SANTIAGO VIERTEL

PROGRAMAÇÃO MATEMÁTICA E IMERSÕES MÉTRICAS PARA APROXIMAÇÕES EM PROBLEMAS DE CORTE

Dissertação apresentada como requisito parcial à obtenção do grau de Mestre. Programa de Pós-Graduação em Informática, Setor de Ciências Exatas, Universidade Federal do Paraná.
Orientador: Prof. Dr. André Luiz Pires Guedes

CURITIBA

2014

V665p Viertel, Santiago
Programação matemática e imersões métricas para aproximações em problemas de corte / Santiago Viertel. – Curitiba, 2014.
99f. : il., tab.

Dissertação (mestrado) - Universidade Federal do Paraná, Setor de Ciências Exatas, Programa de Pós-graduação em Informática.

Orientadora: André Luiz Pires Guedes
Bibliografia: p. 96-99.

1. Programação (Matemática). 2. Algoritmos de computador. 3. Espaços métricos. I. Guedes, André Luiz Pires. II. Universidade Federal do Paraná. III. Título.

CDD: 519.7

SANTIAGO VIERTEL

PROGRAMAÇÃO MATEMÁTICA E IMERSÕES MÉTRICAS PARA APROXIMAÇÕES EM PROBLEMAS DE CORTE

Dissertação aprovada como requisito parcial à obtenção do grau de Mestre no Programa de Pós-Graduação em Informática da Universidade Federal do Paraná, pela Comissão formada pelos professores:

Orientador: Prof. Dr. André Luiz Pires Guedes
Departamento de Informática, UFPR

Coorientador: Prof. Dr. André Luís Vignatti
Departamento de Informática, UFPR

Prof. Dr. Jaime Cohen
Departamento de Informática, UEPG

Prof. Dr. Murilo Vicente Gonçalves da Silva
Departamento Acadêmico de Informática, UTFPR

Curitiba, 05 de maio de 2014

Ministério da Educação
Universidade Federal do Paraná
Programa de Pós-Graduação em Informática

PARECER

Nós, abaixo assinados, membros da Banca Examinadora da defesa de Dissertação de Mestrado em Informática, do aluno Santiago Viertel, avaliamos o trabalho intitulado, *"Programação matemática e imersões métricas para aproximações em problemas de corte"*, cuja defesa foi realizada no dia 05 de maio de 2014, às 14:00 horas, no Departamento de Informática do Setor de Ciências Exatas da Universidade Federal do Paraná. Após a avaliação, decidimos pela:

(X) **aprovação** do candidato. ( ) **reprovação** do candidato.

Curitiba, 05 de maio de 2014.

Prof. Dr. André Guedes
DINF/UFPR – Orientador

Prof. Dr. André Vignatti
DINF/UFPR – Co-Orientador

Prof. Dr. Jaime Cohen
UEPG – Membro Externo

Prof. Dr. Murilo Vicente Gonçalves da Silva
UTFPR – Membro Interno

MINISTÉRIO DA EDUCAÇÃO - UNIVERSIDADE FEDERAL DO PARANÁ - Setor de Ciências Exatas - Coordenação de Pós-Graduação em Informática

AGRADECIMENTOS

Agradeço a algumas pessoas especiais que fizeram parte dessa etapa da minha vida. Aquelas pessoas que confiam todos os dias no meu potencial, os meus pais Ana Maria Viertel e Osni Viertel e os meus irmãos Giuliana Viertel e Roger Viertel. Vocês são as pessoas que mais me inspiram a lutar em busca das minhas realizações e dedico essa vitória a vocês.

Agradeço aos professores Claudio Cesar de Sá e André Luiz Pires Guedes que me ajudaram durante o processo seletivo do mestrado. Agradeço também aos professores Renato Carmo, André Luiz Pires Guedes, Jaime Cohen e Murilo Vicente Gonçalves da Silva pelas suas importantes considerações na qualificação e na defesa final. A qualidade do conteúdo presente nesse trabalho está relacionada às suas opiniões expressas nessas duas etapas do curso. Agradeço também ao meu coorientador André Luís Vignatti que norteou com responsabilidade o desenvolvimento do presente trabalho.

Agradeço aos meus primos Selma Alvina Gonçalves Bartapelli e Julio Cesar Bartapelli que me disponibilizaram um espaço físico e me apoiaram do início ao fim do curso. Agradeço também aos meus amigos de Joinville e de Curitiba que, de forma direta ou indireta, sempre me deram forças e confiaram no meu potencial.

SUMÁRIO

LISTA DE FIGURAS

RESUMO

Os algoritmos de aproximação são capazes de gerar soluções próximas da ótima demandando tempo de execução polinomial. Vários dos algoritmos de aproximação existentes na literatura solucionam problemas com base em valores numéricos obtidos em soluções de programas matemáticos, como programas lineares e programas vetoriais. Os problemas do corte máximo, do corte multisseparador mínimo e do corte mais disperso podem ser modelados com programas matemáticos inteiros que, se solucionados, resultam em soluções ótimas. Porém, solucionar programas matemáticos inteiros demanda tempo exponencial exigindo, assim, que sejam criados novos programas matemáticos por meio de relaxações. Uma relaxação admite valores contínuos de solução, o que torna possível encontrar a solução ótima do programa relaxado em tempo polinomial. Em contrapartida, para que sejam encontradas soluções próximas da ótima, faz-se necessárias tomadas de decisão baseadas em análises sobre as soluções dos programas relaxados. Os três problemas apresentados são modelados por programas matemáticos relaxados e solucionados com algoritmos desenvolvidos com base em análises geométricas sobre as soluções desses programas. As soluções dos programas lineares relaxados que modelam os problemas do corte multisseparador mínimo e do corte mais disperso são processadas e são visualizadas como pontos cujas distâncias são calculadas pela norma ℓ_1. Sub-rotinas que imergem espaços métricos finitos gerais em espaços métricos definidos pela norma ℓ_1 se mostraram importantes no desenvolvimento de algoritmos de aproximação. Para solucionar o problema do corte mais disperso, são utilizados algoritmos de imersão que possuem uma certa taxa de distorção das distâncias, sendo essa taxa o principal fator para a determinação da garantia de aproximação do algoritmo como um todo.

Palavras-chave: problemas de otimização NP-difíceis, algoritmos de aproximação, programação matemática, espaços métricos, imersões, algoritmos aleatorizados.

ABSTRACT

Approximation algorithms can generate near optimal solutions requiring execution in polynomial time. Many approximation algorithms in the literature follow the idea of searching for numerical values from mathematical programming formulations, usually linear or vector programming, of the problem in consideration. The problems of maximum cut, multiway cut and sparsest cut can be modeled with integer mathematical programs that result in optimal solutions. However, solving an integer mathematical program is a NP-hard problem, so an usual approach is the creation of new mathematical programs through relaxations. A relaxation admits continuous solution, which makes possible to find the relaxed program optimal solution in polynomial time. However, in order to find near optimal solutions one should perform an analyses of this solution obtained by relaxed programs. The three problems presented here are modeled by relaxed mathematical programs and solved by algorithms based on geometric analyses on the solutions of these programs. The solutions of the relaxed linear programs that model the problems of the multiway cut and the sparsest cut are seen as points whose distances are calculated by the ℓ_1 norm. Subroutines that embed general finite metric spaces in metric spaces defined by the ℓ_1 norm proved to be important in approximation algorithms. In order to solve the sparsest cut problem, the approach is using embedding algorithms that have a certain distortion rate of the distances, being this rate the main factor determining the performance guarantee of the algorithm.

Keywords: NP-hard optimization problems, approximation algorithms, mathematical programming, metric spaces, embeddings, randomized algorithms.

CAPÍTULO 1

INTRODUÇÃO

De modo geral, sistemas que precisam manipular uma grande quantidade de objetos precisam ter um *feedback* rápido. Esses sistemas computacionais têm uma particularidade em comum: realizar o processamento de uma carga extensa de dados em um curto espaço de tempo. Muitas vezes essa particularidade é também uma necessidade, fazendo com que algumas abordagens precisem ser usadas para que o tempo de processamento se mantenha limitado. Dessa forma, existe uma necessidade de criar e estudar algoritmos que possuem execução rápida para que os sistemas atuais possam satisfazer essa necessidade de processamento de grandes cargas em um curto período de tempo. Uma das áreas da ciência da computação que abrange esses tipos de estudo é conhecida como *análise de algoritmos*.

Existem problemas que ainda hoje não possuem algoritmos que encontram as melhores soluções possíveis em tempo hábil. De forma geral, os problemas computacionais são divididos em duas classes diferentes: os problemas *tratáveis* e os *intratáveis*. Os problemas tratáveis são aqueles para os quais existem algoritmos (não necessariamente já descobertos) que os solucionam demandando um tempo que pode ser limitado por uma função polinomial na quantidade de dados referentes à entrada do problema. Já os problemas intratáveis são aqueles para os quais não existem algoritmos que os solucionam em tempo polinomial no tamanho da entrada. Um algoritmo é dito *algoritmo polinomial* se a função que determina o seu tempo de execução é um polinômio em função do tamanho da entrada do algoritmo. Dessa forma, um problema computacional é tratável se existe um algoritmo polinomial que o soluciona.

O fato de ainda não existir algoritmo polinomial que soluciona um problema não significa que ele seja menos importante. Pelo contrário, esses problemas são os mais estudados atualmente na comunidade acadêmica. Muitos problemas comuns do mundo

real ainda são intratáveis e precisam ser abordados de uma forma diferenciada. Parte dos problemas intratáveis são *problemas de otimização*, cujo objetivo se resume em encontrar a melhor resposta dentro de um conjunto de respostas possíveis. Parte dos problemas de otimização envolve a busca de uma resposta ótima por meio de combinação de elementos discretos. Quando o conjunto domínio de um problema de otimização é formado por variáveis discretas, então o problema é chamado de *problema de otimização combinatória*.

Um problema de otimização combinatória possui um conjunto de elementos resposta para cada entrada diferente, essa última também chamada de *instância*. Nessa classe de problemas existe uma função que associa cada solução da instância com um valor racional positivo denominado *valor da solução*. O objetivo de problemas de otimização combinatória é, dada uma instância, encontrar a resposta que possui o menor valor da solução, para *problemas de minimização*; ou o maior valor da solução, para *problemas de maximização*.

Muitos problemas existentes na atualidade podem ser entendidos como problemas de otimização combinatória, que podem ser também problemas NP-*difíceis*. Informalmente falando, Garey e Johnson classificam um problema de otimização como NP-difícil se ele é tão difícil de solucionar quanto os problemas mais difíceis na classe NP [16], sendo esses últimos os problemas NP-completos. Ainda não são conhecidas soluções polinomiais para nenhum problema NP-difícil. A solução que demanda tempo polinomial de um único problema NP-difícil implica na existência de solução que demanda tempo polinomial para todos os problemas NP-completos. O melhor valor da solução pode ser o maior valor se for um problema de maximização ou o menor se for um problema de minimização. Mais conceitos sobre tempo de execução de algoritmos, problemas de otimização combinatória e problemas NP-difíceis podem ser encontrados no livro escrito por Garey e Johnson [16].

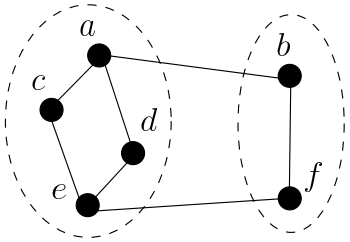
São estudadas no presente trabalho diferentes abordagens para solucionar alguns problemas pré-selecionados NP-difíceis de cortes em grafos. Essas abordagens geram *algoritmos de aproximação*, que são algoritmos polinomiais que encontram respostas cujo valor da solução segue uma taxa de aproximação em relação ao valor da solução da melhor resposta. Tais algoritmos podem fazer uso de diversas ferramentas. Algumas ferramentas

usadas e que são abordadas nesse trabalho são: programas lineares, programas vetoriais e métricas. Todos esses conceitos são melhor abordados no Capítulo 2 enquanto que, no que segue, são mostradas explicações breves sobre o que são cortes em grafos, algoritmos de aproximação, programas matemáticos e métricas.

1.1 Cortes em grafos

Um *corte* em um grafo $G = (V, E)$ corresponde a uma partição do conjunto de vértices V em dois subconjuntos disjuntos U e $W = V - U$. É possível determinar um conjunto de arestas $F \subseteq E$ formado por todas as arestas que possuem um vértice em U e o outro vértice em W. Esse conjunto de arestas é chamado de *conjunto de corte*. Um *k-corte* é uma variação do conceito de corte que corresponde a uma partição do conjunto de vértices V em k subconjuntos disjuntos $C_1, \ldots, C_k$. Para esse caso, o conjunto de corte é formado por todas as arestas que possuem um de seus vértices pertencente a um subconjunto disjunto C_i e o outro pertencente a outro subconjunto disjunto C_j, sendo $i \neq j$. Dependendo do contexto do problema, será usado simplesmente o termo corte para denotar um k-corte por questões de simplificação da nomenclatura.

Supondo que $F \neq \emptyset$, caso as arestas pertencentes ao conjunto de corte F sejam removidas do grafo original G, o grafo resultante passa a ter um número maior de componentes conexos em comparação a G. A Figura 1.1 ilustra os conjuntos U, W e F em um grafo G apresentado. É possível visualizar na figura que se as arestas contidas em F forem removidas, é acrescido o número de componentes conexos no grafo.



$U = \{a, c, d, e\}$ $W = \{b, f\}$
$F = \{\{a, b\}, \{e, f\}\}$

Figura 1.1: Corte em um grafo

O *tamanho* do corte é definido como sendo a cardinalidade do conjunto de corte. No

caso de grafos com pesos nas arestas, o *peso* do corte é definido como sendo a soma dos pesos das arestas pertencentes ao conjunto de corte. É possível notar que, caso o grafo possua pesos unitários nas arestas, o peso de um corte é igual ao tamanho do mesmo corte. Serão utilizadas em alguns momentos as notações *custo* e *capacidade* como sendo o peso.

Existem inúmeros problemas na literatura que buscam cortes em grafos não direcionados. Normalmente é requisitado que esses cortes possuam algumas propriedades, como um corte com o menor peso, por exemplo. No presente trabalho, são apresentadas soluções para problemas cujos objetivos são encontrar cortes que possuem certas propriedades.

1.2 Algoritmos de aproximação

Muitos problemas importantes de otimização discreta existentes na literatura pertencem ao conjunto de problemas NP-difíceis. Esses tipos de problemas podem ser tratados de três formas diferentes ao se buscar soluções [9]: o desenvolvimento de algoritmos que executam em tempo exponencial; subdividir o problema em casos mais específicos e solucioná-los com um algoritmo que executa em tempo polinomial; e usar abordagens que encontram soluções próximas da solução ótima em tempo polinomial. Essa última categoria se divide em algoritmos de aproximação e heurísticas.

De forma geral, supondo que $\mathsf{P} \neq \mathsf{NP}$, então é impossível: (i) desenvolver um algoritmo que encontre soluções ótimas, (ii) em tempo polinomial e (iii) para todas as instâncias de um problema NP-difícil [37]. Pelo menos um dentre (i), (ii) ou (iii) precisa ser desconsiderado ao se lidar com problemas NP-difíceis. Algoritmos de aproximação são desenvolvidos com o intuito de encontrar em tempo polinomial soluções que sejam próximas da resposta ótima para qualquer instância.

