beta} \\
&= \delta\left(1-\frac{2-\rho}{2}+\rho\left(\frac{4}{2-\rho}-1\right)\right)+\rho+\frac{2-\rho}{2} \qquad (5.4)\\
&< \left(\frac{2-\rho}{2+\rho}\right)^2\left(1-1+\frac{\rho}{2}+\frac{4\rho}{2-\rho}-\rho\right)+\rho+1-\frac{\rho}{2} \qquad (5.5)\\
&= \left(\frac{2-\rho}{2+\rho}\right)^2\left(-\frac{\rho}{2}+\frac{4\rho}{2-\rho}\right)+\frac{\rho}{2}+1 \\
&= \frac{(2-\rho)\cancel{(2+\rho)}}{(2+\rho)^2}\left(\frac{-\rho(2-\rho)+8\rho}{2\cancel{(2+\rho)}}\right)+\frac{\rho}{2}+1 \\
&= \frac{(2-\rho)}{(2+\rho)^2}\,\frac{\rho}{2}\,(-(2-\rho)+8)+\frac{\rho}{2}+1 \\
&= \frac{\rho}{2}\left(\frac{(2-\rho)}{(2+\rho)^2}(-(2-\rho)+8)+1\right)+1 \\
&= \frac{\rho}{2(2+\rho)^2}\left((2-\rho)(6+\rho)+(2+\rho)^2\right)+1 \\
&= \frac{\rho}{2(2+\rho)^2}\,(4+\cancel{4\rho}+\cancel{\rho^2}+12+\cancel{2\rho}-\cancel{6\rho}-\cancel{\rho^2})+1 \\
&= \frac{\rho}{2(2+\rho)^2}\,16+1 \\
&= \frac{8\rho}{(2+\rho)^2}+1+1-1 \\
&= 2+\frac{8\rho-(2+\rho)^2}{(2+\rho)^2} \\
&= 2+\frac{8\rho-(4+4\rho+\rho^2)}{(2+\rho)^2} \\
&= 2-\frac{4-4\rho+\rho^2}{(2+\rho)^2} \\
&= 2-\frac{(2-\rho)^2}{(2+\rho)^2}\ .
\end{aligned}
$$

A igualdade (5.4) é obtida substituindo o valor de $\beta=\frac{2}{2-\rho}$. A desigualdade (5.5) é obtida aplicando a hipótese de que $\delta<(\frac{2-\rho}{2+\rho})^2$. ■

A tabela 6.2 mostra um comparativo dos possíveis fatores de aproximação de R-PCST-ABHK em função de ρ.

MinST	ρ	Fator de R-PCST-ABHK
Zelikovsky [Zel93]	1,83	1,9982
Robin e Zelikovsky [RZ05]	1,55	1,9839
Byrka et al. [BGRS10]	1,39	1,9672
$\text{opt}_{\text{MinST}}$	1,00	1,8889

Tabela 5.1: *Fatores de aproximação de* R-PCST-ABHK.

5.4 Candidata a solução T^{GW}

Se T é uma árvore devolvida pelo algoritmo R-PCST-GW(G, r, c, π), então, pelo teorema 4.10 temos que:

$$c(T) + 2\,\pi(\overline{T}) \leq 2\,\text{opt}(G, r, c, \pi)\ .$$

Como nosso principal interesse é prover um bom limitante para a expressão $c(T) + \pi(\overline{T})$, é natural calcular este limitante perturbando as penalidades da instância de entrada, a fim de eliminar o fator 2 que multiplica as penalidades.

Demonstração do teorema 5.1: Este teorema afirma que se T^{GW} é a árvore obtida no passo 1 do algoritmo R-PCST-ABHK(G, r, c, π, β), então vale que:

$$c(T^{GW}) + \pi(\overline{T^{GW}}) \leq (2-\delta)\,\text{opt}\ .$$

Considere a instância do problema R-PCST$(G, r, c, \frac{1}{2}\pi)$ obtida multiplicando-se todas as penalidades por $\frac{1}{2}$. Seja $\text{opt}_{\frac{1}{2}}$ o valor ótimo desta nova instância. E seja T^{GW} a árvore devolvida por R-PCST-GW$(G, r, c, \frac{1}{2}\pi)$.

Aplicando o teorema 4.10 temos que

$$\begin{aligned} c(T^{GW}) + 2\left(\frac{1}{2}\pi\right)(\overline{T^{GW}}) &\leq 2\,\text{opt}_{\frac{1}{2}} \\ c(T^{GW}) + \pi(\overline{T^{GW}}) &\leq 2\,\text{opt}_{\frac{1}{2}}\ . \end{aligned}$$

Entretanto, o objetivo é obter um limitante para o custo da árvore em relação ao valor ótimo da instância original opt. Seja O uma solução da instância R-PCST(G, r, c, π). Temos que

$$\begin{aligned} c(T^{GW}) + \pi(\overline{T^{GW}}) &\leq 2\,\text{opt}_{\frac{1}{2}} \\ &\leq 2\left(c(O) + \frac{1}{2}\,\pi(\overline{O})\right) \qquad (5.6) \\ &\leq 2c(O) + \pi(\overline{O}) \\ c(T^{GW}) + \pi(\overline{T^{GW}}) &\leq 2\,\text{opt} - \pi(\overline{O})\ . \qquad (5.7) \end{aligned}$$

A desigualdade (5.6) vale pois a árvore ótima O é viável para a instância $(G, r, c, \frac{1}{2}\pi)$.

Já a última desigualdade (5.7) vale pois:

$$\text{opt} = c(O) + \pi(\overline{O}) \Rightarrow 2\,c(O) = 2\,\text{opt} - 2\,\pi(\overline{O}) \ .$$

Lembrando que δ é a fração do ótimo que corresponde a penalidades,

$$\delta = \frac{\pi(\overline{O})}{\text{opt}} \ ,$$

temos

$$\begin{aligned} c(T^{GW}) + \pi(\overline{T^{GW}}) &\leq 2\,\text{opt} - \delta\text{opt} \\ &= (2{-}\delta)\,\text{opt} \ . \end{aligned}$$

■

Logo, para qualquer ε que não depende da instância R-PCST(G, r, c, π), se $\delta \geq \varepsilon$, então T^{GW} é uma $(2{-}\varepsilon)$-aproximação. Este fato vem ao encontro da intuição de que o algoritmo R-PCST-GW lida bem especialmente com instâncias cuja árvore ótima paga muita penalidade.

5.5 Candidata a solução T^{ST}

Uma ideia ingênua para obter uma $(2{-}\varepsilon)$-aproximação quando $\delta < \varepsilon$ é executar R-PCST-GW – possivelmente perturbando as penalidades – para obter uma árvore T, e então executar um algoritmo para MinST com V_T como os vértices terminais. Isso pode não ter o efeito desejado, conforme o exemplo abaixo.

Dado um número inteiro $k \geq 2$ e um número z positivo arbitrariamente pequeno, construimos uma instância do R-PCST, onde as penalidades são 0 ou ∞. Ademais, como opt não permite pagar penalidades infinitas e todas as outras penalidades valem zero, tem-se que $\pi(\overline{O}) = 0$, logo, $\delta = 0$.

A instância é composta por um $2k$-circuito alternando entre vértice com penalidade zero e com penalidade ∞, conectados por arestas de custo 1. A raiz da árvore é algum vértice que tem penalidade ∞. Há ainda um vértice com penalidade zero no centro do circuito, com arestas radiais de custo $1{+}z$ para cada vértice com penalidade ∞ do circuito. A figura 5.1 representa a instância com $k = 3$.

Neste caso, a árvore T produzida por R-PCST-GW é um caminho contendo todas as arestas do circuito exceto duas arestas adjacentes a um vértice com penalidade zero. T custa $2k{-}2$ e contém todos os vértices exceto dois com penalidade zero: um do circuito e o vértice do meio (figura 5.2).

Entretanto, a árvore ótima O (figura 5.3) contém apenas as arestas radiais e custa $k(1{+}z)$. Em contraste com T, O contém apenas um vértice de penalidade zero, o do meio.

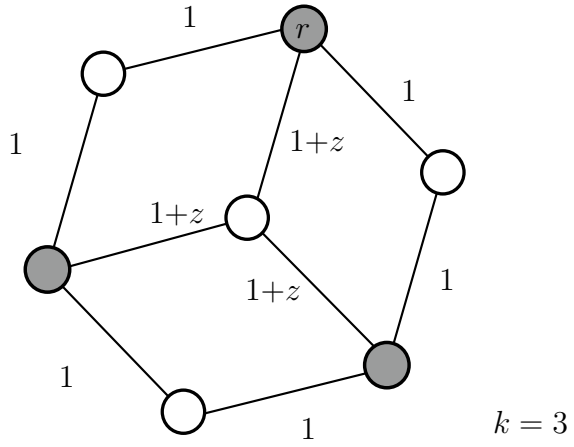


Figura 5.1: *Instância com $k = 3$, onde vértices brancos têm penalidade zero e vértices cinza têm penalidade ∞.*

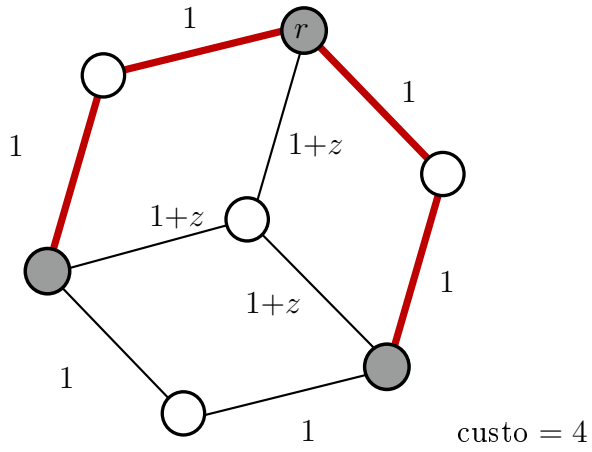


Figura 5.2: *As arestas espessas compõem a árvore T devolvida pelo algoritmo* R-PCST-GW.

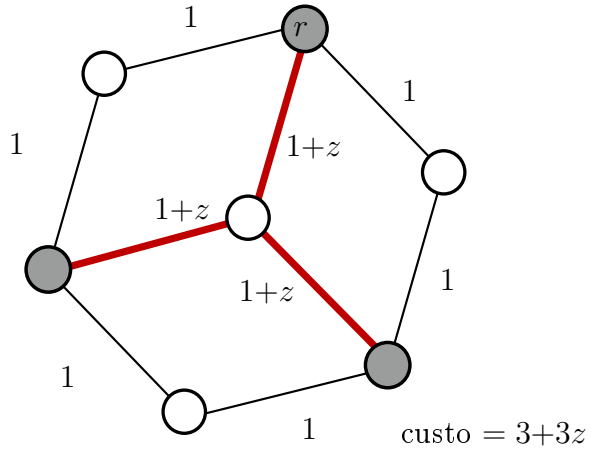


Figura 5.3: *Árvore ótima para esta instância.*

A razão de aproximação $\frac{2k-2}{k(1+z)}$ tende a 2 quando $k \to \infty$ e $z \to 0$. Embora $c(T)$ seja essencialmente 2 opt, a árvore T é na verdade a árvore de Steiner ótima para esta instância com vértices terminais V_T. Logo, executar algum algoritmo para MinST não ajuda a encontrar uma árvore melhor.

Seja R-PCST$(G, r, c, \beta\pi)$ – com $\beta > 1$ – a instância obtida aumentando o valor das penalidades da instância original. Seja D_β a união de todos os conjuntos de vértices de qualquer componente desativado durante a execução de R-PCST-Expansão$(G, r, c, \beta\pi)$.

Seja MinST$_\rho$ um algoritmo com fator de aproximação ρ para o problema MinST, com $\rho < 2$. Para a segunda solução, seja T^{ST} a árvore devolvida pela execução de MinST$_\rho(G, c, \overline{D_\beta})$. Os vértices em $\overline{D_\beta}$ são enviados como terminais para o algoritmo MinST$_\rho$.

Note que o fato de escolher $\overline{D_\beta}$ ao invés de V_{TGW} cobre o exemplo ruim da seção anterior, porque naquele caso, D_β é o conjunto de todos vértices de penalidade zero, então removendo D_β, nos livramos dos $k-1$ vértices de penalidade zero do circuito que fazem parte de V_{TGW} e causavam o problema.

Para obter o fator de aproximação de T^{ST}, primeiro vamos limitar $\pi(\overline{T^{ST}})$ e em seguida $c(T^{ST})$.

Como a árvore T^{ST} contém pelo menos os vértices $\overline{D_\beta}$, ela paga no máximo $\pi(D_\beta)$ de penalidade. Portanto, ao invés de limitar $\pi(\overline{T^{ST}})$, é suficiente limitar $\pi(D_\beta)$. O lema a seguir mostra que $\pi(D_\beta)$ pode ser insignificante se β é grande o suficiente.

Lema 5.4 *Suponha que O seja uma árvore solução do problema* R-PCST(G, r, c, π) *e seja β um número real maior do que 1.*

Se D_β é o conjunto dos vértices em componentes desativados durante a execução de R-PCST-Expansão$(G, r, c, \beta\pi)$*, então vale que*

$$\pi(D_\beta) \leq \frac{\text{opt}_\beta}{\beta} \leq \left(\frac{1-\delta}{\beta} + \delta\right)\text{opt} \ ,$$

onde opt_β *é o custo de uma solução de* R-PCST$(G, r, c, \beta\pi)$, opt *é o custo da árvore ótima* O *da instância* R-PCST(G, r, c, π) *e* $\delta = \frac{\pi(\overline{O})}{\text{opt}}$.