Encontrar soluções que são "boas o suficiente" ao invés de uma solução ótima é uma abordagem interessante, principalmente quando é possível encontrá-la em tempo polinomial. O foco desse trabalho se dá na análise de algoritmos que encontram soluções próximas da ótima em tempo polinomial para problemas de corte NP-difíceis. Algoritmos com o objetivo de realizar tal tarefa são denominados *algoritmos de aproximação*. São

analisados no presente trabalho algoritmos de aproximação aplicados em problemas de otimização discreta.

1.3 Programação matemática

Existem problemas cujo objetivo é encontrar soluções que maximizam ou minimizam um determinado objetivo dados recursos limitados e restrições. Se for possível especificar o objetivo como uma função sobre variáveis que representam esses recursos e restrições em equações ou inequações sobre as mesmas variáveis, então possivelmente tem-se um problema de *programação matemática*. A programação matemática pode ser dividida em *programação linear* e *programação não linear*.

A programação linear possui a função objetivo e as restrições definidas por funções lineares, enquanto que na programação não linear não existe essa exigência. Isso faz com que o conjunto dos problemas de programação não linear seja maior em comparação ao conjunto dos problemas de programação linear. De fato, um programa linear inclusive pode ser formulado como um programa não linear. Pelo fato da programação não linear ser mais abrangente, ela é um problema mais difícil de resolver.

Existem vários tipos de programação não linear, porém esse trabalho apresentará somente algoritmos que fazem uso de um tipo específico de programação não linear: a *programação semidefinida*. A programação semidefinida é um tipo de programação não linear que possui parte de suas restrições formuladas por funções que podem ser não lineares. Dessa forma, um programa linear também pode ser formulado como um programa semidefinido, mas o contrário não é necessariamente verdade. De forma geral, o conjunto que engloba todos os programas não lineares engloba também o conjunto de programas semidefinidos que engloba o conjunto de programas lineares.

A programação linear e a programação semidefinida podem trazer ótimas alternativas para solucionar problemas, mostrando-se eficientes na busca de soluções, bem como poderosas ferramentas matemáticas para avaliar algoritmos. Para alguns problemas, o uso de programação semidefinida pode aumentar substancialmente a aproximação de um algoritmo.

1.4 Métricas

Métricas são um tipo específico de funções que medem a distância entre todos os pares de elementos pertencentes a um conjunto. Para ser considerada uma métrica, uma função de distância deve atribuir um valor real positivo a todos os pares de elementos pertencentes ao conjunto e deve satisfazer a outras três restrições: (i) se a distância entre dois elementos é igual a zero, então ambos fazem referência ao mesmo elemento e o contrário também é verdadeiro; (ii) é simétrica, ou seja, a distância entre um par de elementos em um sentido é igual a distância entre o mesmo par no sentido oposto e outra (iii) denominada *desigualdade triangular*. A desigualdade triangular (iii) é uma propriedade que impõe a seguinte restrição na função de distância d: $d(i, j) \leq d(i, k)+d(k, j)$ para todo i, j e $k \in V$. Isso quer dizer que o tamanho de um caminho que interliga um par de vértices e que passa por um ou mais vértices intermediários é obrigatoriamente igual ou maior à distância entre esses dois vértices. Já a primeira restrição (i) faz com que as métricas não atribuam valores zerados de distância, a não ser que a distância foi medida entre um elemento e ele mesmo. Essa propriedade é importante e pode ser utilizada para encontrar vértices que referenciam um mesmo elemento. Quando a distância entre um par de vértices é igual a zero, então ambos os vértices fazem referência um mesmo elemento.

As métricas são capazes de modelar matematicamente vários problemas de cortes em grafos. Pelo fato das métricas serem uma característica presente em alguns problemas, elas podem ser utilizadas como ferramenta para solucioná-los. Assim como programação matemática, essa é uma ferramenta poderosa para criar e analisar algoritmos de aproximação por meio de suas propriedades. Muitos problemas importantes de otimização discreta, como aqueles que buscam cortes em grafos, podem ser solucionados por algoritmos que foram criados e analisados por meio dessa ferramenta.

1.5 Objetivos

O objetivo do presente trabalho é estudar e analisar diferentes algoritmos de aproximação aplicados em três problemas de cortes em grafos NP-difíceis. Outro objetivo é estudar as

técnicas de programação matemática e de aproximação por espaços métricos usados nos algoritmos. Os objetivos específicos são:

1. Estudar o conceito de algoritmos de aproximação e alguns exemplos conceitualmente simples;
2. Estudar técnicas que fazem uso de programação linear e programação vetorial;
3. Estudar algoritmos de aproximação para o problema do corte multisseparador mínimo;
4. Estudar o conceito de espaços métricos e técnicas de aproximação de métricas em árvores;
5. Estudar algoritmos de aproximação para o problema do corte mais disperso.

1.6 Resultados obtidos

O presente trabalho mostra uma forma alternativa de solucionar o problema do corte mais disperso projetada pelo mestrando. Essa solução é apresentada e demonstrada na seção 4.3 e foi desenvolvida com base nos estudos realizados durante todo o período do mestrado e na solução adotada por Aumann e Rabani [4]. Apesar dessa solução alternativa não ser a melhor existente, nota-se a sua importância no ponto de vista conceitual do presente trabalho. Para que ela pudesse ser desenvolvida, foi estudado e apresentado o conceito de métricas em árvores e como um espaço métrico finito pode ser imergido em métricas em árvores, conceitos esses utilizados na solução apresentada pelo mestrando. O correto emprego dessas técnicas em uma nova solução desenvolvida pelo próprio mestrando traz um resultado considerável obtido no presente trabalho.

O presente trabalho possui um embasamento teórico escrito na língua portuguesa que explica e exemplifica de forma didática a aplicação de conceitos matemáticos como espaços métricos, imersões métricas e programação matemática no desenvolvimento de algoritmos de aproximação. Materiais que apresentam tais estudos redigidos na língua local são

escassos e de difícil acesso. O conteúdo disponibilizado no presente trabalho pode servir como uma fonte inicial de estudos que abordam os conceitos mencionados.

Nota-se que o correto entendimento dos resultados obtidos no estudo de cada algoritmo apresentado exige um conhecimento inicial e dedicação do estudante. O próprio desenvolvimento pessoal e da maturidade técnica do mestrando são também resultados importantes obtidos no presente trabalho. A dificuldade no aprendizado do conteúdo adquirido de diferentes fontes confiáveis propiciou o mestrando a desenvolver uma evolução intelectual que poderá ser utilizada para realizar pesquisas inéditas no âmbito da análise teórica de algoritmos.

1.7 Estrutura do trabalho

É mostrado no Capítulo 1 a justificativa e problemática que são tratadas no trabalho. São mostrados também alguns conceitos iniciais, o objetivo geral e específicos bem como os resultados obtidos no mestrado. No Capítulo 2 são apresentadas explicações sobre os problemas estudados, algoritmos de aproximação, sobre dois tipos de programas matemáticos e sobre espaços métricos, esses usados para solucionar e analisar os algoritmos estudados. No Capítulo 3 são apresentados conceitos sobre espaços métricos modelados em formato de árvores e sobre imersões métricas. No Capítulo 4 são mostradas diferentes soluções que fazem uso de técnicas de modelagem por programação matemática e por espaços métricos para solucionar alguns problemas de cortes em grafos. No Capítulo 5 são apresentadas algumas conclusões realizadas sobre as análises dos algoritmos apresentados e algumas propostas de trabalhos futuros.

CAPÍTULO 2

FUNDAMENTAÇÃO TEÓRICA

São abordados nesse capítulo vários conceitos necessários para o entendimento dos algoritmos que são estudados no trabalho. Caso o leitor já possua conhecimento sobre os conceitos de programação linear, programação vetorial, espaços métricos finitos, espaços métricos infinitos e algoritmos de aproximação, é possível continuar a leitura a partir do próximo capítulo. Porém, é aconselhável uma leitura rápida de cada um desses conceitos para que o leitor se familiarize com as notações utilizadas no restante do trabalho e conheça sobre os problemas estudados.

2.1 Problemas estudados

São apresentados a seguir os principais problemas que envolvem cortes em grafos analisados nesse documento.

2.1.1 Corte s-t mínimo

Problema 2.1 (definição encontrada em [4])**.** *No* problema do corte s-t mínimo *(minimum s-t cut problem) é dado como entrada um grafo não direcionado* $G = (V, E)$*, com pesos não negativos* w_e *para cada aresta* $e \in E$ *e um par de vértices selecionados* s *e* $t \in V$*. O problema tem como objetivo particionar o conjunto de vértices* V *em dois subconjuntos* U *e* $W = V - U$ *de tal forma que o vértice selecionado* s *esteja em um subconjunto e o vértice selecionado* t *esteja no outro subconjunto. Não obstante, busca-se também o corte cujo peso seja o menor possível.*

A Figura 2.1 ilustra um grafo G com pesos em cada aresta e um par de vértices selecionados s e t. Também é possível visualizar um corte no mesmo grafo por meio dos conjuntos de vértices U e W, que particionam o conjunto V. Também é possível perceber

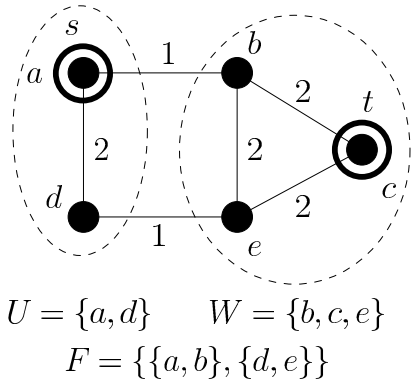


Figura 2.1: Corte mínimo entre dois vértices de um grafo

que o conjunto de corte F tem peso igual a 2. Qualquer outro corte que separa os vértices s e t no grafo G tem um peso maior do que o corte ilustrado, sendo esse o motivo dele ser considerado o corte mínimo que separa os vértices s e t.

Existem problemas de natureza mais prática que podem ser entendidos como o problema do corte s-t mínimo. Suponha que uma empresa petrolífera possua uma rede de dutos espalhados pelo território de um país que interligam estações de distribuição de petróleo. Cada duto possui uma capacidade máxima de escoamento de petróleo. Existem nessa rede duas estações principais: uma que libera o fluxo de petróleo pela rede e outra que recebe todo petróleo excedente. Ao aumentar o fluxo pela rede, alguns dutos em especial limitarão a capacidade de vazão da rede como um todo. Esses dutos são chamados de *gargalo* da rede. O problema de encontrar o fluxo máximo em uma rede de escoamento é conhecido como *problema do fluxo máximo*.

O problema do fluxo máximo tem grande relação com o problema do corte s-t mínimo. Um resultado famoso provado independentemente em [12] e em [15] mostra que a capacidade do fluxo máximo da rede é igual ao peso do corte mínimo. Assim sendo, os dutos devolvidos como resposta de um algoritmo que resolve o problema do corte s-t mínimo são também aqueles que são os gargalos na rede de fluxo de petróleo. Os pesos correspondem às capacidades de cada duto, s corresponde à estação que libera o fluxo, t corresponde à estação que recebe o fluxo excedente e o conjunto de corte F é o gargalo.

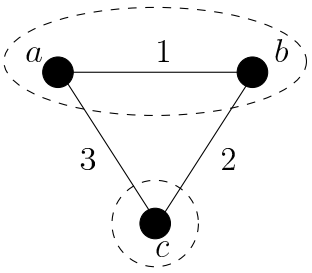
O problema do fluxo máximo possui algoritmos polinomiais que o solucionam, sendo um deles o algoritmo de Edmonds-Karp [9]. Esse algoritmo executa em tempo $O(nm^2)$

onde n é o número de vértices e m é o número de arestas. A prova do teorema sobre fluxo máximo e corte mínimo, bem como a análise do algoritmo de Edmonds-Karp, que também pode ser utilizado para solucionar o problema do corte s-t mínimo, não são apresentadas no presente trabalho e podem ser encontradas em [9]. Orlin apresentou um algoritmo otimizado em [29] que soluciona o problema do fluxo máximo em tempo $O(nm)$. Apesar do problema do corte s-t mínimo não ser NP-difícil, faz-se necessário o seu estudo pelo fato do algoritmo apresentado na seção 4.2.1 utilizar o algoritmo para o problema do corte s-t mínimo como sub-rotina.

2.1.2 Corte máximo

Problema 2.2 (definição encontrada em [37])**.** *No* problema do corte máximo *(maximum cut problem) tem-se como entrada um grafo não direcionado* $G = (V, E)$ *e um peso não negativo* w_e *para cada aresta* $e \in E$*. O objetivo é particionar o conjunto de vértices* V *em dois subconjuntos* U *e* $W = V - U$ *de forma que seja maximizada a soma dos pesos das arestas que separam os dois subconjuntos.*

Karp provou em 1972 que esse e outros 20 problemas são NP-difíceis [23]. A Figura 2.2 ilustra a explicação dada, onde as arestas que fazem parte do conjunto de corte F são $\{a, c\}$ e $\{b, c\}$ e seu peso é igual a 5.



$$U = \{a, b\} \quad W = \{c\}$$
$$F = \{\{a, c\}, \{b, c\}\}$$

Figura 2.2: Corte máximo de um grafo

O problema do corte máximo pode ser aplicado, por exemplo, para criar grupos correlatos de pessoas em uma rede social. Cria-se um grafo que relaciona cada pessoa da rede com seus amigos, sendo que cada vértice corresponde a uma pessoa e cada aresta

corresponde a uma relação de amizade entre elas. O peso das arestas determina o grau de proximidade entre as pessoas. Quanto menor o valor do peso da aresta, mais próximas as pessoas são. Ao executar o algoritmo do corte máximo nesse grafo, o algoritmo retornará dois grupos de pessoas que possuem mais proximidade. Ou seja, o algoritmo separa o grafo em dois grupos por grau de proximidade. Caso seja executado o mesmo algoritmo novamente em cada partição do grafo após o corte, cada partição é novamente subdividida em grupos de pessoas por grau de proximidade. Assim, é possível de separar um conjunto de pessoas em diferentes grupos de pessoas que são mais próximas entre si.

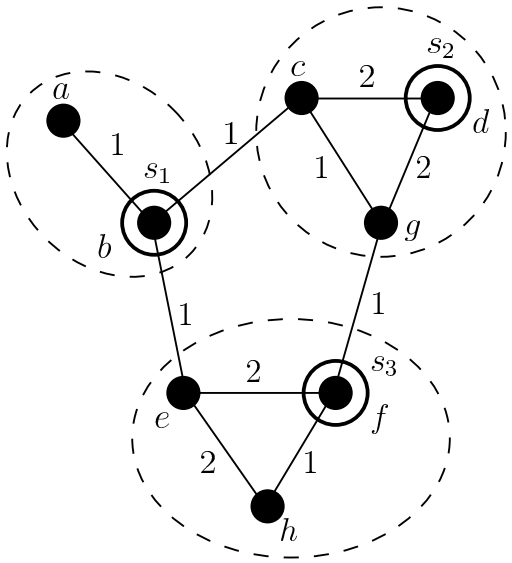
2.1.3 Corte multisseparador mínimo

Problema 2.3 (definição encontrada em [37])**.** *No* problema do corte multisseparador mínimo *(multiway cut problem) é dado como entrada um grafo não direcionado* $G = (V, E)$*, custos não negativos* c_e *para cada aresta* $e \in E$ *e* k *vértices selecionados* $s_1, \ldots, s_k \in V$*. O objetivo é particionar o conjunto de vértices* V *em* k *subconjuntos disjuntos* $C_1, \cdots, C_k$ *de forma que cada vértice selecionado* s_i *pertença a um subconjunto* C_i*. Além disso, o* k*-corte encontrado deve ter o custo mínimo.*

Como existe um subconjunto para cada vértice selecionado, nesse problema busca-se não apenas uma bipartição do conjunto de vértices V (como por exemplo, no problema do corte máximo), mas uma partição em k subconjuntos. Esse problema também é NP-difícil, como provado por Dahlhaus, Johnson, Papadimitriou, Seymour e Yannakakis em 1994 [11].