Demonstração: Por construção, um conjunto é desativado quando fica saturado. Logo vale que $\sum_{S \subseteq D_\beta} y_S = \beta\pi(D_\beta)$, já que o algoritmo é executado sobre a instância $(G, r, c, \beta\pi)$. Segue que,

$$\begin{aligned}
\beta\pi(D_\beta) &= \sum_{S \subseteq D_\beta} y_S && (5.8)\\
&\leq \sum_{S \subseteq V_G \setminus \{r\}} y_S && (5.9)\\
&\leq \text{opt}_\beta && (5.10)\\
&\leq c(O) + \beta\pi(\overline{O}) && (5.11)\\
&= c(O) + \pi(\overline{O}) + (\beta - 1)\pi(\overline{O})\\
&= \text{opt} + \beta\delta\,\text{opt} - \delta\,\text{opt} && (5.12)\\
&= (1 + \beta\delta - \delta)\text{opt}\ .
\end{aligned}$$

Logo,

$$\begin{aligned}
\pi(D_\beta) &\leq \left(\frac{1}{\beta} + \delta - \frac{\delta}{\beta}\right)\text{opt}\\
&= \left(\frac{1-\delta}{\beta} + \delta\right)\text{opt}\ . && (5.13)
\end{aligned}$$

A desigualdade (5.9) vale pois $D_\beta \subseteq V_G \setminus \{r\}$. A desigualdade (5.10) é obtida pelo lema da dualidade. A próxima desigualdade (5.11) vale devido à viabilidade da árvore O na instância $(G, r, c, \beta\pi)$, isto é, a árvore O é uma candidata a solução do problema R-PCST$(G, r, c, \beta\pi)$.

A igualdade (5.12) é obtida substituindo $\pi(\overline{O})$ por δopt. Por fim, a igualdade (5.13) conclui esta demonstração.

■

Para limitar $c(T^{ST})$, nós provamos o seguinte fato: começando com uma árvore ótima O, o custo de estender esta árvore para conectar os vértices terminais em $\overline{D_\beta} - V_O$ não é maior que $(2\beta\delta)$opt. Isso garante a existência de uma árvore que contém pelo menos $\overline{D_\beta}$ cujo custo não é muito maior do que opt, desde que δ seja pequeno. Por isso o algoritmo para o MinST não pagará muito mais quando executado com o conjunto de terminais $\overline{D_\beta}$. Este resultado é um corolário do seguinte lema:

Lema 5.5 *Sejam* T, y, $\mathcal{L}_F$ *e* $\mathcal{Z}$ *a árvore, o vetor de variáveis duais e a coleção de conjuntos de vértices devolvidos pelo algoritmo* R-PCST-Expansão$(G, r, c, \beta\pi)$, *com* $\beta > 1$. *Seja* $D_\beta = \bigcup_{S \in \mathcal{Z}} S$ *e seja* I *qualquer subconjunto de* V_G *que contém* r. *Além disso, seja* $A = \overline{D_\beta} \setminus I = \overline{I} \setminus D_\beta$.

Então existe uma floresta $K \subseteq T$ *que possui as seguintes propriedades:*

1. *V_K contém todos os vértices em A;*
2. *Cada árvore na floresta K inclui exatamente um vértice de I;*
3. $c(K) \leq 2\sum_{S\subseteq\overline{I}} y_S \leq 2\beta\pi(\overline{I})$.

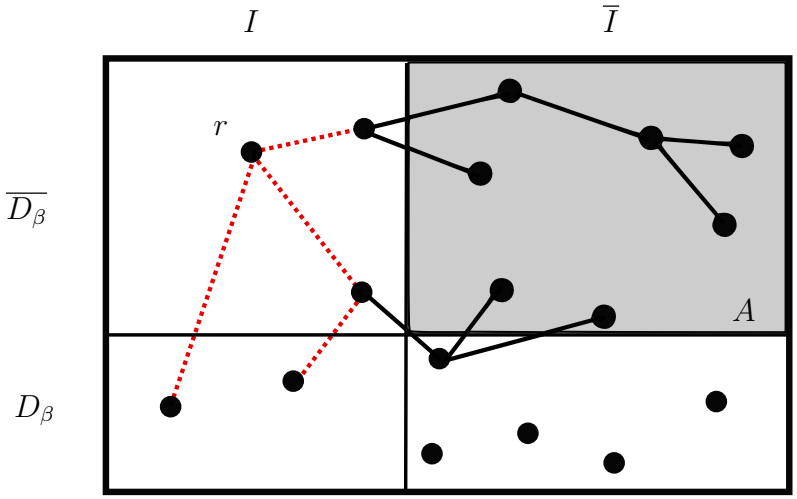


Figura 5.4: *O diagrama ilustra os subconjuntos I e D_β de V_G. As arestas sólidas representam a floresta K.*

Demonstração: Quando fizermos referência a uma iteração do algoritmo R-PCST-Expansão, entende-se que é uma iteração do algoritmo PCST-Expansão que é executado no R-PCST-Expansão.

A floresta K é construída por meio de duas fases de remoção de arestas a partir da árvore T. Descrevemos estas duas fases, mostrando que K satisfaz as propriedades 1 e 2. Em seguida, apresentamos a demonstração da propriedade 3.

Lembramos que $A = \overline{D_\beta} \setminus I$ e que a floresta K que vamos construir deve conectar cada vértice de A a um vértice de I. Pela definição de D_β, $\overline{V_T} \subseteq D_\beta$. Então T já satisfaz a propriedade 1.

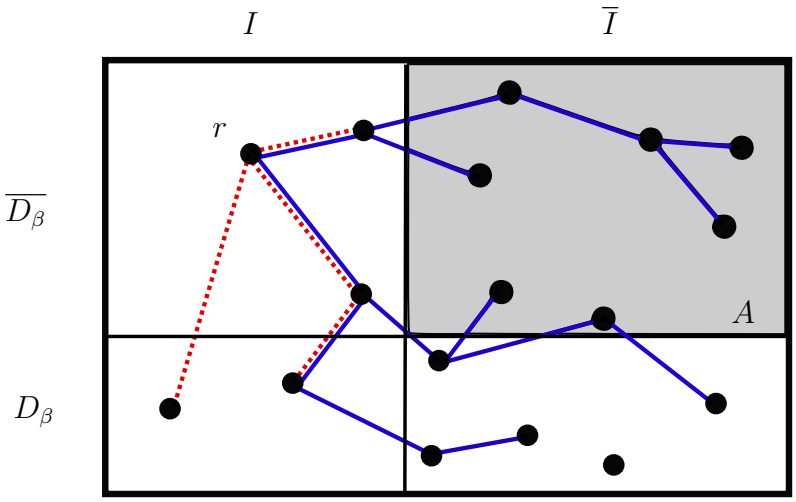


Figura 5.5: *As arestas em linhas pontilhadas conectam vértices em I e as arestas sólidas formam a árvore T.*

Usaremos a linguagem algorítmica para descrever as duas fases da construção da floresta K.

> **PrimeiraFase**(G, T_0, I)**:** Recebe um grafo G, uma árvore T_0 de G e um subconjunto de vértices I de V_G. Devolve uma floresta K_1 que satisfaz as propriedades 1 e 2 do lema 5.5.

Passo 1: Seja $K_1 = T_0$.

Passo 2: Percorra as arestas de K_1 na ordem inversa à que foram adicionadas à floresta T_0 durante a execução de R-PCST-Expansão$(G, r, c, \beta\pi)$. Para cada aresta uv faça:

Caso único: Existem dois vértices distintos i e j em I que pertencem ao mesmo componente da floresta K_1, mas não pertencem ao mesmo componente da floresta $K_1 - uv$.