A Figura 2.3 mostra um corte de custo 3 que particiona o conjunto de vértices V em três subconjuntos disjuntos diferentes: um que contém o vértice escolhido s_1, outro que contém o vértice s_2 e outro que contém o vértice s_3. Percebe-se que, caso as arestas pertencentes ao conjunto de corte F sejam removidas do grafo G, é criado um grafo resultante G_r que possui cada vértice selecionado ligado a um componente conexo diferente.

Esse problema tem aplicações na área de computação distribuída, como apresentado por [21]. Nesse caso, cada vértice do grafo representa um objeto que deve ser armazenado em um computador. O custo de cada aresta do grafo representa a quantidade de



$$F = \{\{b, c\}, \{b, e\}, \{f, g\}\}$$

Figura 2.3: Corte multisseparador mínimo de um grafo

comunicação necessária entre dois objetos. Todos os objetos precisam ser distribuídos em k computadores, em especial os objetos s_i que devem estar armazenados obrigatoriamente no computador i. Se dois objetos estão alocados em um mesmo computador, a comunicação entre eles é desconsiderada pois evita mais tráfego na rede. O objetivo é distribuir os objetos nos k computadores de forma que a comunicação entre os computadores seja minimizada. Cada subconjunto C_i encontrado após a solução do problema lista os objetos que estarão armazenados em cada computador i e a soma dos custos das arestas que interligam dois computadores representa a quantidade de comunicação entre eles.

2.1.4 Corte mais disperso

Problema 2.4 (definição encontrada em [37])**.** *No* problema do corte mais disperso *(sparsest cut problem) é dado como entrada um grafo $G = (V, E)$ não direcionado, valores não negativos de capacidades c_e para cada aresta $e \in E$ e k pares de vértices selecionados (s_i, t_i) com demandas não negativas associadas d_i, sendo s_i e $t_i \in V$ para todo $1 \leq i \leq k$. Seja $F = \delta(S)$ o conjunto de corte formado por todas as arestas em E com somente um vértice em S, o objetivo é encontrar dois conjuntos de vértices $S \subseteq V$ e $V - S$ que*

minimiza a razão $\rho(S)$ apresentada na equação 2.1.

$$\rho(S) = \frac{\sum_{e \in \delta(S)} c_e}{\sum_{i:|S \cap \{s_i, t_i\}|=1} d_i} \tag{2.1}$$

Em outras palavras, o problema do corte mais disperso é aquele que minimiza a razão da capacidade total das arestas do conjunto de corte pela soma das demandas separadas por ele. Analisando a equação 2.1, busca-se selecionar as arestas do conjunto de corte $F = \delta(S)$ que minimizam a capacidade total e, ao mesmo tempo, busca-se maximizar a quantidade de demanda separada. Assim sendo, o melhor corte é aquele que possui a melhor razão de capacidades das arestas do conjunto de corte por demandas separadas. Nem todos os pares selecionados (s_i, t_i) obrigatoriamente precisam ser separados pelo corte, mas pelo menos a separação de um par é indispensável, pois senão o denominador seria igual a 0.

O fluxo de cada demanda i, da origem s_i até o destino t_i, pode gerar um número exponencial de caminhos possíveis no grafo. Porém, o objetivo do problema não está em encontrar tais caminhos, mas sim em identificar arestas importantes do grafo para realizar o fluxo total. Esse é um problema que busca a maior relação custo-benefício, existindo, assim, um cálculo de custo em função do valor da capacidade total do corte. Se minimizar a capacidade, minimizará o custo. Para minimizar a função objetivo, deve-se minimizar o numerador e maximizar o denominador. De forma geral, o objetivo do problema pode ser entendido como quebrar os caminhos no grafo que geram o maior fluxo com o mínimo de custo para quebrá-los. Algumas vezes a escolha de uma aresta a mais como pertencente às arestas do conjunto de corte pode separar mais uma demanda, mas o custo-benefício geral para separá-la é aumentado, inviabilizando a separação.

Esse é um problema NP-difícil [16] e pode ser aplicado com o objetivo de identificar vias críticas que interligam um conjunto de várias cidades. Vias essas que, se estiverem congestionadas, causam um sério desligamento entre um conjunto de cidades consideradas cidades polo. Para esse exemplo, o conjunto de arestas E do grafo G corresponde às vias automobilísticas, o conjunto de vértices V corresponde às várias cidades interligadas e cada via e possui uma capacidade de fluxo de automóveis por minuto c_e. São presseleci-

onados i pares de cidades polo que precisam ser interligadas (s_i, t_i) e é associada a cada par uma demanda de fluxo de carros d_i que tem origem em uma cidade s_i e destino em outra cidade t_i. Quanto maior o valor da demanda entre um par de cidades selecionadas, maior é a importância desse par de cidades permanecer interligado. Um governo federal ou estadual precisa identificar um conjunto de vias críticas $\delta(S)$ que, se forem danificadas de forma a interromper o trânsito que por elas passa, o fluxo de carros que transita com origem e destino nas diferentes cidades polo decai substancialmente. Como o objetivo é encontrar um conjunto limitado de vias críticas, busca-se minimizar o fluxo cortado total de automóveis caso as vias críticas sejam interrompidas ($\sum_{e \in \delta(S)} c_e$). Porém, busca-se, ao mesmo tempo, maximizar a quantidade de fluxo cortado de automóveis que transitam com origem e destino nas cidades polo ($\sum_{i:|S \cap \{s_i,t_i\}|=1} d_i$). Minimizar a razão entre o fluxo total cortado pelas vias críticas e o fluxo cortado entre as cidades polo garante a separação dos principais pares de cidades polo com a escolha do menor número de vias.

2.2 Algoritmos de aproximação

A Definição 2.1 apresenta o conceito formal de algoritmos de aproximação.

Definição 2.1. *Um* algoritmo α-aproximado *para um problema de otimização é um algoritmo polinomial que, para todas as instâncias do problema, ele produz uma solução cujo valor está em um fator α em comparação ao valor da solução ótima do problema.*

O fator α é denominado de *garantia de aproximação* do algoritmo, mas também pode ser chamado como *relação de aproximação* ou *fator de aproximação*. A garantia de aproximação de um algoritmo de aproximação expressa a proximidade entre o valor de uma resposta qualquer gerada pelo algoritmo e o valor da resposta ótima do problema. Os valores da garantia de aproximação são maiores do que 1 para problemas de minimização e menores do que 1 para problemas de maximização. Dessa forma, quando o seu valor é igual a 1, independente se o problema for de maximização ou de minimização, então o algoritmo produz uma solução ótima.

Para qualquer solução criada por um algoritmo de aproximação, a garantia de apro-

ximação informa o valor limitante da relação entre essa solução e a solução ótima. Para um melhor entendimento, seja $S(I)$ o valor de uma solução devolvida por um algoritmo de aproximação e $OPT(I)$ o valor da solução ótima. Caso o problema seja de maximização então a inequação 2.2 é usada, caso seja de minimização, a inequação 2.3 é usada.

$$\frac{S(I)}{OPT(I)} \geq \alpha, \ \forall I \in \mathcal{I} \tag{2.2}$$

$$\frac{S(I)}{OPT(I)} \leq \alpha, \ \forall I \in \mathcal{I} \tag{2.3}$$

Por exemplo, com base na Definição 2.1, um algoritmo $\frac{1}{2}$-aproximado é um algoritmo que soluciona um problema de maximização, que executa em tempo polinomial e gera uma solução que, para qualquer instância, tem valor de no mínimo metade do valor da solução ótima. Da mesma forma, um algoritmo 2-aproximado é um algoritmo que soluciona um problema de minimização, que executa em tempo polinomial e que, para qualquer instância, gera uma solução cujo valor é no máximo o dobro do valor da solução ótima.

As principais características que diferenciam os métodos heurísticos dos algoritmos de aproximação estão justamente na garantia de aproximação e no tempo de execução. Nos métodos heurísticos não existe essa necessidade de saber a proximidade de uma resposta nem quanto tempo o algoritmo fica executando até encontrá-la. A determinação desses dois fatores são indispensáveis para que um algoritmo seja considerado um algoritmo de aproximação. Nos algoritmos de aproximação, existe uma garantia de aproximação e o algoritmo precisa executar em tempo polinomial no tamanho da entrada.

O valor da garantia de aproximação é encontrado sempre com base no pior caso de execução do algoritmo. Ou seja, mesmo quando a maior parte das instâncias são mais próximas da resposta ótima, matematicamente se assume a execução do pior caso como base para o cálculo da garantia de aproximação. Isso faz com que, na prática, muitos algoritmos devolvam uma resposta que muitas vezes é mais próxima da solução ótima do que a própria garantia de aproximação obtida de maneira teórica e analítica.

2.3 Modelos de programação matemática

São apresentados a seguir os conceitos dos três tipos de programas matemáticos utilizados nesse trabalho: o programa linear, o programa semidefinido e o programa vetorial.

2.3.1 Programação linear

O principal objetivo da programação linear é encontrar um vetor não negativo x com valores em $\mathbb{Q}$ que minimiza uma dada função objetivo de x. Essa função objetivo deve respeitar um conjunto de restrições lineares que também são escritas em função de x. Formalmente falando, dados um vetor c de tamanho n com valores pertencentes a $\mathbb{Q}$, um vetor b de tamanho m com valores pertencentes a $\mathbb{Q}$ e uma matriz $A = (a_{ij})$ de dimensão $m \times n$ com valores pertencentes a $\mathbb{Q}$, a *solução ótima* do programa linear

$$\begin{aligned} \text{minimizar} \quad & \sum_{j=1}^{n} c_j x_j & & \\ \text{sujeito a} \quad & \sum_{j=1}^{n} a_{ij} x_j \geq b_i, & i = 1, \ldots, m, & \\ & x_j \geq 0, & j = 1, \ldots, n & \end{aligned} \tag{2.4}$$

é um vetor x de tamanho n que minimiza a função objetivo linear $\sum_{j=1}^{n} c_j x_j$ sujeito ao conjunto de restrições também lineares $\sum_{j=1}^{n} a_{ij} x_j \geq b_i$, para $i = 1, \ldots, m$ e $x_j \geq 0$, para $j = 1, \ldots, n$. Cada linha do programa linear referente a uma restrição é na verdade um conjunto de restrições. Cada restrição desse conjunto é um somatório de elementos que é normalmente composto de n elementos, ou seja, a quantidade de células existentes no vetor x.

Existem algoritmos eficientes na prática que são capazes de resolver programação linear com milhares de variáveis e restrições. O método *Simplex* é um exemplo de algoritmo capaz de resolver programação linear que, apesar de demandar tempo exponencial no pior caso, possui um desempenho prático satisfatório na maior parte das instâncias. Mais conceitos sobre o método Simplex podem ser encontrados em [30]. Os métodos de *pontos interiores* são algoritmos polinomiais que também podem ser aplicados, esses abordados

com mais ênfase por Nesterov e Nemirovskii [28]. Um desses métodos é o *método dos elipsoides*, esse abordado melhor em [18]. Esses métodos não são explicados no presente trabalho; para nosso objetivo basta saber que eles existem e que são executados em tempo polinomial.

O vetor x é chamado de *variável* do programa linear. A variável de um programa linear é um vetor de valores numéricos pertencentes ao conjunto dos números racionais. É necessário que o conjunto de valores de x satisfaça todas as restrições. Qualquer valoração de x que satisfaz todas as restrições é chamada de *resposta* ou *solução factível*. Caso exista pelo menos uma valoração de x que satisfaça todas as restrições do programa linear, então o programa é dito como *factível*.

Existem casos também em que é impossível encontrar uma valoração de x de forma que todas as restrições impostas pelo programa linear sejam satisfeitas. Isso acontece quando uma restrição inviabiliza outras, fazendo com que pelo menos uma delas não seja satisfeita, independente da valoração de x. Quando isso acontece, o programa linear é *infactível*.

Quando é encontrada uma solução factível que resulta no melhor valor possível para a função objetivo, então a solução factível encontrada é uma *solução ótima*. Quando uma solução ótima foi encontrada, o programa linear foi *solucionado*.

O modelo de programa linear apresentado na formulação 2.4 é conhecido como *forma canônica*. Existem variações do modelo de programação linear apresentado, como os casos onde se busca maximizar a função objetivo ao invés de minimizá-la, onde existem restrições que são equações ao invés de inequações e onde x pode conter valores negativos. No entanto, o modelo do programa linear apresentado é geral o suficiente para representar todas essas variações, como mostrado a seguir.

Um programa linear que maximiza a função objetivo $\sum_{j=1}^{n} c_j x_j$ pode ser reescrito como a minimização da função $-\sum_{j=1}^{n} c_j x_j$. No segundo caso, uma equação $\sum_{j=1}^{n} a_{ij} x_j = b_i$ pode ser expressa como um par de inequações $\sum_{j=1}^{n} a_{ij} x_j \geq b_i$ e $-\sum_{j=1}^{n} a_{ij} x_j \geq -b_i$. Por último, uma componente x_j do vetor x que possa conter valores negativos pode ser expressa em termos de duas variáveis não negativas x_j^+ e x_j^-, substituindo x_j por $x_j^+ - x_j^-$ na função

objetivo e nas restrições.

Existe também outra variação de programas lineares conhecida como *programação linear inteira*. Essa variação requer que todos os valores das componentes do vetor x sejam inteiros. Assim, por exemplo, é possível declarar x_j como pertencente aos naturais ou ser limitado a um subconjunto de valores inteiros como $x_j \in \{-1, 0, 1\}$. Porém, ao contrário da programação linear comum, onde já foram descritos algoritmos polinomiais, ainda não são conhecidos algoritmos polinomiais para solucionar programação linear inteira.

De fato, o problema de solucionar programação linear inteira é conhecido como sendo um problema de otimização NP-difícil [16]. Apesar disso a programação linear inteira é uma ferramenta usual para modelar de forma compacta problemas de otimização combinatória. Além disso, existem inúmeros problemas que podem ser solucionados por algoritmos de aproximação polinomiais que fazem uso das propriedades da programação linear inteira.

2.3.2 Programação semidefinida

A programação semidefinida é similar à programação linear, possuindo também uma função objetivo e um conjunto de restrições. A principal diferença entre as duas se encontra nas restrições adicionais que tornam a solução do programa uma *matriz positiva simétrica semidefinida*. Como será explicado a seguir, parte das restrições adicionais são funções não lineares, fazendo com que, de maneira geral, todo programa semidefinido seja um programa não linear.

Programas semidefinidos ainda não possuem algoritmos que os solucionam em tempo polinomial. Um dos motivos disso também se dá pelo fato dos valores exatos das soluções poderem ser irracionais. Porém existe o método dos elipsoides que soluciona programas semidefinidos em tempo polinomial com um fator de erro ϵ, abordado por Grötschel, Lovász e Schrijver [18]. O método dos elipsoides não será estudado no trabalho e será assumido que o erro ϵ da solução devolvida pelo algoritmo aumenta uma parcela desprezível na garantia de aproximação do algoritmo que o utiliza como sub-rotina [37].

$$\begin{aligned} \text{minimizar ou maximizar} \quad & \sum_{i,j} c_{ij}x_{ij} \\ \text{sujeito a} \quad & \sum_{i,j} a_{ijk}x_{ij} = b_k, \quad \forall k, \\ & x_{ij} = x_{ji}, \qquad \forall i,j, \\ & X = (x_{ij}) \succeq 0. \end{aligned} \tag{2.5}$$

No modelo de programa semidefinido apresentado, as variáveis x_{ij} são definidas para $1 \leq i,j \leq n$. Pelo fato das variáveis de um programa semidefinido formarem uma matriz simétrica, a matriz das variáveis também é quadrada. Existe um conjunto de k restrições que modela o problema a ser solucionado. Isso faz com que seja necessária a especificação de um vetor tridimensional a que especifica todo o conjunto de restrições. Esse vetor armazena o coeficiente de todas as células de cada matriz de variáveis, células essas presentes em cada restrição do conjunto de restrições.

O conjunto de restrições que torna a solução uma matriz positiva simétrica semidefinida é formado pelos dois últimos conjuntos de restrições do programa 2.5. Para que a matriz resultante seja *simétrica*, existem as restrições $x_{ij} = x_{ji}$ para todo i e j que impõem tal propriedade. O último conjunto de restrições, que possui a notação $X = (x_{ij}) \succeq 0$, impõe que a matriz X seja *positiva semidefinida*, finalizando a modelagem do programa semidefinido. Dessa forma, a solução do programa resulta obrigatoriamente em uma *matriz positiva simétrica semidefinida*.

Faz-se necessário apresentar mais algumas notações para um melhor entendimento do que se trata uma matriz positiva simétrica semidefinida, essa apresentada na Definição 2.2. A notação X^T corresponde à matriz transposta da matriz X. Os vetores $v \in \mathbb{R}^n$ são vetores de colunas (verticais) com tamanho n e que armazenam valores reais. A notação $v^T v$ representa o valor real resultante do produto escalar entre o vetor coluna v transposto e ele mesmo. Já a notação vv^T é a multiplicação do vetor coluna v com ele mesmo transposto, resultando em uma matriz de dimensão $n \times n$.

Definição 2.2 (apresentada em [37]). *Uma matriz $X \in \mathbb{R}^{n \times n}$ é uma matriz positiva*

semidefinida se para todo $y \in \mathbb{R}^n$, $y^T X y \geq 0$.

Como o programa semidefinido 2.5 impõe que a solução resulte em uma matriz positiva simétrica semidefinida, então será assumido no decorrer do texto que toda matriz positiva semidefinida é simétrica também. Dessa forma, $X = X^T$. Toda matriz positiva simétrica semidefinida possui algumas propriedades listadas no Teorema 2.1, provado em [36].

Teorema 2.1 (provado em [36])**.** *Seja* X *uma matriz simétrica de valores reais e dimensão* $n \times n$*, então as seguintes afirmações são equivalentes:*

1. X *é positiva semidefinida;*
2. *Todos os autovalores de* X *são não negativos;*
3. *Existe uma matriz* V *de valores reais e dimensão* $n \times n$ *de forma que* $X = V^T V$.

É possível decompor uma matriz positiva simétrica semidefinida X em tempo polinomial de tal forma que $X = U \cdot D \cdot U^T$ por meio da *decomposição de Cholesky* [36]. Após essa decomposição, a matriz D é diagonal e possui o elemento D_{ii} como sendo o i-ésimo autovalor de X. É possível obter uma aproximação das matrizes V^T e V, calculando em tempo polinomial $V^T = U \cdot D^{\frac{1}{2}}$ e $V = D^{\frac{1}{2}} \cdot U^T$. Os cálculos das matrizes V^T e V exigem que sejam realizadas operações de raiz quadrada. Como X é simétrica e positiva semidefinida, então todos os autovalores de X são não negativos, existindo assim a possibilidade de calcular a raiz quadrada. Os cálculos das matrizes V^T e V resultam em uma aproximação pelo fato da decomposição $X = V^T V$ poder resultar em componentes com valores irracionais.

Como toda solução de um programa semidefinido é uma matriz positiva simétrica semidefinida, então os itens do Teorema 2.1 podem ser utilizados para comprovar que existe uma outra formulação equivalente à programação semidefinida: a *programação vetorial*.

2.3.2.1 Programação vetorial

A programação vetorial é um tipo de programação não linear cujas variáveis são vetores unidimensionais $v_i \in \mathbb{R}^n$ e as restrições e função objetivo são lineares sobre o produto escalar entre os vetores v_i. O valor n define tanto a dimensão dos vetores como a quantidade de vetores variáveis. No modelo de programação vetorial a seguir, o produto escalar entre v_i e v_j é denotado por $v_i \cdot v_j$.

$$\begin{aligned} \text{minimizar ou maximizar} \quad & \sum_{i,j} c_{ij}(v_i \cdot v_j) & \\ \text{sujeito a} \quad & \sum_{i,j} a_{ijk}(v_i \cdot v_j) = b_k, & \forall k, \\ & v_i \in \mathbb{R}^n, & i = 1, \ldots, n. \end{aligned} \tag{2.6}$$

A equivalência entre o programa vetorial 2.6 e o programa semidefinido 2.5 é provada pelo Teorema 2.2.

Teorema 2.2. *O programa vetorial 2.6 é equivalente ao programa semidefinido 2.5.*

Demonstração. Será mostrado que para cada solução factível do programa 2.6, existe uma solução factível para o programa 2.5 com o mesmo valor da solução e que o contrário também é verdade. Seja os vetores $v_1, \ldots, v_n$ uma solução factível para o programa vetorial 2.6. Seja V a matriz que tem suas colunas formadas por $v_1, \ldots, v_n$. Calcule a matriz $X = V^T V$. Como X é o resultado da multiplicação entre a matriz V e a sua transposta, então a matriz X é simétrica. Utilizando a terceira afirmação do Teorema 2.1, como a matriz X é simétrica e existe uma matriz V de forma que $X = V^T V$, então X é positiva semidefinida. Como X é simétrica e positiva semidefinida, então nos resta mostrar a equivalência das restrições restantes e da função objetivo. Como $X = V^T V$ e as colunas de V são formadas pelos vetores $v_1, \ldots, v_n$, então $X_{ij} = v_i \cdot v_j$. Como X é uma solução factível para o programa 2.5 e $X_{ij} = v_i \cdot v_j$, então $\sum_{i,j} c_{ij} X_{ij} = \sum_{i,j} c_{ij}(v_i \cdot v_j)$, fazendo com que as funções objetivos de 2.5 e 2.6 resultem no mesmo valor. Também é possível observar a equivalência entre as restrições $\sum_{i,j} a_{ijk} X_{ij} = b_k$ e $\sum_{i,j} a_{ijk}(v_i \cdot v_j) = b_k$.

Realizando o caminho inverso, seja a matriz X uma solução factível para o programa semidefinido 2.5. Como X é uma matriz positiva simétrica semidefinida, então existe uma matriz V tal que $X = V^T V$. Tal matriz V pode ser obtida em tempo polinomial pela decomposição de Cholesky, como explicado anteriormente. Seja os vetores $v_1, \ldots, v_n$ as colunas de V e a solução do programa 2.6. Como $X = V^T V$, então $X_{ij} = v_i \cdot v_j$, fazendo com que as funções objetivos de 2.5 e 2.6 resultem no mesmo valor. Como existem soluções factíveis com o mesmo valor da solução para ambos os programas, então eles são equivalentes. □

É importante informar que tal equivalência é verdade somente quando as matrizes resultantes forem positivas simétricas semidefinidas. Como dito anteriormente, será assumido que toda matriz positiva semidefinida é simétrica também. O termo "programa vetorial" será abreviado em alguns momentos como *PV*.

2.4 Espaços métricos

Um *espaço métrico* é definido por uma tupla (V, d), sendo V um conjunto de elementos e d uma função sobre os elementos pertencentes a V. A função d é denominada *função de distância* e é responsável por atribuir um valor real positivo a todos os subconjuntos de dois elementos $\{i, j\}$ possíveis de serem formados com os elementos pertencentes ao conjunto V. A função de distância é caracterizada principalmente por não possuir valores negativos e por ser uma função simétrica, ou seja, o mesmo valor de distância é atribuído aos pares ordenados (i, j) e (j, i), de forma que seja desprezível a ordem dos elementos em um par. Não obstante, uma função de distância deve resultar no valor 0 caso o par seja formado por elementos iguais. Segue a definição formal de função de distância.

Definição 2.3. *Dado um conjunto V, uma* função de distância *sobre V é uma função $d : V \times V \to \mathbb{R}_+$ que é simétrica e satisfaz a restrição $d(i, i) = 0$, para todo $i \in V$.*

Outro requisito para que um espaço seja considerado um espaço métrico é a necessidade que a função de distância seja uma *métrica*, sendo essa definição explicada logo adiante.

2.4.1 Métricas

Alguns algoritmos apresentados utilizam o conceito de um tipo mais específico de função de distância: as métricas. Segue a definição matemática de métricas.

Definição 2.4 (apresentada em [26]). *Uma métrica d é uma função $d : V \times V \to \mathbb{R}_+$ tal que:*

- $d(i, j) = 0$ *se e somente se* $i = j$;
- $d(i, j) = d(j, i)$ *para todos* i *e* $j \in V$;
- $d(i, j) \leq d(i, k) + d(k, j)$ *para todos* i, j *e* $k \in V$.

Existe também o conceito de *semimétrica*, ou *pseudométrica*, que é similar ao da Definição 2.4, com exceção que a primeira propriedade não necessariamente é obedecida. Dessa forma, se existe uma distância $d(i, j) = 0$, então não necessariamente os vértices i e j fazem referência ao mesmo vértice ($i = j$ ou $i \neq j$). A distinção entre métricas e semimétricas será ignorada, sendo as duas tratadas no presente trabalho unicamente como métricas. Com o objetivo de simplificar a notação, no decorrer do trabalho será utilizada a notação d_{ij} como sendo equivalente a $d(i, j)$.

A Definição 2.4 afirma que uma métrica define as distâncias entre cada par de vértices pertencentes a V. Dessa forma, uma métrica pode também ser representada por um grafo completo não orientado com pesos maiores que 0 em cada aresta. O peso de uma aresta $\{i, j\}$ representa a distância d_{ij} entre os vértices i e $j \in V$ na métrica d.

Existe uma forma usual de modelar problemas de cortes em grafos com o uso de métricas. Seja S um conjunto disjunto pertencente a um corte do grafo $G = (V, E)$ de forma que $S \subseteq V$, então pode-se definir uma métrica d de forma que $d_{uv} = 1$ se e somente se $u \in S$ e $v \notin S$ ou $v \in S$ e $u \notin S$, e $d_{uv} = 0$ caso contrário. Ou seja, a distância entre dois vértices é igual a 1 se eles estão separados pelo corte, e a distância entre eles é igual a 0 caso contrário. Dessa forma, o peso de arestas que fazem parte do conjunto de corte é igual a 1 e o peso das arestas restantes é igual a 0. Assim, um problema no grafo G com objetivo de maximizar ou minimizar um corte calculado pela soma dos pesos w_e das

arestas $e \in E$ pode ser solucionado ao encontrar uma métrica d, calculada por algoritmos que resolvem programação linear. Esse problema pode ser entendido como o problema de encontrar uma métrica d, representada pelas variáveis d_{uv} de um programa linear com objetivo de maximizar ou minimizar a função objetivo $\sum_{e=\{u,v\}\in E} w_e d_{uv}$.

A modelagem de problemas em programas matemáticos inteiros, como no exemplo supracitado, não implica em uma solução polinomial. O problema de programação matemática inteira é NP-difícil. Para solucionar isso, relaxa-se o programa matemático para que ele possa assumir valores fracionários. Dessa forma, é possível solucioná-lo com algoritmos polinomiais. São apresentados no Capítulo 4 relaxamentos de programas inteiros de forma que o problema seja modelado por um programa cujas restrições mantém as variáveis com as propriedades de métricas.

2.4.2 Espaços métricos finitos

Os espaços métricos podem ser classificados em dois tipos distintos: os *espaços métricos finitos* e os *espaços métricos infinitos*. A principal diferença entre eles está na cardinalidade do conjunto V da tupla (V, d). Nos espaços métricos finitos, o conjunto de vértices V é finito e formado por n elementos. Nesses tipos de espaços métricos, a função de distância d pode ser especificada por um conjunto de $\binom{n}{2}$ valores reais positivos dispostos em uma matriz de dimensão $n \times n$, como no exemplo mostrado na Tabela 2.1. Para o mesmo exemplo, é definida uma função de distância sobre o conjunto de elementos $V = \{v_1, v_2, v_3, v_4\}$.

d	v_1	v_2	v_3	v_4
v_1	0	1	2	6
v_2		0	3	7
v_3			0	4
v_4				0

Tabela 2.1: Função de distância d de um espaço métrico finito

Cada célula da matriz que define a função de distância d faz referência a um par de elementos pertencentes a V. A mesma matriz é capaz de determinar todos os pares possíveis de serem formados, definindo com completude todas as distâncias existentes

no espaço métrico finito. Pelo fato da função de distância ser simétrica, os elementos localizados abaixo da diagonal principal da matriz podem ser desconsiderados. Da mesma forma, como a matriz representa uma função de distância, a diagonal principal é formada unicamente por elementos zerados, sendo que essa diagonal referencia todos os pares de elementos que se repetem (v_i, v_i).

Existe uma correspondência entre espaços métricos finitos e grafos. É possível obter um espaço métrico a partir de um grafo e o contrário também é possível. Dado um grafo $G = (V', E)$ com n vértices ligados por arestas com pesos, o espaço métrico (V, d), obtido por meio do grafo G, é composto pelo conjunto $V = V'$ e pela métrica d definida como sendo os valores de tamanho dos menores caminhos que interligam cada par de vértices i e $j \in V'$. A métrica d, apresentada na Tabela 2.1, é obtida por meio do grafo G, ilustrado na Figura 2.4.

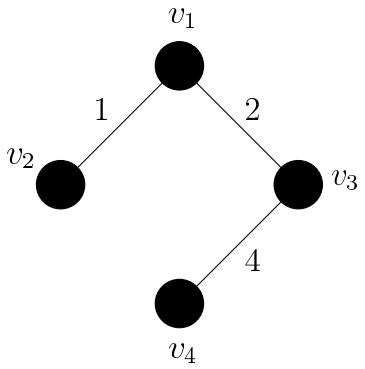


Figura 2.4: Grafo G correspondente ao espaço métrico (V, d)

Consecutivamente, é possível obter o grafo G por meio do espaço métrico (V, d). Para isso, cria-se um grafo completo de forma que o peso de cada aresta $\{v_i, v_j\}$ seja igual à distância $d_{v_i v_j}$. A Figura 2.5 ilustra o grafo recém-criado por meio da métrica definida na Tabela 2.1. Apesar do grafo ilustrado na Figura 2.5 representar o espaço métrico (V, d), ele ainda não é isomorfo ao grafo G, ilustrado na Figura 2.4. De fato, é possível simplificar o grafo obtido removendo arestas excedentes que, caso sejam retiradas, não mudam distância alguma na métrica d. As arestas possíveis de serem removidas do grafo ilustrado na Figura 2.5 sem que d seja modificada, são: $\{v_1, v_4\}$, $\{v_2, v_3\}$ e $\{v_2, v_4\}$. Após a remoção dessas arestas, obtém-se isomorfismo entre os grafos e a equivalência entre o grafo G e o espaço métrico (V, d). O grafo resultante da remoção de todas as arestas

excedentes é chamado de *grafo crítico*.