Comece uma nova iteração com $K_1 - uv$ no papel de K_1.

Passo 2: Devolva K_1.

As remoções de arestas feitas nas iterações da PrimeiraFase preservam o fato de que cada componente de K_1 contém pelo menos um vértice de I. Este fato vale inicialmente porque T e I contêm r. Nenhuma aresta removida no caso único isola um vértice que pertence a A. Portanto, a propriedade 1 continua satisfeita ao final desta fase.

Sejam dois vértices distintos i e j em $I \cap V_{T_0}$ e seja a aresta uv a última aresta adicionada à T_0 no caminho que conecta i a j. Ao analisar a aresta uv no caso único da PrimeiraFase, ela é removida. Então, K_1 termina com exatamente um vértice de I por componente, satisfazendo a propriedade 2.

Para a descrição da segunda fase, lembremos a notação usada para coleção laminar:

$$\mathcal{L}_F[\overline{I}] := \{S \in \mathcal{L}_F : S \subseteq \overline{I}\} \ .$$

A segunda fase funciona como o algoritmo PCST-PODA, exceto pela definição da coleção $\mathcal{Z}$ para verificar pontes. Seja a coleção $\mathcal{Z}' = \mathcal{Z} \cap \mathcal{L}_F[\overline{I}]$ de conjuntos de vértices que foram desativados durante a R-PCST-EXPANSÃO e que pertencem à coleção $\mathcal{L}_F[\overline{I}]$.

> **SegundaFase($G, K_1, \mathcal{Z}'$):** Recebe um grafo G, a floresta K_1 devolvida pela primeira fase da construção, e uma coleção de subconjuntos de vértices de G. Devolve uma floresta K_2 que satisfaz as propriedades 1 e 2 do lema 5.5.

Cada iteração começa com uma floresta K_2. No início da primeira iteração tem-se $K_2 = K_1$.

Caso 1: K_2 não tem pontes em $\mathcal{Z}'$

Devolva K_2 e pare.

Caso 2: K_2 tem alguma ponte em $\mathcal{Z}'$

Seja S em $\mathcal{Z}'$ tal que $|\delta_{K_2}(S)| = 1$.

Comece uma nova iteração com $K_2 - S$ no papel de K_2.

Todo conjunto S que forma ponte em $\mathcal{Z}'$, isto é, $|\delta_{K_2}(S)| = 1$, é removido da floresta K_2. Como $\mathcal{Z}'$ contém apenas vértices de $D_\beta \setminus I$, nenhum vértice de A ou I é removido. Além disso, qualquer conjunto de $\mathcal{Z}'$ não contém uma árvore inteira de K_2, porque cada árvore de K_2 inclui exatamente um vértice de I. Portanto, as propriedades 1 e 2 são satisfeitas ao final da construção de K_2. A figura 5.6 ilustra as fases desta contrução.

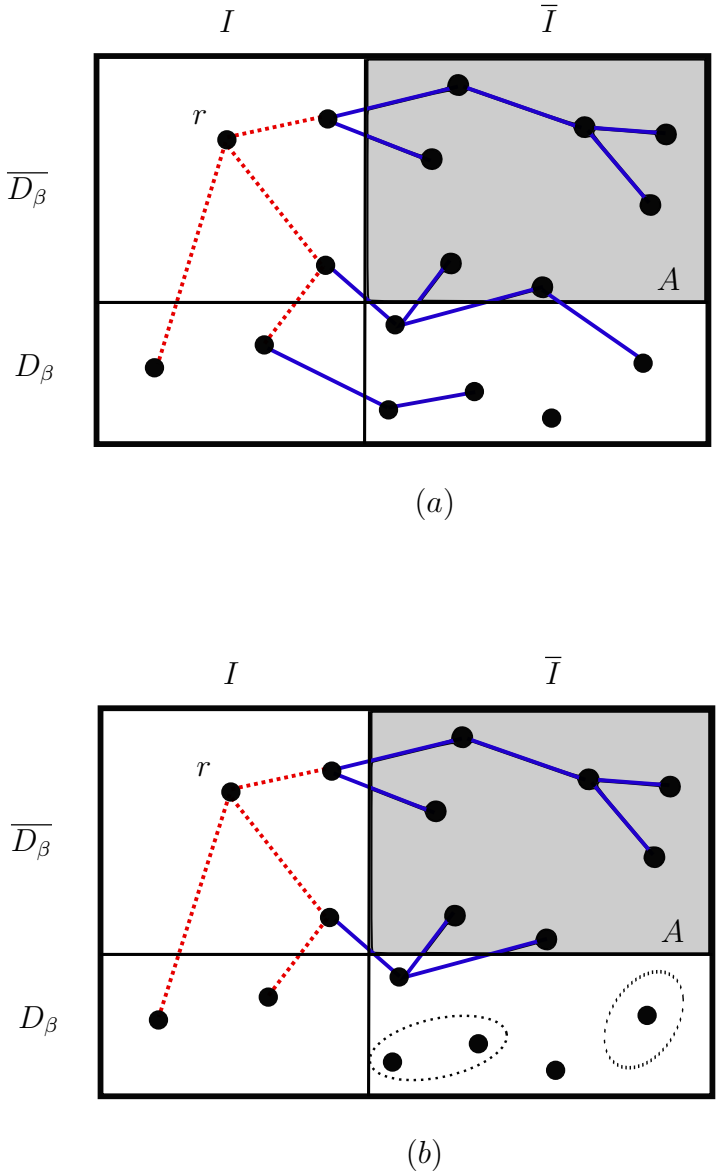


Figura 5.6: (a) *Na primeira fase, as arestas de T que conectam dois vértices de I são removidas.* (b) *Na segunda fase, os conjuntos pontilhados que foram desativados durante a execução de* R-PCST-Expansão *e que não contêm vértices de I são removidos.*

Seja $K = K_2$. Resta mostrar que a floresta K satisfaz a propriedade 3:

$$c(K) \leq 2 \sum_{S \subseteq \overline{I}} y_S \ .$$

Esta parte da demonstração tem muita semelhança com a análise do fator de aproximação do algoritmo para o problema da árvore de Steiner (seção 3.7).

Apenas as variáveis duais dos subconjuntos em $\mathcal{L}_F$ têm valor diferente de zero. Por isso,

a propriedade 3 é equivalente a

$$c(K) \;\leq\; 2 \sum_{S \in \mathcal{L}_F[\overline{I}]} y_S \;=\; 2y(\mathcal{L}_F^*[\overline{I}]) \;. \tag{5.14}$$

Primeiramente, é preciso estabelecer uma relação fundamental entre K e a coleção $\mathcal{L}_F^* \setminus \mathcal{Z}$ dos componentes ativos em uma iteração arbitrária do algoritmo R-PCST-Expansão. Esta relação é descrita pela afirmação (5.15) mais adiante.

Dada uma iteração do algoritmo R-PCST-Expansão$(G, r, c, \beta\pi)$, $\mathcal{L}_F^*$ denota a coleção laminar maximal de conjuntos de vértices dos componentes de F. Denotamos os conjuntos ativos e inativos de $\mathcal{L}_F^*$ respectivamente por $\mathcal{L}_F^* \setminus \mathcal{Z}$ e $\mathcal{Z}^*$.