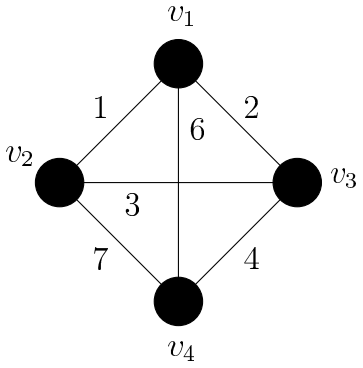


Figura 2.5: Grafo completo obtido por meio do espaço métrico (V, d)

2.4.3 Espaços métricos infinitos

O conjunto de vértices V de um espaço métrico infinito (V, d) pode ser associado aos infinitos pontos existentes em um espaço vetorial. Normalmente, este conjunto é definido por um conjunto $\mathbb{R}^k$, sendo k um valor inteiro maior ou igual a 1. A função de distância d normalmente é definida com alguma *norma de Minkowski* ℓ_p, sendo $1 \leq p \leq +\infty$. Uma norma é uma função que associa um vetor do conjunto V a um valor real não negativo denominado *comprimento*. Uma norma de Minkowski define a função de distância pela fórmula 2.7, onde $v \in V$ e $\|v\|_p$ corresponde ao comprimento de v na norma ℓ_p. A distância entre dois vértices v_1 e $v_2 \in V$ é calculada como $\|v_1 - v_2\|_p$.

$$\|v\|_p = \left(\sum_{i=1}^{k} |v_i|^p \right)^{1/p} \tag{2.7}$$

Algumas normas de Minkowski são familiares, como o caso da norma ℓ_1, também conhecida como distância de Manhattan, e a norma ℓ_2, também conhecida como distância Euclidiana. Qualquer norma de Minkowski obedece a propriedade da desigualdade triangular. Assim, qualquer função de distância que seja uma norma de Minkowski é também uma métrica. O espaço métrico $(\mathbb{R}^3, \ell_2)$ é um exemplo de espaço métrico infinito definido pelo espaço vetorial Euclidiano em três dimensões. Será assumido ℓ_p^n como sendo o espaço métrico infinito definido pelo conjunto infinito de pontos $\mathbb{R}^n$ e pela norma de Minkowski ℓ_p, ou seja, $(\mathbb{R}^n, \ell_p)$.

CAPÍTULO 3

IMERSÕES MÉTRICAS

3.1 Imersões

No estudo de espaços métricos normalmente faz-se necessário comparar diferentes espaços métricos. As *imersões* provêm uma forma matemática de mapear um espaço métrico em outro. A Definição 3.1 formaliza os conceitos de *função de imersão* e de imersão *isométrica*.

Definição 3.1. *Dados dois espaços métricos* (V, d) *e* (V', d')*, imersão é uma função* $f : V \to V'$*. Uma imersão é isométrica se para todo* v_1 *e* $v_2 \in V$*,* $d(v_1, v_2) = d'(f(v_1), f(v_2))$*.*

De forma geral, uma *função de imersão* converte um vértice pertencente a um espaço métrico (V, d) em um vértice correspondente em outro espaço métrico (V', d'). Existem inúmeras aplicações do conceito de imersão, uma delas é a imersão das diferentes posições geográficas, formadas por um valor numérico de latitude e um de longitude, do globo terrestre no planisfério (mapa planar). Nesse exemplo, tem-se o globo terrestre representado pelo espaço métrico formado pelos infinitos pontos localizados na superfície de um esferoide e o planisfério representado pelo espaço métrico formado pelos infinitos pontos localizados no mapa planar. Outro exemplo comum é a imersão das diferentes cores do sistema RGB, utilizado em monitores e televisores, no sistema de cores CMYK, utilizado na impressão física de documentos coloridos. Nesse exemplo, as diferentes cores no sistema RGB são representadas por três valores reais que variam de 0 a 255 (inclusive), sendo que uma coordenada representa a quantidade de cor vermelha, outra representa a quantidade de cor verde e a outra representa a quantidade de cor azul. Assim sendo, as diferentes cores, formadas por três valores, pertencem a um cubo tridimensional maciço de lado igual a 255. Já no sistema CMYK, as cores são representadas por um conjunto de

quatro valores reais que variam de 0 a 1 (inclusive), sendo que esses valores representam a quantidade de cor ciano, a quantidade de cor magenta, a quantidade de cor amarelo e a quantidade de cor preta. Dessa forma, têm-se a imersão dos pontos que formam o cubo tridimensional maciço de lado igual a 255 no cubo tetradimensional unitário maciço.

Uma imersão é dita *isométrica* se não existe diferença entre os valores de distância dos dois espaços métricos após a imersão. Um espaço métrico (V, d) é *ℓ_p-imersivo* se existe uma função de imersão isométrica $f : V \to \mathbb{R}^m$, sendo $m > 0$, do espaço métrico (V, d) em outro espaço métrico definido pela norma ℓ_p, ou seja, $d(u, v) = \|f(u) - f(v)\|_p$, sendo u e $v \in V$. É provado a seguir que qualquer espaço métrico finito é ℓ_∞-imersivo.

3.1.1 Imersão no ℓ_∞

O teorema apresentado a seguir mostra um exemplo de imersão isométrica e foi provado também por Williamson e Shmoys [37]. Ele mostra a imersão de espaços métricos finitos gerais em espaços métricos definidos pela norma ℓ_∞.

Teorema 3.1. *Qualquer espaço métrico finito (V, d) é ℓ_∞-imersivo.*

Demonstração. Seja $V = \{v_1, \ldots, v_n\}$. Define-se a função de imersão $f : V \to \mathbb{R}^n$ de forma que:

$$f(v_i) = (d(v_1, v_i), \ldots, d(v_n, v_i)).$$

O valor do comprimento de um vértice $v \in V$ localizado no espaço métrico universal ℓ_∞^n é igual ao maior valor modular de coordenada de v, ou seja, $\|v\|_\infty = \max_{k=1}^{n} \left|v^k\right|$, sendo v^k o valor da k-ésima coordenada de v. Dessa forma, segue a prova algébrica do teorema.

$$\begin{aligned} \|f(v_i) - f(v_j)\|_\infty &= \max_{k=1}^{n} |d(v_k, v_i) - d(v_k, v_j)| && (3.1) \\ &\leq d(v_i, v_j) && (3.2) \end{aligned}$$

Na equação 3.1, foi aplicada a fórmula da distância entre os vértices v_i e v_j dispostos espaço métrico universal ℓ_∞^n após a imersão. Obtém-se essa equação pelo fato do valor da

k-ésima coordenada do vetor $f(v_i) - f(v_j)$ ser igual a $d(v_k, v_i) - d(v_k, v_j)$. A inequação 3.2 é obtida com a aplicação da desigualdade triangular, sendo ℓ_∞ uma métrica. Obtém-se essa inequação pelo fato de que, pela desigualdade triangular, para qualquer k, $d(v_i, v_k) - d(v_k, v_j) \leq d(v_i, v_j)$ e $d(v_k, v_j) - d(v_i, v_k) \leq d(v_i, v_j)$ implica em $|d(v_i, v_k) - d(v_k, v_j)| \leq d(v_i, v_j)$. Por outro lado, como a j-ésima coordenada do vetor $f(v_i) - f(v_j)$ é igual a $d(v_j, v_i) - d(v_j, v_j) = d(v_i, v_j)$ e o valor do comprimento do vetor $f(v_i) - f(v_j)$ é igual ao maior valor modular de coordenada, então $\|f(v_i) - f(v_j)\|_\infty \geq d(v_i, v_j)$. Combinando essa última inequação com a inequação 3.2, obtém-se a equação $\|f(v_i) - f(v_j)\|_\infty = d(v_i, v_j)$. Prova-se então o teorema, sendo a função de imersão f isométrica pelo fato das distâncias terem sido preservadas para todos os pares de vértices. □

3.1.2 Distorção

Apesar de já ter sido provado que existe uma imersão isométrica de qualquer espaço métrico finito em ℓ_∞^n, são raros os casos onde são possíveis imersões isométricas em espaços métricos definidos por normas diferentes da ℓ_∞. Normalmente permite-se mapeamentos cujas distâncias são alteradas. As distâncias podem ser tanto aumentadas como diminuídas conforme uma razão após uma imersão. Dados dois espaços métricos (V, d) e (V', d') e o mapeamento $f : V \to V'$, a *contração* de f é o fator máximo que as distâncias são diminuídas. Em contrapartida, a *expansão* de f é o fator máximo que as distâncias são aumentadas. Os cálculos do fator de contração β e do fator de expansão γ são mostrados nas equações 3.3 e 3.4, respectivamente.

$$\beta = \max_{v_1, v_2 \in V} \frac{d(v_1, v_2)}{d'(f(v_1), f(v_2))} \tag{3.3}$$

$$\gamma = \max_{v_1, v_2 \in V} \frac{d'(f(v_1), f(v_2))}{d(v_1, v_2)} \tag{3.4}$$

Apesar da função desses dois fatores ser medir o fator máximo que as distâncias aumentam e diminuem após uma imersão, o real fator que determina o quanto um espaço métrico diferencia de outro é a *distorção*. Pelo fato de uma imersão poder tanto aumentar

como diminuir as distâncias, faz-se necessário determinar a faixa de erro da imersão como um todo. O fator de distorção α realiza essa medição e é determinado pelo produto dos fatores de contração e expansão, ou seja, $\alpha = \beta \cdot \gamma$.

Existem algumas propriedades importantes sobre o fator de distorção. Toda imersão isométrica possui fator de distorção igual a 1, pois não existe modificação nas distâncias. Além disso, o fator de distorção é invariante sobre a operação de escala das distâncias, ou seja, as distâncias dos dois espaços métricos são modificadas conforme uma mesma taxa após a operação de escala. A distorção de um mapeamento f pode ser vista também como o menor valor $\alpha \geq 1$ de forma que exista um valor $r > 0$, para todo v_1 e $v_2 \in V$ e que obedeça a seguinte desigualdade:

$$r \cdot d(v_1, v_2) \leq d'(f(v_1), f(v_2)) \leq \alpha \cdot r \cdot d(v_1, v_2).$$

O estudo de imersões traz importantes contribuições no projeto de algoritmos. Entre essas contribuições, as mais importantes são [32]:

- **Instâncias como espaços métricos**: a entrada do problema se comporta como um espaço métrico ou pode ser transformada em um. A ideia central é imergir um espaço métrico dado em outro que possui propriedades que podem ser utilizadas para solucionar o problema de forma mais simples. Nesse caso, é possível obter um algoritmo de aproximação de forma que a sua garantia de aproximação sofra um aumento relativo ao fator de distorção da imersão.

- **Relaxamentos de programas inteiros**: o problema é formulado como um programa matemático inteiro que pode ser relaxado e a solução ótima se comporta como a definição de um espaço métrico. Ao arredondar a solução ótima, podem ser utilizadas técnicas que fazem uso de imersões com distorção e resultam, consequentemente, em soluções aproximadas.

- **Problemas sobre espaços métricos**: existem problemas cujo objetivo é encontrar um espaço métrico mais próximo de outro. Existem exemplos de tais problemas na área de biologia molecular.

3.2 Métricas e árvores

Espaços métricos podem também ser definidos por árvores. Árvores podem ser visualizadas de forma hierárquica e possuem propriedades que muitas vezes podem ser utilizadas na criação de algoritmos de aproximação. Williamson e Shmoys [37] conceituam *métricas em árvores* como sendo espaços métricos que definem as distâncias como sendo o tamanho do (único) menor caminho em uma árvore. De forma geral, uma métrica em árvore pode ser modelada por um grafo conexo que não possui ciclos. A Figura 3.1(a) mostra graficamente uma métrica em árvore (V, d) e a Figura 3.1(b) mostra o mesmo espaço métrico visualizado hierarquicamente, sendo o vértice e considerado o nó raiz da árvore. É possível observar também que toda função de distância d modelada por uma árvore sobre o conjunto de vértices V, é também uma métrica, pois para todos os vértices u, v e $w \in V$, $d_{uv} \leq d_{uw} + d_{wv}$ é verdadeiro (desigualdade triangular) e todas as distâncias d_{uv} são obrigatoriamente iguais a d_{vu}.

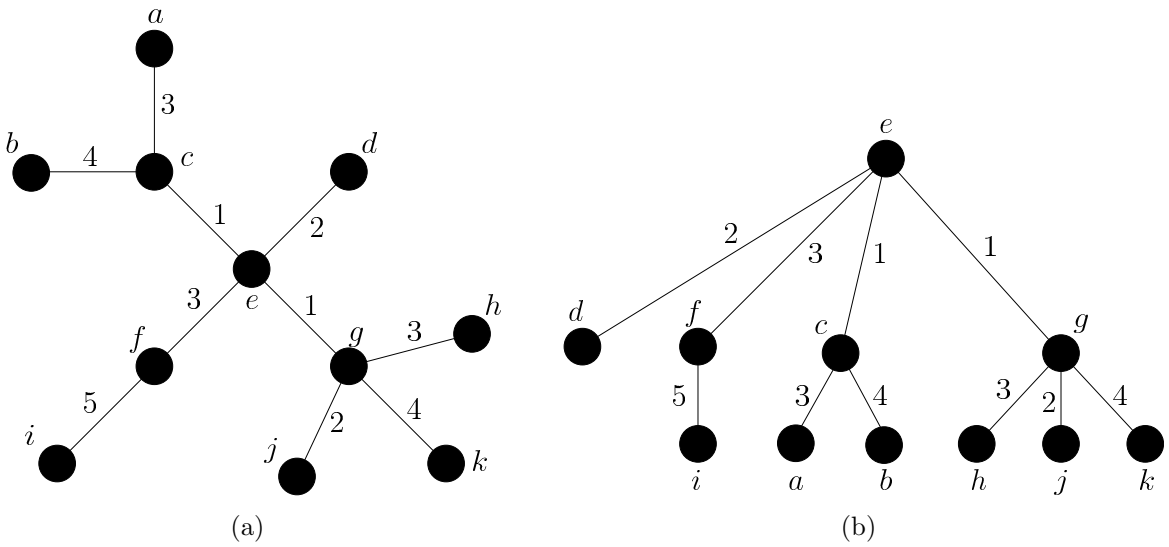


Figura 3.1: Métrica em árvore

Métricas em árvores são interessantes de serem estudadas em algoritmos. Existem problemas NP-difíceis em grafos que se tornam polinomiais se restritos às árvores. Um bom exemplo é apresentado na solução projetada por Awerbuch e Azar [5] para o problema NP-difícil internacionalmente conhecido como *buy-at-bulk network*. Essa solução não será estudada no presente trabalho, mas é um bom exemplo de aplicações de métricas em

árvores com o objetivo de solucionar problemas NP-difíceis. As principais vantagens das métricas em árvores envolvem a apresentação de um conceito de divisão e conquista para solucionar problemas sobre espaços métricos e que a sua aplicação em alguns algoritmos *on-line* de aproximação traz um considerável aumento na garantia de aproximação. Será estudado na seção 4.3 um algoritmo que realiza imersão de espaços métricos finitos gerais em métricas em árvores com o objetivo de solucionar o problema do corte mais disperso.

3.2.1 Imersões determinísticas

É importante estudar imersões em métricas em árvores, vista a sua aplicabilidade no projeto de algoritmos. Em contrapartida, não existem imersões isométricas de espaços métricos finitos gerais em métricas em árvores. Rabinovich e Raz provaram em um de seus trabalhos [31] essa inexistência, independente se for ou não adicionados vértices intermediários na imersão, denominados pelos autores como *vértices de Steiner*. Imersões de espaços métricos finitos definidos por ciclos em métricas em árvores, por exemplo, têm um fator de distorção de no mínimo $\Omega(n)$, sendo n o número de vértices. Segue o teorema.