Digamos que dois elementos S e S' de $\mathcal{L}_F^*$ são **adjacentes** se existe uma aresta em K com uma extremidade em S e outra em S'. Esse conceito de adjacência define um grafo que chamamos de H sobre a coleção laminar $\mathcal{L}_F^*$. Como K foi obtida a partir de T_0 apenas por meio de remoção de arestas, é um subgrafo de T_0. Qualquer circuito em H corresponderia a um circuito em T_0, o que é impossível. Portanto, H é uma floresta.

No início de cada iteração, vale a desigualdade (equivalente ao lema 3.3):

$$\sum_{S \in \mathcal{L}_F^* \setminus \mathcal{Z}} |\delta_H(S)| \leq 2|(\mathcal{L}_F^* \setminus \mathcal{Z})[\overline{I}]| \;. \tag{5.15}$$

Toda folha desativada de H contém algum vértice de I, pois se existisse algum conjunto folha $S \in \mathcal{L}_F^*$ desativado e que não contém vértices de I, ele seria removido na segunda fase da construção da floresta K. Além disso, é possível verificar que cada componente da floresta H possui exatamente um conjunto que contém algum vértice de I.

Seja h o número de componentes em H. Como cada componente de H possui exatamente um conjunto que contém algum vértice de I, sabemos que existem exatamente h conjuntos que contêm algum vértice de I.

Seja l o número de folhas inativas em H. Como toda folha inativa de H contém algum vértice de I, temos que o número de conjuntos ativos que contêm algum vértice de I é no máximo $h - l$.

A coleção $\mathcal{L}_F^* \setminus \mathcal{Z}$ pode ser particionada em duas coleções:

- $(\mathcal{L}_F^* \setminus \mathcal{Z})[\overline{I}]$: conjuntos ativos que não contêm nenhum vértice de I;
- $\{S : S \in \mathcal{L}_F^* \setminus \mathcal{Z} \; e \; S \cap I \neq \emptyset\}$: conjuntos ativos que contêm algum vértice de I.

Então,

$$\begin{aligned} |\mathcal{L}_F^* \setminus \mathcal{Z}| &= |(\mathcal{L}_F^* \setminus \mathcal{Z})[\overline{I}]| \;+\; |\{S : S \in \mathcal{L}_F^* \setminus \mathcal{Z} \; e \; S \cap I \neq \emptyset\}| \\ &\leq |(\mathcal{L}_F^* \setminus \mathcal{Z})[\overline{I}]| \;+\; (h - l) \;. \end{aligned}$$

Por isso, vale que

$$|(\mathcal{L}_F^* \setminus \mathcal{Z})[\overline{I}]| \geq |\mathcal{L}_F^* \setminus \mathcal{Z}| - (h - l) \ . \tag{5.16}$$

O número de vértices em H é $|\mathcal{L}_F^*| = |\mathcal{L}_F^* \setminus \mathcal{Z}| + |\mathcal{Z}^*|$, então o número de arestas em H é $(|\mathcal{L}_F^* \setminus \mathcal{Z}| + |\mathcal{Z}^*| - h)$. Como, a soma dos graus dos vértices de qualquer grafo é igual ao dobro do número de arestas, vale que

$$\sum_{S \in \mathcal{L}_F^*} |\delta_H(S)| = 2(|\mathcal{L}_F^* \setminus \mathcal{Z}| + |\mathcal{Z}^*| - h) \ . \tag{5.17}$$

Como existem l folhas desativadas em H e todo vértice que não é folha tem grau pelo menos 2, vale que

$$\begin{aligned} \sum_{S \in \mathcal{Z}^*} |\delta_H(S)| &\geq 2(|\mathcal{Z}^*| - l) + l \\ &= 2|\mathcal{Z}^*| - l \ . \end{aligned} \tag{5.18}$$

Combinando os resultados obtidos em (5.17) e (5.18), temos que

$$\begin{aligned} \sum_{S \in \mathcal{L}_F^* \setminus \mathcal{Z}} |\delta_H(S)| &= \sum_{S \in \mathcal{L}_F^*} |\delta_H(S)| - \sum_{S \in \mathcal{Z}^*} |\delta_H(S)| \\ &\leq 2(|\mathcal{L}_F^* \setminus \mathcal{Z}| + |\mathcal{Z}| - h) - (2|\mathcal{Z}^*| - l) \\ &= 2|\mathcal{L}_F^* \setminus \mathcal{Z}| - 2h + l \\ &= 2(|\mathcal{L}_F^* \setminus \mathcal{Z}| - (h - l)) - l \\ &\leq 2(|\mathcal{L}_F^* \setminus \mathcal{Z}| - (h - l)) \\ &\leq 2|(\mathcal{L}_F^* \setminus \mathcal{Z})[\overline{I}]| \ . \end{aligned} \tag{5.19}$$

A desigualdade (5.19) é obtida a partir de (5.16) e conclui a demonstração da afirmação (5.15).

Provaremos agora que no início de cada iteração do R-PCST-Expansão vale a desigualdade (equivalente a 3.9):

$$\sum_{S \in \mathcal{L}_F^*} |\delta_H(S)| y_S \ \leq \ 2 \sum_{S \in \mathcal{L}_F^*[\overline{I}]} y_S \ = \ 2y(\mathcal{L}_F^*[\overline{I}]) \ . \tag{5.20}$$

A desigualdade é válida no início da primeira iteração de R-PCST-Expansão, quando $y = 0$. Suponha que a desigualdade seja válida no início de uma iteração qualquer. Durante a iteração, y_S é acrescido de ε se e somente se $S \in \mathcal{L}_F^* \setminus \mathcal{Z}$. Portanto, o lado esquerdo de (5.20) é acrescido de

$$\sum_{S \in \mathcal{L}_F^* \setminus \mathcal{Z}} |\delta_H(S)| \varepsilon \ .$$

Enquanto o lado direito é acrescido de

$$2 \sum_{S \in (\mathcal{L}_F^* \setminus \mathcal{Z})[\overline{I}]} \varepsilon \; = \; 2|(\mathcal{L}_F^* \setminus \mathcal{Z})[\overline{I}]| \; \varepsilon \; .$$

A desigualdade 5.15 garante que o incremento do lado esquerdo não é maior que do lado direito. Portanto a desigualdade 5.20 vale no início da iteração seguinte.

Finalmente, concluimos que

$$\begin{aligned} c(K) &= \sum_{e \in K} c_e \\ &= \sum_{e \in K} \sum_{S : e \in \delta_K(S)} y_S && (5.21) \\ &= \sum_{S \in \mathcal{L}_F^*} |\delta_K(S)| y_S && (5.22) \\ &\leq 2 \sum_{S \in \mathcal{L}_F^*[\overline{I}]} y_S \; = \; 2 \sum_{S \subseteq \overline{I}} y_S && (5.23) \\ &\leq 2\beta\pi(\overline{I}) \; . && (5.24) \end{aligned}$$

A igualdade 5.21 vale pois toda aresta e da floresta K está justa por y:

$$c_e = \sum_{S : e \in \delta_K(S)} y_S \; .$$

A igualdade (5.22) segue da anterior por meio de reorganização dos somatórios. A desigualdade (5.23) segue de (5.20). Finalmente, a desigualdade (5.24) vale pois y respeita $\beta\pi$ e conclui a demonstração de que K satisfaz as propriedades 1, 2 e 3.

■

Corolário 5.6 *Seja D_β o conjunto de vértices que pertenceram a conjuntos saturados durante a execução do algoritmo* R-PCST-Expansão$(G, r, c, \beta\pi)$. *Existe uma árvore T tal que*

$$c(T) \leq (1 + (2\beta - 1)\delta)\text{opt}$$

e inclui os vértices em $\overline{D_\beta}$.

Demonstração: Seja $I = V_O$ o conjunto de vértices de uma árvore ótima O para a instância do problema R-PCST(G, r, c, π). Certamente I contém o vértice raiz r, pois O é solução do problema.

Considere a instância R-PCST$(G, r, c, \beta\pi)$ e este conjunto I. Segundo o lema 5.5, existe uma floresta K que satisfaz as propriedades 1, 2 e 3 do lema.

Seja $T = K \cup O$ a árvore obtida pela união da floresta K e da árvore ótima O. Pela propriedade 2 do teorema 5.5, cada componente de K contém exatamente um vértice de O. Logo, T é, de fato, uma árvore. Pela propriedade 1, K contém todos os vértices em $A = \overline{D_\beta} \setminus O$. Portanto, a árvore T contém os vértices em $\overline{D_\beta}$ e possivelmente alguns vértices de D_β também.

Pela propriedade 3, vale que

$$c(K) \;\leq\; 2\,(\beta\pi(\overline{O})) \;=\; 2\beta(\delta\text{opt})\ .$$

Como $c(O) + \pi(\overline{O}) = \text{opt}$ e $\pi(\overline{O}) = \delta\text{opt}$, temos que $c(O) = (1-\delta)\text{opt}$.

Portanto,

$$c(T) \;=\; c(K \cup O) \;\leq\; (1 + (2\beta - 1)\delta)\text{opt}\ .$$

A figura 5.4 do lema 5.5 ilustra a união da árvore ótima (arestas em linhas pontilhadas) com a floresta K (arestas em linhas sólidas).

■

Demonstração do teorema 5.2: Este teorema afirma que, se T^{ST} é a árvore obtida no passo 2 do algoritmo R-PCST-ABHK(G, r, c, π, β), então vale que:

$$c(T^{ST}) + \pi(\overline{T^{ST}}) \leq \left(\rho(1 + (2\beta - 1)\delta) \quad + \quad \frac{1-\delta}{\beta} + \delta\right)\text{opt}\ .$$

O corolário 5.6 mostra que existe uma árvore de Steiner que contém pelo menos os vértices terminais $\overline{D_\beta}$ e que tem custo máximo de $(1 + (2\beta - 1)\delta)\text{opt}$.

Como o algoritmo MinST$_\rho$ é uma ρ-aproximação, temos que

$$c(T^{ST}) \leq \rho\,(1 + (2\beta - 1)\delta)\text{opt}\ .$$

Além disso, como a árvore T^{ST} contém pelo menos os vértices em $\overline{D_\beta}$, ela paga no máximo $\pi(D_\beta)$ de penalidade:

$$\pi(\overline{T^{ST}}) \leq \pi(D_\beta) \leq \left(\frac{1-\delta}{\beta} + \delta\right)\text{opt}$$

pelo lema 5.4. Portanto, somando estes limitantes, temos que

$$c(T^{ST}) + \pi(\overline{T^{ST}}) \leq \left(\rho(1 + (2\beta - 1)\delta) \quad + \quad \frac{1-\delta}{\beta} + \delta\right)\text{opt}\ .$$

■

Com isso, concluimos as demonstrações dos limitantes de T^{GW} e T^{ST} (teoremas 5.1 e 5.2) e, consequentemente, a análise do fator de aproximação de R-PCST-ABHK (teorema 5.3).

Capítulo 6

Considerações finais

Neste trabalho, estudamos o problema da árvore de Steiner (MinST) e o problema da árvore de Steiner com coleta de prêmios na sua versão sem raiz (PCST) e com raiz (R-PCST). Descrevemos e analisamos algoritmos de aproximação para estes problemas que se apoiam no método primal-dual. Além disso, analisamos com detalhes o primeiro algoritmo para o problema R-PCST com fator de aproximação estritamente menor do que 2. Neste capítulo apresentamos as principais conclusões, nossas percepções durante este estudo e possíveis trabalhos futuros.

A tabela a seguir apresenta uma síntese dos problemas e algoritmos apresentados nesta dissertação e seus fatores de aproximação.

Problema	**Algoritmo**	**Fator de aproximação**
MinST	MinST-GW	2 [dCCD+01]
PCST	PCST-GW	$2-\frac{2}{n}$ [FFFdP07]
R-PCST	R-PCST-GW	$2-\frac{1}{n-1}$ [GW95]
R-PCST	R-PCST-ABHK	$2-\varepsilon$ [ABHK11]

Tabela 6.1: *Resumo dos problemas, algoritmos e seus fatores de aproximação.*

R-PCST-ABHK × PCST-GW

A tabela a seguir faz um comparativo entre os fatores de aproximação $2-\varepsilon$ e $2-\frac{2}{n}$ dos algoritmos R-PCST-ABHK e PCST-GW, respectivamente, em relação $n := |V_G|$, a quantidade de vértices de uma instância.

MinST	ρ	$\mathbf{2-\varepsilon}$	**n** >?
Zelikovsky (1993)	1,83	1,9982	1111
Robin e Zelikovsky (2005)	1,55	1,9839	124
Byrka et al.(2010)	1,39	1,9672	60
$\text{opt}_{\text{MinST}}$	1,00	1,8889	18

Tabela 6.2: *Comparativo entre os fatores de aproximação dos algoritmos* R-PCST-ABHK *e* PCST-GW.

A interpretação é de que, se o algoritmo R-PCST-ABHK usa como caixa preta um algoritmo MinST_ρ com $\rho = 1,83$, ele só tem fator de aproximação menor do que o fator de PCST-GW para instâncias com quantidade de vértices superior a 1111. Se é usado um algoritmo MinST_ρ com $\rho = 1,55$, R-PCST-ABHK tem fator de aproximação menos do que o fator de PCST-GW para instâncias com $n > 124$. E, finalmente, se R-PCST-ABHK usa como subrotina um MinST_ρ com $\rho = 1,39$, que é o menor fator de aproximação conhecido para MinST, o algoritmo R-PCST-ABHK tem fator melhor do que o PCST-GW para instâncias com quantidade de vértices superior a 18.

Prize-collecting

Além do problema da árvore de Steiner, existem outros problemas que podem ser generalizados para uma versão *prize-collecting*. A adição deste conceito de "coleta de prêmios" por meio de penalidades faz com que o conjunto de candidatos a solução seja bem mais amplo.