Teorema 3.2 (Rabinovich e Raz [31])**.** *Um espaço métrico definido por um ciclo de comprimento n não pode ser imerso em outro espaço métrico definido por uma árvore com distorção menor do que $\Omega(n)$.*

Não será apresentada a prova formal do Teorema 3.2, mas existe uma forma simples de visualizá-lo. É dado um ciclo de comprimento n com pesos unitários nas arestas. Ao imergir esse grafo em uma árvore, é necessário que o ciclo seja quebrado, visto que árvores não contém ciclos. Remover uma aresta arbitrária do grafo é uma forma simples de imergir o grafo em árvore quebrando o ciclo. Assim sendo, o grafo imergido passa a ser um grafo linear com n vértices. Após a imersão, as distâncias entre alguns vértices se mantiveram intactas enquanto outras sofreram aumento.

Analisando o pior caso de expansão de distância, se a aresta $\{u, v\}$ foi removida, a distância que sofreu maior expansão foi d_{uv}. Faz-se necessário que todo o grafo seja percorrido para realizar o caminho entre u e v, que nesse caso, percorre-se uma distância

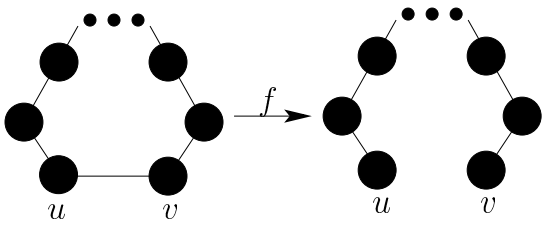


Figura 3.2: Imersão de um ciclo em árvore

total igual a $n-1$. Comparando com a distância antes da imersão, que era igual a 1, essa imersão possui fator de expansão igual a $\Omega(n)$. A Figura 3.2 ilustra um ciclo qualquer com a aresta $\{u, v\}$ removida após a imersão f. Em casos onde manipula-se espaços métricos finitos gerais, fazer uso de métodos probabilísticos de imersão aproxima o fator de distorção da média de distorção das distâncias.

3.2.2 Imersões probabilísticas

Vista uma breve análise de imersão de espaços métricos finitos gerais em métricas em árvores, agora será estudada a média dos casos. Para isso, faz-se necessária uma análise probabilística sobre a distribuição de todas as árvores possíveis de serem geradas após a imersão de um espaço métrico finito geral em uma métrica em árvore. Pelo fato dessa análise ser probabilística, o valor de distorção α não é um valor determinístico, mas sim um valor esperado.

Como não está sendo analisando o pior caso, existe a possibilidade de deixar esse fator α mais próximo do valor médio de distorção. Fakcharoenphol, Rao e Talwar apresentaram um algoritmo de imersão de espaços métricos finitos gerais em métricas em árvores [14] com base nos trabalhos realizados por Bourgain [8]. Bourgain provou que qualquer espaço métrico (V, d) imerge probabilisticamente em ℓ_1 com valor esperado de distorção igual a $O(\log n)$. Fakcharoenphol, Rao e Talwar mostraram um procedimento aleatorizado de imersão com distorção esperada $O(\log n)$. Esse procedimento é capaz de criar em tempo polinomial uma métrica em árvore que garantidamente possui distorção esperada $O(\log n)$.

3.2.2.1 Imersão de espaços métricos finitos em árvores

Antes de apresentar o algoritmo de imersão de espaços métricos finitos gerais em métricas em árvores, faz-se necessária a compreensão da estrutura de uma métrica em árvore. Uma métrica em árvore (V', T), é um espaço métrico finito definido por um conjunto finito de vértices V e uma árvore T definida sobre o conjunto de vértices $V' \supseteq V$ com pesos não negativos em cada aresta de T. Como $V' \supseteq V$, a árvore T pode possuir nós a mais além daqueles compostos pelos vértices já contidos em V. Esses nós excedentes intermeiam os caminhos entre cada par de vértices u e $v \in V$ na árvore T. Como será explicado adiante, a distância T_{uv} é igual a soma dos pesos das arestas da árvore T contidas no caminho entre os vértices u e $v \in V'$.

A imersão em métricas em árvores é feita por um algoritmo que subdivide o espaço métrico que engloba todos os vértices contidos em V distanciados pela métrica d em outros subespaços menores. Cada subespaço encontrado é novamente dividido em outros subespaços. Essas divisões são realizadas até o momento em que cada subespaço contenha somente um único vértice de V. Ao final, tem-se uma árvore de subespaços que define em cada nível quais são os vértices pertencentes a V que são mais próximos entre si. Os autores referenciam esse processo de corte do conjunto de vértices baseado nas distâncias entre eles como *decomposição hierárquica de cortes* do espaço métrico finito definido pela métrica d. A Figura 3.3 ilustra graficamente a explicação dada, onde um espaço métrico que define um conjunto S é subdividido em outros espaços métricos que definem os subconjuntos S_1, S_2 e S_3.

Após essa decomposição, tem-se uma nova métrica em árvore (V', T) que segue algumas propriedades. A decomposição gera uma árvore com somente um nó raiz e com exatamente $\log_2 \Delta + 1$ níveis. Será explicado posteriormente o motivo dela possuir essa quantidade de níveis e como Δ é definido. Cada nó da árvore gerada representa um subconjunto de V. Cada nível da árvore possui um conjunto de nós que corresponde a um particionamento do conjunto de vértices V. Todos os vértices pertencentes a V estão obrigatoriamente presentes em cada nível de forma que nenhum vértice esteja em mais de um nó da árvore no mesmo nível.

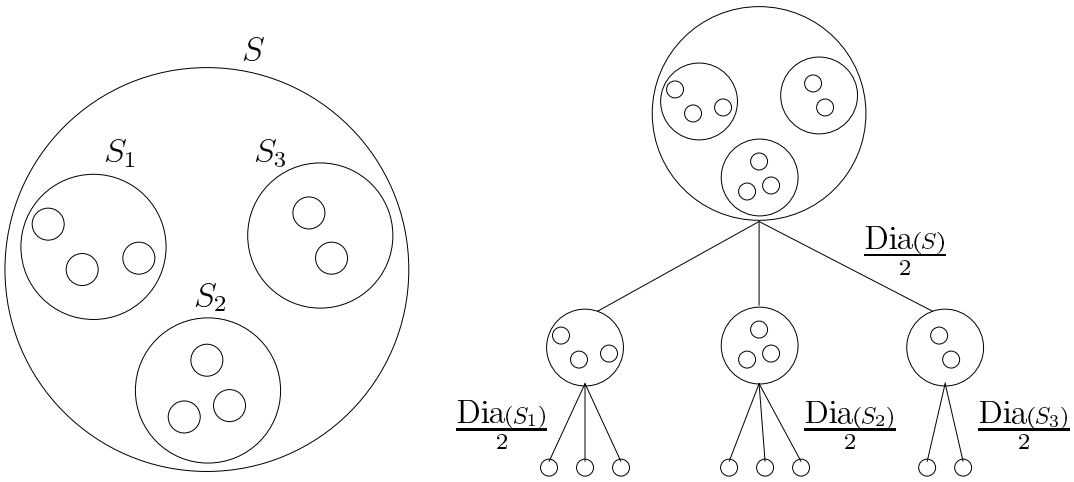


Figura 3.3: Decomposição hierárquica de cortes
[14]

Como é possível visualizar na Figura 3.4, o nó raiz da árvore está presente no nível $\log_2 \Delta$ e, por se tratar de um nó único naquele nível, ele representa todo o conjunto de vértices V. Já as folhas da árvore estão todas obrigatoriamente no nível 0 e cada uma representa exatamente um único vértice de V. Isso faz com que a partição do conjunto V referente ao nível zero seja a n-partição de conjuntos unitários de V. Cada folha da árvore corresponde a um vértice de V, enquanto que cada nó corresponde a um vértice de V', por isso $V' \supseteq V$. Os nós existentes nos níveis intermediários da árvore, correspondem a algum subconjunto S do conjunto de vértices V. Os nós filhos daquele que corresponde ao subconjunto S compõem um particionamento do subconjunto S e assim sucessivamente.

O subespaço referente a um nó, que representa o subconjunto S, é limitado por uma *bola* $B(v, r)$ na métrica d centralizada em algum vértice de S. Uma bola $B(v, r)$ é definida como o conjunto de vértices contidos em V que estão internos a um círculo de raio r na métrica d com centro posicionado sobre o vértice $v \in V$. Assim, segue a definição formal de bola [26].

$$B(v, r) = \{u \in V : d_{uv} \leq r\}$$

Essa bola possui o seu raio r variável em função do nível da árvore que o subespaço faz parte. Os autores assumiram que as bolas dos nós que são folhas possuem raios iguais e com valor escolhido aleatoriamente entre $\frac{1}{2}$ e 1, mais especificamente $r \in \left[\frac{1}{2}, 1\right)$. O tamanho do raio duplica a cada subida de nível da árvore, aumentando também o

subespaço até que ele contenha todos os vértices do conjunto V. Assim sendo, o raio de uma bola que está em um nível i da árvore é igual a um valor no intervalo $[2^{i-1}, 2^i)$.

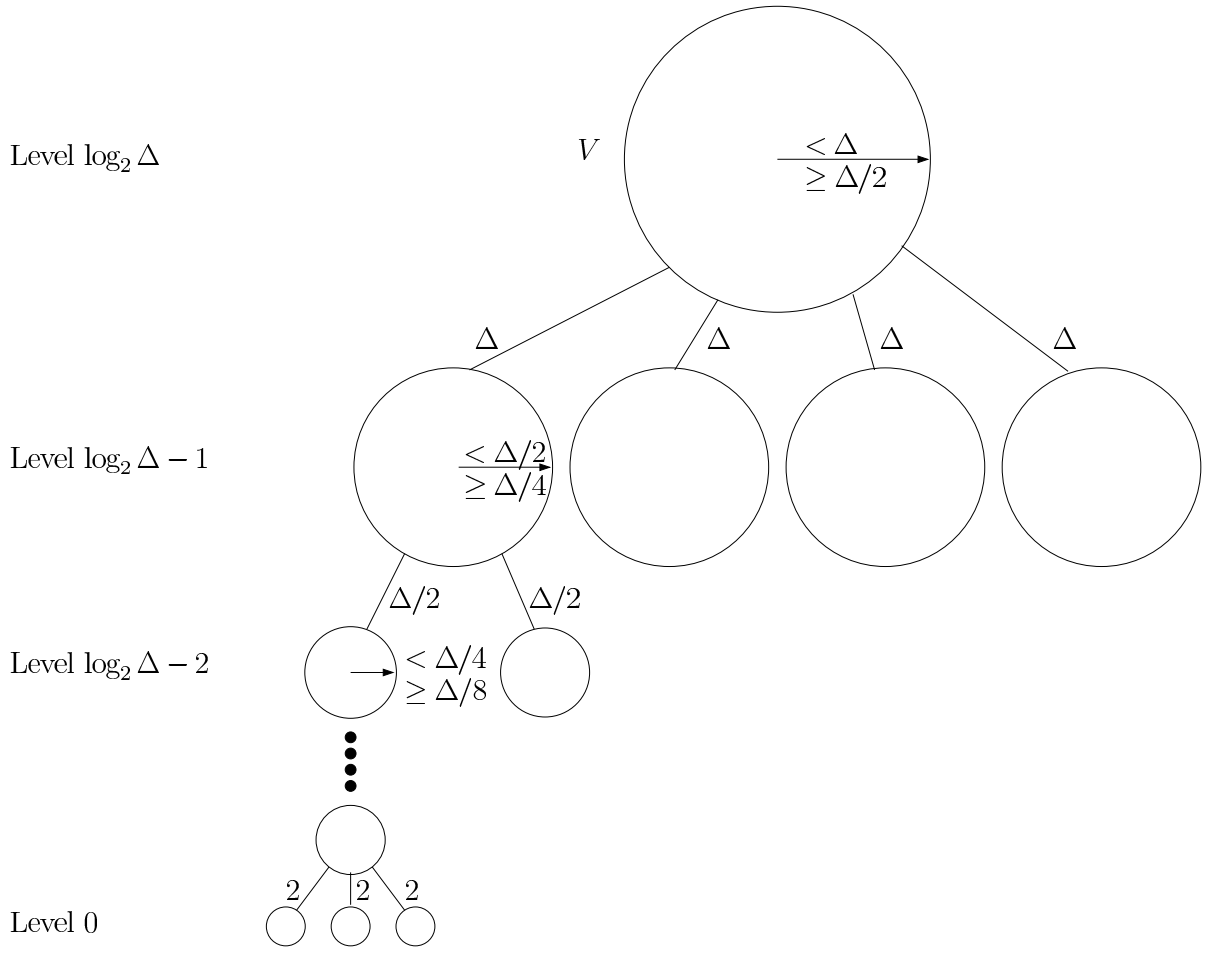


Figura 3.4: Decomposição hierárquica de cortes de um espaço métrico [37]

Como cada vértice se encontra sozinho nas folhas, então o raio do espaço métrico centralizado em cada vértice no nível 0 será um valor entre $\frac{1}{2}$ e 1. Para que cada vértice esteja sozinho nos seus referidos espaços no nível 0 da árvore, é necessário que todas as distâncias d_{uv} na métrica d sejam maiores ou iguais a 1. Isso define um limite inferior de distância para o correto funcionamento do algoritmo: $d_{uv} \geq 1$, para todo $u \neq v$. Isso pode ser assumido para qualquer espaço métrico finito, visto que é possível escalar a métrica de forma que todas as distâncias possuam valores maiores ou iguais a 1. Caso a menor distância possua um valor entre 0 e 1, é possível escalar a métrica realizando a divisão de todas as distâncias pelo valor da menor distância. Isso fará com que a métrica seja escalada de tal forma que a menor distância seja igual a 1.

Como os subespaços são centralizados em qualquer vértice de V e como o nó raiz contém todos os nós pertencentes a V, então o menor valor possível de raio para o subespaço referente ao nó raiz precisa ser no mínimo a distância entre o par de vértices mais distantes na métrica d. Considerando esse fato, com a definição de que os limites dos raios dos subespaços são potências de 2, pode-se calcular o limitante superior do raio do subespaço referente ao nó raiz. Sabendo que o raio de um subespaço é limitado inferiormente por 2^{i-1}, para que a bola no nível i possa englobar todos os vértices, é necessário que $2^{i-1} > \max_{u,v} d_{uv}$, ou seja, $2^i > 2\max_{u,v} d_{uv}$. Assim, define-se $\Delta = \min\{2^i | 2^i > 2\max_{u,v} d_{uv}, i \in \mathbb{N}\}$. Em outras palavras, Δ é igual à menor potência de 2 maior do que $2\max_{u,v} d_{uv}$.

As arestas entre dois níveis i e $i-1$ têm tamanhos iguais ao limitante superior dos raios dos subespaços referentes aos nós no nível i. A cada descida no nível da árvore, os tamanhos das arestas decrescem pela metade sendo essas também determinadas por uma potência de 2. A árvore possui os nós folhas no nível 0 e o nó raiz no nível $\log_2 \Delta$. Os limites dos raios dos subespaços sempre são divididos por 2 a cada descida no nível da árvore. A Figura 3.4 mostra graficamente a explicação dada sobre o conceito de decomposição hierárquica de cortes.

Será provado o Lema 3.1 fazendo uso das definições apresentadas e das propriedades existentes em uma métrica em árvore criada pela decomposição hierárquica de cortes. Esse lema será importante posteriormente para calcular a distorção do algoritmo que gera a métrica em árvore (V', T).

Lema 3.1. *Qualquer árvore T obtida por meio da decomposição hierárquica de cortes de um espaço métrico finito (V, d) tem as distâncias $T_{uv} \geq d_{uv}$ para todos os pares de vértices u e $v \in V$. Além disso, se o primeiro ancestral comum de u e v está no nível i, então $T_{uv} \leq 2^{i+2}$.*

Demonstração. Seja i igual ao nível do primeiro nó ancestral comum entre os vértices u e v em T. Como a distância T_{uv} é igual ao tamanho do caminho que interliga os vértices u e v, então T_{uv} é igual a soma dos tamanhos das arestas contidas nos caminhos do nível 0 até o nível i multiplicado por 2. Multiplica-se por 2 porque o caminho entre dois vértices

folhas u e v na árvore T envolve a subida do nível 0 até o nível i e a descida do nível i até o nível 0. Como o percurso de subida do nível 0 até um nível i tem tamanho igual ao percurso de descida do nível i até o nível 0 e como o tamanho das arestas abaixo do nível i é igual a 2^i, então $T_{uv} = 2 \cdot \sum_{j=1}^{i} 2^j$. Assim,

$$\begin{aligned} T_{uv} &= 2 \cdot \sum_{j=1}^{i} 2^j \\ &= 2(2^1 + \ldots + 2^i) \\ &= 2(2^{i+1} - 2) \\ &= 2^{i+2} - 4 \\ &\leq 2^{i+2}. \end{aligned}$$

Para os vértices u e v pertencerem a um conjunto S, é necessário que ambos pertençam à bola que limita o subespaço referente a S. O caso em que u e v pertencem a S acontece quando a distância entre eles é no máximo igual ao diâmetro da bola. Então a distância entre u e v é igual a no máximo duas vezes o limitante superior do raio de um subespaço localizado no nível i da árvore, que é igual a 2^i. Dessa forma, o limitante superior da distância entre u e v na métrica d em um subespaço no nível i é $d_{uv} < 2^{i+1}$. Como a distância entre u e v na árvore é calculada por $T_{uv} = 2^{i+2} - 4$, o limitante superior da distância entre os mesmos vértices na métrica d é $d_{uv} < 2^{i+1}$ e os níveis da árvore cujos vértices u e v podem pertencer a S é $i \geq 1$, então o lema segue.

$$\begin{aligned} T_{uv} &= 2^{i+2} - 4 \\ &\geq 2^{i+1} \\ &> d_{uv} \end{aligned}$$

□

3.2.2.2 Análise do algoritmo de imersão em árvore

Dados os conceitos e propriedades que envolvem métricas em árvores, agora será apresentado o algoritmo aleatorizado que gera uma métrica em árvore. Logo após, ele será analisado e será mostrado todo o cálculo da distorção fazendo uso dos conceitos apresentados. Ele gera uma métrica em árvore (V', T) que aproxima um espaço métrico finito (V, d) de tal forma que $d_{uv} \leq T_{uv}$ e $\mathsf{E}[T_{uv}] \leq O(\log n) \cdot d_{uv}$, para todo par de vértices u e $v \in V$. Dessa forma, o valor esperado da distorção das distâncias em T entre cada par de vértices é no máximo $O(\log n)$ vezes a distância entre o mesmo par de vértices na métrica d original.

Como descrito no Algoritmo 1, primeiramente são inicializadas todas as variáveis aleatórias. Essas variáveis são: uma permutação aleatória π dos vértices contidos em V e o raio r_0 de todos os subespaços localizados no nível 0 da árvore a ser criada. Como dito anteriormente, os raios dos subespaços localizados no nível 0 são iguais e pertencem ao intervalo $\left[\frac{1}{2}, 1\right)$, sendo r_0 igualado a algum valor aleatório nesse intervalo. Os raios dos subespaços posicionados nos níveis superiores da árvore são calculados com base em r_0. Sendo r_0 um valor entre $\frac{1}{2}$ e 1, então os valores de raio de todos os subespaços localizados em um nível i são iguais e se encontram no intervalo $[2^{i-1}, 2^i)$. Cada conjunto $\mathcal{C}(i)$ criado no algoritmo contém todos os nós localizados no nível i da árvore criada.

Para se obter a métrica em árvore, basta executar um procedimento que particiona um subconjunto S sucessivas vezes, para cada nó encontrado a cada particionamento. Para realizar o particionamento de um subconjunto S localizado em um nível i da árvore, o algoritmo atribui todos os vértices $u, \ldots, v \in S$ ao primeiro subespaço processado localizado no nível $i - 1$, de forma que $u, \ldots, v$ estão internos a ele. A sequência de processamento de cada subespaço realizada pelo algoritmo é dada pela permutação π e cada subespaço é centralizado no vértice $\pi(j)$, sendo j uma variável que referencia um elemento da permutação π.

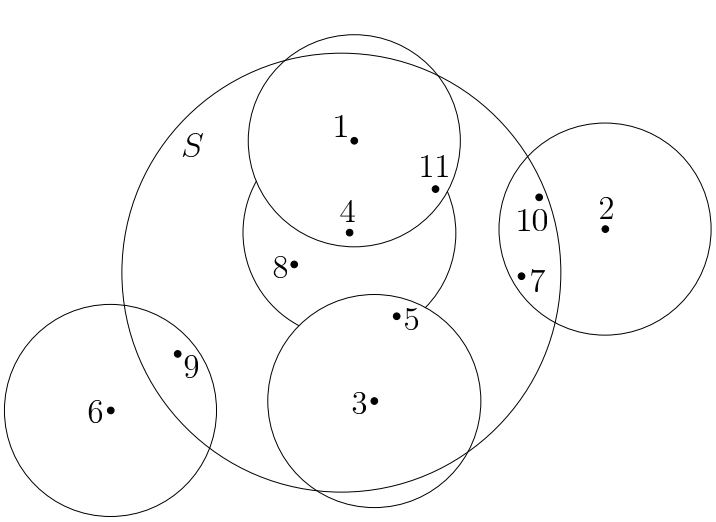
Pelo fato de ser considerado cada termo da permutação π, a cada particionamento são processados até $|V|$ subespaços diferentes. Isso faz com que outros vértices, além daqueles que pertencem ao subconjunto S, também sejam processados e, consequentemente, sejam

```
Escolha uma permutação aleatória π de V;
Faça Δ ser a menor potência de 2 maior do que 2 max_{u,v} d_uv;
Escolha r_0 ∈ [1/2, 1) aleatoriamente e uniformemente;
para i ← 1 até log_2 Δ faça  r_i = 2^i · r_0 ;
C(log_2 Δ) = {V};
Crie o nó correspondente a V;
para i ← log_2 Δ até 1 faça
    C(i − 1) = ∅;
    para todo C ∈ C(i) faça
        S ← C;
        para j ← 1 até n faça
            se B(π(j), r_{i−1}) ∩ S ≠ ∅ então
                Adicione B(π(j), r_{i−1}) ∩ S em C(i − 1);
                Remove B(π(j), r_{i−1}) ∩ S de S;
            fim
        fim
        Crie nós correspondentes a todos os conjuntos de C(i − 1);
        Ligue os nós criados ao nó correspondente a C com arestas de tamanho 2^i;
    fim
fim
```

Algoritmo 1: Algoritmo de decomposição hierárquica de cortes

criados subespaços centralizados em vértices que não pertencem a S. Porém, todos os vértices que não pertencem a S não são inclusos em seu particionamento. A árvore está criada após a execução *top-down* do processo de particionamento em cada subconjunto S encontrado a partir do conjunto V, esse referente ao nó raiz.

A Figura 3.5 ilustra o processo de partição de um subconjunto S. No início da execução do Algoritmo 1, para o exemplo mostrado na figura, a permutação inicial π é a sequência crescente do vértice 1 até o vértice 11. Os vértices que pertencem a S são: 1, 3, 4, 5, 7, 8, 9, 10 e 11. Como o primeiro vértice da permutação π é o vértice 1, então primeiramente foi criado o subconjunto S_1, seguido do subconjunto S_2 e assim por diante. Para criar o subconjunto S_1, são determinados quais vértices que pertencem a bola que limita o subespaço centralizado no vértice 1. Os vértices que estão nesse subespaço, que fazem parte de S e que ainda não foram selecionados por outro subconjunto anteriormente são adicionados no subconjunto S_1. Logo após é dada continuidade ao algoritmo com a determinação de S_2 e dos outros subconjuntos de forma consecutiva. Nesse mesmo exemplo, o conjunto $\mathcal{C}(i)$ contém os subconjuntos S_1, S_2, S_3, S_4 e S_6.



$S = \{1, 3, 4, 5, 7, 8, 9, 10, 11\}$
$\pi = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11\}$

$S_1 = \{1, 4, 11\}$
$S_2 = \{7, 10\}$
$S_3 = \{3, 5\}$
$S_4 = \{8\}$
$S_5 = \emptyset$
$S_6 = \{9\}$
$S_7 = \emptyset$
$S_8 = \emptyset$
$S_9 = \emptyset$
$S_{10} = \emptyset$
$S_{11} = \emptyset$

Figura 3.5: Execução do algoritmo de decomposição hierárquica de cortes [37]

Para que seja comprovada a distorção de uma árvore gerada pelo algoritmo que realiza a decomposição hierárquica de cortes, primeiramente faz-se necessária a comprovação dos Teoremas 3.3 e 3.4. Ambos determinam valores de probabilidade com raciocínios sobre os eventos que ocorrem ao ser executado o algoritmo. Em algum momento, o algoritmo separa um par de vértices u e $v \in V$ em diferentes subespaços. Isso obrigatoriamente acontece porque todos os nós folhas da árvore fazem referência a um único vértice de V. A seguir são apresentados os dois eventos que ocorrem toda vez que um par de vértices é separado em diferentes subespaços.

São dados um par de vértices u e $v \in V$ e o nível $i+1$ do primeiro nó ancestral comum entre u e v na árvore T. Como o nível desse nó é $i + 1$, então os vértices u e v estão separados no nível i da árvore. Isso obrigatoriamente acontece, senão $i+1$ não é o nível do primeiro nó ancestral comum entre u e v. Durante a execução do algoritmo, a separação de u e v acontece quando outros dois eventos ocorrem simultaneamente: quando um vértice w *escolhe* $\{u, v\}$ no nível i e quando o vértice w *corta* $\{u, v\}$ no nível i. Cada um dos dois eventos são explicados a seguir.

O evento que ocorre quando um vértice $w \in V$ for o primeiro vértice na permutação aleatória π tal que pelo menos um dos dois vértices u ou v pertença a bola $B(w, r_i)$ no nível i da árvore, será denominado como w *escolhe* $\{u, v\}$ no nível i. Esse evento será

representado algebricamente pela variável aleatória S_{iw}. O valor de S_{iw} é igual a 1 se e somente se w for o primeiro vértice da permutação aleatória tal que u ou v pertença a bola $B(w, r_i)$ no nível i e; S_{iw} é igual a 0 caso contrário. Note que esse evento não elimina a hipótese de ambos u e v pertencerem a bola $B(w, r_i)$. É necessário que outro evento também aconteça simultaneamente para que u e v sejam obrigatoriamente separados no nível i.

O evento que ocorre quando um vértice w separa os vértices u e v em um nível i da árvore será denominado como w *corta* $\{u, v\}$ no nível i. Esse evento será representado algebricamente pela variável aleatória X_{iw}. O valor de X_{iw} é igual a 1 se e somente se exatamente um dos vértices u ou v pertença a bola $B(w, r_i)$ no nível i e; X_{iw} é igual a 0 caso contrário. Com a definição desse outro evento, é possível calcular algebricamente a distorção realizada pelo Algoritmo 1.

Como para separar um par de vértices u e v deve, em algum momento, acontecer simultaneamente os dois eventos acima em um nível i, ou seja, o evento $X_{iw} \wedge S_{iw}$, então faz-se necessário determinar as probabilidades $\Pr[X_{iw}]$ e $\Pr[S_{iw}|X_{iw}]$. Para isso, seguem os cálculos das duas probabilidades nos Teoremas 3.3 e 3.4, respectivamente.

Teorema 3.3. *A probabilidade de* w cortar $\{u, v\}$ *no nível* i *da árvore* T*, sendo* u, v *e* $w \in V$*, é:*

$$\Pr[X_{iw}] = \frac{|[2^{i-1}, 2^i) \cap [d_{uw}, d_{vw})|}{2^{i-1}}.$$

Demonstração. Sem perder generalidade, seja u o vértice mais próximo de w no par de vértices u e v, ou seja, $d_{uw} \leq d_{vw}$. Assim sendo, a probabilidade de w *cortar* u e v no nível i é igual a probabilidade de $u \in B(w, r_i)$ e $v \notin B(w, r_i)$. Quando isso acontece, o valor do raio r_i está obrigatoriamente entre d_{uw} e d_{vw}, ou seja, $d_{uw} \leq r_i \leq d_{vw}$, graficamente ilustrado na Figura 3.6(a). Como r_i é calculado para estar distribuído uniformemente no intervalo $[2^{i-1}, 2^i)$ e como w só corta u e v se $d_{uw} \leq r_i < d_{vw}$, então segue o cálculo da probabilidade $\Pr[X_{iw}]$.

$$\begin{aligned} \Pr[X_{iw}] &= \frac{|[2^{i-1}, 2^i) \cap [d_{uw}, d_{vw})|}{|[2^{i-1}, 2^i)|} \\ &= \frac{|[2^{i-1}, 2^i) \cap [d_{uw}, d_{vw})|}{2^{i-1}} \end{aligned} \tag{3.5}$$

Os limites do raio r_i são determinados pelo intervalo $[2^{i-1}, 2^i)$, então a divisão pelo seu comprimento resulta na proporção da probabilidade calculada na equação 3.5. O comprimento de $[d_{vw},d_{uw})$ resulta na variação de comprimento que o r_i pode ter para que w *corte* u e v no nível i. Porém, como o intervalo $[d_{uw}, d_{vw})$ pode ter seus limites fora de $[2^{i-1}, 2^i)$, então essa variação é dada como a intersecção entre os dois intervalos, como mostra a Figura 3.6(b). Assim sendo, a probabilidade de w *cortar* $\{u, v\}$ no nível i é a razão do comprimento do intervalo $[2^{i-1}, 2^i) \cap [d_{uw}, d_{vw})$ pelo comprimento do intervalo $[2^{i-1}, 2^i)$, mostrada na equação 3.5.

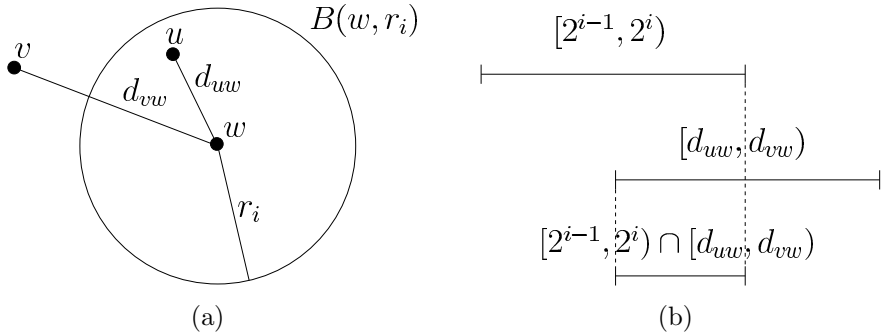


Figura 3.6: Probabilidade de w *cortar* u e v

□

Teorema 3.4. *O valor da probabilidade de* w escolher $\{u, v\}$ *no nível* i *da árvore* T*, dado que* w corta $\{u, v\}$*, é* $\Pr[S_{iw}|X_{iw}] = \frac{1}{j}$*, sendo* w *o* j*-ésimo vértice mais próximo de* u *ou* v *em* d*. Além disso, o valor da probabilidade* $\Pr[S_{iw}|X_{iw}]$ *independe no nível* i *da árvore, por isso* $\Pr[S_{iw}|X_{iw}] = b_w$*.*

Demonstração. Como é suposto que X_{iw} já aconteceu, então a probabilidade $\Pr[S_{iw}|X_{iw}]$ é igual a probabilidade de w ser o primeiro vértice a *escolher* u ou v. Para determinar essa probabilidade, ordene os vértices $w \in V$ com base na menor distância d entre w

e os vértices u e v, ou seja, $\min(d_{wu}, d_{wv})$. Após isso, tem-se uma lista ordenada dos vértices em V que começa com os mais próximos aos vértices u ou v e termina com os mais distantes. Sendo w o j-ésimo vértice mais próximo de u ou v, então a probabilidade de w ser o primeiro vértice a *escolher* u ou v é igual a probabilidade de w estar antes em π dos $j-1$ vértices mais próximos a u ou v. Assim sendo, a probabilidade $\Pr[S_{iw}|X_{iw}]$ é igual a $\frac{1}{j}$, sendo $1 \leq j \leq n$. Dessa forma, prova-se parte do teorema com a determinação de $\Pr[S_{iw}|X_{iw}] = \frac{1}{j}$.