No caso da árvore de Steiner, os candidatos a solução de uma instância do problema são as subárvores que conectam os vértices terminais. Quando a função penalidade é adicionada substituindo os vértices terminais e resultando no problema da árvore de Steiner com coleta de prêmios, os candidatos a solução passaram a ser todas as subárvores. Intuitivamente, para uma instância do PCST, todos os vértices são "terminais" no sentido de que queremos uma árvore que conecte mais vértices quanto possível. Entretanto, existe a possibilidade de deixar de conectar alguns vértices. Esta possibilidade está sujeita a uma multa: pagar pelas penalidades destes vértices.

A seguir, citamos outros exemplos de problemas que podem ser generalizados para uma versão com coleta de prêmios.

O problema do caixeiro viajante com coleta de prêmios (*Price-Collecting Travelling Salesman Problem*) [ABHK11] é definido como:

> **Problema PCTSP(G, c, π):** Dados um grafo conexo G, um custo c_e em $\mathbb{Q}_{\geq}$ para cada aresta e em E_G e uma penalidade π_v em $\mathbb{Q}_{\geq}$ para cada vértice v em V_G, encontrar um circuito C que minimize o valor $c(C) + \pi(\overline{C})$.

No problema do caixeiro viajante, os candidatos a solução são circuitos que contêm todos os vértices do grafo. Com a coleta de prêmios, qualquer circuito é candidato a solução e o valor de um candidato está sujeito à função π.

O problema do caminho mínimo com coleta de prêmios (*Prize-Collecting Path Problem*) [ABHK11] é definido como:

Problema PC-Path(G, c, π): Dados um grafo conexo G, um custo c_e em $\mathbb{Q}_{\geq}$ para cada aresta e em E_G e uma penalidade π_v em $\mathbb{Q}_{\geq}$ para cada vértice v em V_G, encontrar um caminho P que minimize o valor $c(P) + \pi(\overline{P})$.

No problema original do caminho mínimo, são dados dois vértices s e t e os candidatos a solução são todos os caminhos que conectam s a t. Ao considerar o problema com coleta de prêmios, temos a impressão de que o problema é facilitado. Qualquer caminho do grafo é candidato a solução e seu valor está sujeito às penalidades.

Com ou sem a raiz?

Neste texto foram considerados problemas onde havia um vértice especial, chamado de raiz (seção 4.9, 5.1). Noutros problemas não consideramos a presença deste vértice especial (capítulo 3, seção 4.6).

As versões enraizada e não-enraizada costumam ser equivalentes do ponto de vista de complexidade computacional.

Do ponto de vista de consumo de tempo é inconveniente executarmos um algoritmo para a versão enraizada para todo possível candidato a raiz. Este inconveniente pode ter sido a principal motivação dos trabalhos de Minkoff [JMP00] e Feofiloff [FFFdP07].

No entanto, em termos de demonstrações, percebemos que a presença do vértice raiz introduz uma assimetria muito conveniente, como no algoritmo JMP-Poda (seção 4.2).

Relaxação linear

Gostaríamos de destacar a relaxação linear do problema PCST, apresentada na seção 4.4 e mencionada na seção 4.9:

$$\begin{array}{rrcll} \text{minimize} & cx + \sum_{A\in\mathcal{A}} \pi(A) z_A & & & \\ \text{sob as restrições} & x(\delta_G(S)) + \sum_{A\supseteq S} z_A + \sum_{A\supseteq \overline{S}} z_A & \geq & 1 & \text{para cada } S \text{ em } \mathcal{A}\,, \\ & x & \geq & 0\,, & \\ & z & \geq & 0\,. & \end{array} \tag{6.1}$$

Percebemos que este programa linear é capaz de generalizar uma série de problemas de otimização. Na seção 4.9 iniciamos uma discussão acerca deste fato. Dependendo de cada problema a ser considerado, adaptamos o conceito da coleção $\mathcal{A}$ do programa linear.

Para o problema PCST, $\mathcal{A}$ representa a coleção de todos os subconjuntos de vértices, pois, como dito anteriormente, a intuição é de que queremos conectar todos os vértices do grafo.

Já para a versão enraizada R-PCST, o símbolo $\mathcal{A}$ denota os conjuntos de vértices que não contêm a raiz, já que a raiz deve obrigatoriamente pertencer a árvore final. Neste caso, a intuição é de que queremos conectar todos os vértices à raiz, e com isso, o número de componentes do vetor z é reduzido.

Um outro problema que pode ser considerado sob o ponto de vista deste programa linear é o MinPath, problema do caminho mínimo. Supomos que qualquer conjunto de vértices cujo corte separa s e t paga penalidade "infinita" e, por isso, seu componente no vetor z tem valor zero.

Este problema linear também pode generalizar o problema da árvore geradora mínima, MST. A interpretação é idêntica ao considerarmos o problema PCST: queremos conectar todos os vértices do grafo, então $\mathcal{A}$ denota a coleção de todos os subconjuntos de vértices. Entretanto, neste caso a penalidade para qualquer vértice é infinita, garantindo, ao final, que todos os vértices sejam conectados.

Algoritmo R-PCST-ABHK

O capítulo 5 é dedicado a apresentar o algoritmo com melhor fator de aproximação para o problema R-PCST. Em sua execução, duas árvores são construídas e aquela com menor valor é devolvida.

Curiosamente, o algoritmo R-PCST-ABHK não esboça uma tentativa de resolver a instância original do problema. Na construção das duas árvores, as penalidades do grafo são modificadas e os algoritmos R-PCST-GW e R-PCST-Expansão são usados como subrotina.

Contribuições

Este trabalho concentra os principais aspectos sobre algoritmos de aproximação para o problema da árvore de Steiner com coleta de prêmios. Foi apresentado o arcabouço do método primal-dual com origem no problema da árvore de Steiner e as adaptações necessárias ao considerarmos a versão enraizada ou a versão não-enraizada do problema com coleta de prêmios. Por fim, descrevemos o algoritmo com fator de aproximação estritamente menor do que 2 que se baseia em algoritmos do método primal-dual.

Modificamos a notação utilizada em alguns artigos a fim de desenvolver uma dissertação com padronização tanto para os símbolos e conceitos, quanto para as descrições dos algoritmos. Além disso, procuramos formalizar a corretude dos algoritmos por meio de demonstração das relações invariantes.

Na seção 4.2 desenvolvemos um algoritmo linear (modificação de JMP-Poda-I) para o problema da árvore de Steiner com coleta de prêmios em sua versão não-enraizada e restrito a árvores, que até então era desconhecido. Apesar de o algoritmo resolver o problema não-enraizado é curioso que ele foi o resultado da versão iterativa para o problema enraizado.

Trabalhos futuros

Desenvolvemos um trabalho extremamente teórico a respeito de aproximações para o problema da árvore de Steiner com coleta de prêmios. Seria bastante interessante trabalhar em implementações e observar na prática o comportamento destes resultados.

Além disso, seria interessante explorar o R-PCST-ABHK buscando um algoritmo para o problema PCST, versão não-enraizada, com fator de aproximação também estritamente menor do que 2.

Uma outra possibilidade seria estudar abordagens diferentes para solucionar o problema PCST ou R-PCST. Existem algoritmos que não foram mencionados neste texto que se baseiam em um arcabouço diferente do método primal-dual. Entre eles, um algoritmo recursivo devido a Gutner [Gut08] com o mesmo fator de aproximação de R-PCST-GW, baseado no conceito de *local ratio* e um algoritmo devido a Ljubic [LWP+05] que resolve PCST com otimalidade para instâncias específicas.

Referências Bibliográficas

[ABHK09] Aaron Archer, Mohammad Hossein Bateni, Mohammad Taghi Hajiaghayi e Howard Jeffrey Karloff. Improved approximation algorithms for prize-collecting Steiner tree and TSP. Em *Foundations of Computer Science, 2009. FOCS '09. 50th Annual IEEE Symposium on*, páginas 427 – 436, 2009. 3

[ABHK11] Aaron Archer, Mohammad Hossein Bateni, Mohammad Taghi Hajiaghayi e Howard Jeffrey Karloff. Improved approximation algorithms for prize-collecting Steiner tree and TSP. *SIAM J. Comput.*, 40(2):309–332, 2011. 3, 71, 91, 92

[BGRS10] Jaroslaw Byrka, Fabrizio Grandoni, Thomas Rothvoße Laura Sanità. An improved LP-based approximation for Steiner tree. Em *Proceedings of the 42nd ACM Symposium on Theory of Computing*, STOC '10, páginas 583–592, New York, NY, USA, 2010. ACM. 3, 77

[BGSLW93] Daniel Bienstock, Michel Xavier Goemans, David Simchi-Levi e David Paul Williamson. A note on the prize-collecting traveling salesman problem. *Mathematical Programming*, 59:413–420, 1993. 10.1007/BF01581256. 3

[BR92] Piotr Berman e Viswanathan Ramaiyer. Improved approximations for the Steiner tree problem. Em *Proceedings of the third annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '92, páginas 325–334, Philadelphia, PA, USA, 1992. Society for Industrial and Applied Mathematics. 2

[CLRS01] Thomas H. Cormen, Charles Eric Leiserson, Ronald Linn Rivest e Clifford Stein. *Introduction to Algorithms*. The MIT Press, 2º edição, 2001. 14, 36

[CR41] Richard Courant e Nerbert Robbins. *What is Mathematics? An Elementary Approach to Ideas and Methods*. Oxford University Press, 1941. 2

[CRW04] Fabian Ariel Chudak, Tim Roughgarden e David Paul Williamson. Approximate k-MSTs and k-Steiner trees via the primal-dual method and Lagrangean relaxation. *Math. Program.*, 100:411–421, 2004. 71

[dCCD+01] Marcelo Henriques de Carvalho, Marcia Celioli, Ricardo Dahab, Paulo Feofiloff, Cristina Gomes Fernandes, Carlos Eduardo Ferreira, Kátia Silva Guimarães, Flávio Keidi Miyazawa, José Coelho de Pina, José Augusto Ramos Soares e Yoshiko Wakabayashi. *Uma Introdução Sucinta a Algoritmos de Aproximação*. 23º Colóquio Brasileiro de Matemática - IMPA, RJ, 2001. 5, 91

[Feo97] Paulo Feofiloff. Notas de aula de MAC5781 Otimização Combinatória. `http://www.ime.usp.br/~pf/`, 1997. 9

[Feo03] Paulo Feofiloff. *Algoritmos de Programação Linear*. A ser publicado pela EDUSP, 2003. `http://www.ime.usp.br/~pf/prog-lin/`. 9

[Fer89] Carlos Eduardo Ferreira. O problema de Steiner em grafos: uma abordagem poliédrica. Dissertação de mestrado, Instituto de Matemática e Estatística, Universidade de São Paulo, 1989. 14

[FFFdP07] Paulo Feofiloff, Cristina Gomes Fernandes, Carlos Eduardo Ferreira e José Coelho de Pina. Primal-dual approximation algorithms for the prize-collecting Steiner tree problem. *Information Processing Letters*, 103(5):195 – 202, 2007. 3, 51, 54, 58, 60, 91, 93

[FFFPJ02] P. Feofiloff, C.G. Fernandes, C.E. Ferreira e J.C. Pina Jr. $O(n^2 \log n)$ *implementation of an approximation for the prize-collection Steiner tree problem.* Relatório técnico. Universidade de São Paulo, Instituto de Matemática e Estatística, 2002. 54

[FKW04] Paulo Feofiloff, Yoshiharu Kohayakawa e Yoshiko Wakabayashi. Uma introdução sucinta à teoria dos grafos. *IME-USP*, 7(11):2011, 2004. `http://www.ime.usp.br/~pf/teoriadosgrafos`. 5

[GJ90] Michael Randolph Garey e David Stifler Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1990. 14

[Gut08] S. Gutner. Elementary approximation algorithms for prize-collecting Steiner tree problems. *Inf. Process. Lett.*, 107:39–44, June 2008. 95

[GW95] Michel Xavier Goemans e David Paul Williamson. A general approximation technique for constrained forest problems. *SIAM Jornal on Computing*, 24:296–317, April 1995. 3, 21, 23, 51, 54, 68, 91

[HP99] Stefan Hougardy e Hans Jürgen Prömel. A 1.598 approximation algorithm for the Steiner problem in graphs. Em *Proceedings of the Tenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '99, páginas 448–453, Philadelphia, PA, USA, 1999. Society for Industrial and Applied Mathematics. 2

[JMP00] David Stifler Johnson, Maria Minkoff e Steven Phillips. The prize-collecting Steiner tree problem: Theory and practice. Em *Proceedings of the eleventh annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '00, páginas 760–769, Philadelphia, PA, USA, 2000. Society for Industrial and Applied Mathematics. 3, 36, 93

[Kar72] Richard M. Karp. Reducibility among combinatorial problems. Em R. E. Miller e J. W. Thatcher, editors, *Complexity of Computer Computations*, páginas 85–103. Plenum Press, 1972. 14

[KZ97] Marek Karpinski e Alexander Zelikovsky. New approximation algorithms for the Steiner tree problems. *Journal of Combinatorial Optimization*, 1:47–65, 1997. 10.1023/A:1009758919736. 2

[LWP+05] Ivana Ljubic, René Weiskircher, Ulrich Pferschy, Gunnar Klau, Petra Mutzel e Matteo Fischetti. Solving the prize-collecting Steiner tree problem to optimality. Em *Proceedings of ALENEX, Seventh Workshop on Algorithm Engineering and Experiments*, Vancouver, 2005. 3, 95

[Mac87] Nelson Maculan. *The Steiner problem in graphs.*, volume 31 of *Annals of Discrete Mathematics*. Oxford University Press, 1987. 1

[PS00] Hans Jürgen Prömel e Angelika Steger. A new approximation algorithm for the Steiner tree problem with performance ratio 5/3. *J. Algorithms*, 36(1):89–101, 2000. 2

[RZ05] Gabriel Robins e Alexander Zelikovsky. Tighter bounds for graph Steiner tree approximation. *SIAM Journal on Discrete Mathematics*, 19(1):122–134, May 2005. 3, 77

[Zel93] Alexander Zelikovsky. An 11/6-approximation algorithm for the network Steiner problem. *Algorithmica*, 9:463–470, 1993. 10.1007/BF01187035. 2, 77

[Zel95] Alexander Zelikovsky. Better approximations algorithms the network and Euclidean Steiner tree problem. Relatório técnico, Kishinev, 1995. 2

Índice Remissivo