Concluindo, como $\Pr[S_{iw}|X_{iw}]$ é igual a $\frac{1}{j}$, percebe-se que o cálculo desse valor de probabilidade depende unicamente da posição que w está na lista ordenada de vértices. Assim, o valor de $\Pr[S_{iw}|X_{iw}]$ independe do nível i da árvore, podendo ser referenciado também como b_w. □

Finalmente, o próximo teorema prova que uma métrica em árvore criada por meio de decomposição hierárquica de cortes tem o valor esperado de distorção igual a $O(\log n)$.

Teorema 3.5. *Dado um espaço métrico finito (V, d), sendo $d_{uv} \geq 1$ para todo u e $v \in V$ e $u \neq v$, existe um algoritmo aleatorizado polinomial que gera uma métrica em árvore (V', T), sendo $V \subseteq V'$, de forma que $d_{uv} \leq T_{uv}$ e $\mathsf{E}[T_{uv}] \leq O(\log n) \cdot d_{uv}$.*

Demonstração. Como o Algoritmo 1 executa o particionamento de subconjuntos em $\log_2 \Delta$ níveis diferentes, sendo que em cada nível ele cria no máximo $|V|$ subconjuntos, então conclui-se que ele é polinomial no tamanho da entrada. Isso porque o valor de Δ pode ser no máximo 2^t, sendo t o tamanho em bits da entrada. Como existem $\log_2 \Delta$ níveis diferentes, então a quantidade máxima de níveis na árvore é igual $O(t)$.

Como mostrado no Algoritmo 1, cada subconjunto S criado é o resultado de uma partição desempenhada no nível anterior. É realizado o particionamento em cada subconjunto criado, para todos os $\log_2 \Delta$ níveis da árvore. O critério de parada do algoritmo é alcançado quando se atinge o nível 0, provando, assim, a corretude do algoritmo na criação de uma árvore T que possui as propriedades iguais às definições mostradas na seção 3.2.2.1.

Com a aplicação do Lema 3.1, prova-se parte do teorema em questão, sendo todas

distâncias entre dois vértices na árvore T maior ou igual à distância entre os mesmos vértices na métrica d, ou seja, $d_{uv} \leq T_{uv}$, sendo $d_{uv} \geq 1$. Basta calcular a distorção desse processo de imersão para provar o restante do teorema.

Durante a execução do algoritmo, um par de vértices u e v são separados em algum nível i. Dessa forma, iniciam-se os cálculos da prova com a aplicação do Lema 3.1. Como u e v estão separados no nível i, então o primeiro nó ancestral comum entre eles se encontra no nível $i+1$, obtendo-se a inequação 3.6. Como visto anteriormente, para que u e v sejam separados no nível i, os eventos X_{iw} e S_{iw} precisam ocorrerem simultaneamente. O termo $\max\limits_{i=0,\ldots,\log\Delta-1}(\exists w \in V : X_{iw} \wedge S_{iw})$, na equação 3.7, representa o momento durante a execução do algoritmo que um vértice w *escolherá* e *cortará* os vértices u e v em um nível i da árvore que varia de 0 a $\log\Delta - 1$. Como $X_{iw} \wedge S_{iw}$ só pode ser igual a 1 ou a 0 e obrigatoriamente todos os vértices são separados no nível 0, então o resultado da função max será em algum momento igual a 1. A função max pode ser substituída por somatórios pelo fato do evento $X_{iw} \wedge S_{iw}$ acontecer uma única vez durante toda a execução do algoritmo. Dessa forma, a equação 3.8 possui dois somatórios: um que percorre a árvore nível a nível e outro que percorre os vértices em cada nível.

$$
\begin{aligned}
T_{uv} &\leq 2^{i+3} && (3.6)\\
&= \max_{i=0,\ldots,\log\Delta-1}(\exists w \in V : X_{iw} \wedge S_{iw}) \cdot 2^{i+3} && (3.7)\\
&= \sum_{w\in V}\sum_{i=0}^{\log\Delta-1} (X_{iw} \wedge S_{iw}) \cdot 2^{i+3} && (3.8)
\end{aligned}
$$

Como se busca o valor esperado de distorção das distâncias, então prossegue-se com o cálculo principal da prova na inequação 3.9. Com o uso do teorema da linearidade da esperança e como $(X_{iw} \wedge S_{iw})$ corresponde a uma variável aleatória que possui somente valores iguais a 0 ou 1, então é possível obter a equação 3.10. Parte da equação 3.11 é o resultado da aplicação direta da fórmula da probabilidade: $\Pr[A \wedge B] = \Pr[A|B] \cdot \Pr[B]$. A equação 3.12 pode ser obtida com a aplicação do Teorema 3.4. O componente b_w foi retirado do segundo somatório pelo fato do seu valor ser independente do nível i.

$$\begin{aligned}
\mathsf{E}[T_{uv}] &\leq \mathsf{E}\left[\sum_{w\in V}\sum_{i=0}^{\log\Delta-1}(X_{iw}\wedge S_{iw})\cdot 2^{i+3}\right] && (3.9)\\
&= \sum_{w\in V}\sum_{i=0}^{\log\Delta-1}\Pr[X_{iw}\wedge S_{iw}]\cdot 2^{i+3} && (3.10)\\
&= \sum_{w\in V}\sum_{i=0}^{\log\Delta-1}\Pr[S_{iw}|X_{iw}]\cdot\Pr[X_{iw}]\cdot 2^{i+3} && (3.11)\\
&= \sum_{w\in V} b_w \sum_{i=0}^{\log\Delta-1}\Pr[X_{iw}]\cdot 2^{i+3} && (3.12)
\end{aligned}$$

Faz-se necessário calcular o limitante superior de $\sum_{i=0}^{\log\Delta-1}\Pr[X_{iw}]\cdot 2^{i+3}$ para dar continuidade à prova. O cálculo desse limitante superior é iniciado com a equação 3.13, sendo ela obtida com a multiplicação do valor 2^{i+3} nos dois lados da igualdade dada no Teorema 3.3.

$$\begin{aligned}
2^{i+3}\cdot\Pr[X_{iw}] &= \frac{2^{i+3}}{2^{i-1}}\left|[2^{i-1},2^i)\cap[d_{uw},d_{vw})\right| && (3.13)\\
&= 16\left|[2^{i-1},2^i)\cap[d_{uw},d_{vw})\right| && (3.14)
\end{aligned}$$

A equação 3.15 é obtida adicionando os somatórios em cada lado da igualdade mostrada na equação 3.14. A união dos intervalos $[2^{i-1},2^i)$ nos níveis i da árvore, sendo $0\leq i\leq\log\Delta-1$, define o intervalo $\left[\frac{1}{2},\frac{\Delta}{2}\right)$. Essa união, na maioria dos casos, engloba todo o intervalo $[d_{uw},d_{vw})$. Mas podem existir casos, como o mostrado na Figura 3.6(b), onde o intervalo $[d_{uw},d_{vw})$ é cortado pelo fato de d_{vw} poder ser um valor maior do que $\frac{\Delta}{2}$. Além disso, sabe-se que toda distância em d é obrigatoriamente maior do que $\frac{1}{2}$, pelo fato de $d_{uv}\geq 1$. Isso faz com que o valor de $\left|\left[\frac{1}{2},\frac{\Delta}{2}\right)\cap[d_{uw},d_{vw})\right|$ seja obrigatoriamente menor ou igual ao valor de $|[d_{uw},d_{vw})|$, ou seja, $\left|\left[\frac{1}{2},\frac{\Delta}{2}\right)\cap[d_{uw},d_{vw})\right|\leq|[d_{uw},d_{vw})|$. Com isso, conclui-se que o valor do somatório existente na equação 3.16 possui como limitante superior o comprimento do intervalo $[d_{uw},d_{vw})$, obtendo-se, assim, a inequação 3.17. O limitante superior de $\sum_{i=0}^{\log\Delta-1}\Pr[X_{iw}]\cdot 2^{i+3}$ é obtido também com o uso da propriedade da

desigualdade triangular.

$$\begin{aligned}
\sum_{i=0}^{\log\Delta-1} \Pr[X_{iw}] \cdot 2^{i+3} &= \sum_{i=0}^{\log\Delta-1} 16 \left|[2^{i-1}, 2^i) \cap [d_{uw}, d_{vw})\right| && (3.15)\\
&= 16 \sum_{i=0}^{\log\Delta-1} \left|[2^{i-1}, 2^i) \cap [d_{uw}, d_{vw})\right| && (3.16)\\
&\leq 16 \left|[d_{uw}, d_{vw})\right| && (3.17)\\
&= 16(d_{vw} - d_{uw})\\
&\leq 16 d_{uv}
\end{aligned}$$

Encontrado o limitante superior de $\sum_{i=0}^{\log\Delta-1} \Pr[X_{iw}] \cdot 2^{i+3}$, é possível dar continuidade no cálculo do valor esperado de distorção. Percebe-se que d_{uv} independe do vértice w, podendo esse ser removido do somatório existente na inequação 3.18. De acordo com o Teorema 3.4, o valor de b_w é igual a $\frac{1}{j}$, sendo w o j-ésimo vértice mais próximo de u ou v. Como u e v são vértices pré-determinados e cada vértice obrigatoriamente possui o seu lugar na lista ordenada de distâncias referenciada no Teorema 3.4, então os valores de j seguem uma sequência que inicia em 1 e termina em n não necessariamente ordenada. Assim, $\sum_{w \in V} b_w$ pode ser substituído por $\sum_{j=1}^{n} \frac{1}{j}$, como mostrado na equação 3.19. Sabe-se que $\sum_{j=1}^{n} \frac{1}{j} = O(\log n)$, dessa forma, encontra-se o limitante superior esperado da distorção do algoritmo de decomposição hierárquica de cortes e, consequentemente, prova-se o teorema em questão.

$$\begin{aligned}
\mathsf{E}[T_{uv}] &\leq \sum_{w \in V} b_w \sum_{i=0}^{\log\Delta-1} \Pr[X_{iw}] \cdot 2^{i+3}\\
&\leq \sum_{w \in V} b_w (16 d_{uv}) && (3.18)\\
&= 16 d_{uv} \sum_{w \in V} b_w\\
&= 16 d_{uv} \sum_{j=1}^{n} \frac{1}{j} && (3.19)\\
&= O(\log n) d_{uv}
\end{aligned}$$

□

O Lema 3.1, o Algoritmo 1 e o Teorema 3.5 foram apresentados por Williamson e Shmoys [37], tomando como base o algoritmo e provas obtidas por Fakcharoenphol, Rao e Talwar [14]. Bartal provou em um de seus trabalhos [7] que existem espaços métricos que qualquer aproximação probabilística por métricas em árvores tem a distorção esperada de $\Omega(\log n)$. Dessa forma, o resultado obtido por esse algoritmo é o melhor possível.

3.3 Imersão de métricas em árvores em ℓ_1

Não se pode utilizar o resultado obtido na seção 3.2.2.1 para imergir um espaço métrico definido pela norma ℓ_1 em uma métrica em árvore. Porém, o contrário é possível. Neste caso, existe uma imersão isométrica de métricas em árvore em espaços métricos definidos pela norma ℓ_1. Essa imersão é importante no projeto de algoritmos de aproximação cuja entrada se comporta como espaços métricos. O teorema a seguir mostra uma imersão isométrica de qualquer métrica em árvore no espaço métrico ℓ_1^{n-1}.

Teorema 3.6. *Toda métrica em árvore (V, T) é ℓ_1-imersível.*

Demonstração. Seja r o vértice raiz da árvore T e $\xi_{ur}(e)$ uma função que é igual a 1 se a aresta e da árvore T pertence ao caminho entre u e $r \in V$ na árvore T, ou igual a 0 caso contrário. Como T é uma árvore, então a quantidade de arestas em T é igual a quantidade de elementos em V menos 1, ou seja, $n-1$. Sendo T_e o termo que representa o valor de peso da aresta e, a função $f : V \to \mathbb{R}^{n-1}$ é um mapeamento dos vértices $u \in V$ em ℓ_1^{n-1} de forma que:

$$f(u) = (T_{e_1} \cdot \xi_{ur}(e_1), \cdots, T_{e_{n-1}} \cdot \xi_{ur}(e_{n-1})).$$

Dada a imersão f, inicia-se a prova com o cálculo da distância na norma ℓ_1 entre dois vértices u e v após a imersão. O valor $f(u)_i$ na equação 3.20 corresponde ao valor da i-ésima coordenada do ponto $f(u)$ localizado no espaço métrico ℓ_1^{n-1}. Como visto na imersão f, o valor da i-ésima coordenada de um ponto $f(u)$ é igual a $T_{e_i} \cdot \xi_{ur}(e_i)$. Dessa

forma, a equação 3.21 pode ser obtida, sendo essa uma generalização para todas as arestas $e \in E$ da árvore T. Pelo fato da distância ser um valor positivo, o termo T_e pode ser posto em evidência e retirado do módulo, gerando assim a equação 3.22. Percebe-se que alguns dos termos $\xi_{ur}(e) - \xi_{vr}(e)$ na mesma equação são zerados pelo fato de uma mesma aresta e pertencer a ambos os caminhos de u a r e de v a r na árvore T. As arestas que fazem parte desse conjunto são aquelas que determinam o caminho entre a e r na mesma árvore, sendo a o primeiro vértice ancestral comum entre u e v. Dessa forma, o valor de $|\xi_{ur}(e) - \xi_{vr}(e)|$ é igual ao valor $|\xi_{ua}(e) - \xi_{va}(e)|$ para toda a aresta e, gerando, assim, a equação 3.23. Sendo a o primeiro vértice ancestral comum entre u e v, então não existe intersecção entre os caminhos de u a a e de v a a. Não havendo intersecção, em nenhum momento os valores de $\xi_{ua}(e)$ e $\xi_{va}(e)$ são simultaneamente iguais a 1. Como a função ξ somente adota os valores 1 ou 0, então a função modular existente na equação 3.23 pode ser substituída pela soma $(\xi_{ua}(e) + \xi_{va}(e))$, gerando assim a equação 3.24.

$$\begin{aligned}
\|f(u) - f(v)\|_1 &= \sum_{i=1}^{n-1} |f(u)_i - f(v)_i| && (3.20)\\
&= \sum_{e \in E} |T_e \cdot \xi_{ur}(e) - T_e \cdot \xi_{vr}(e)| && (3.21)\\
&= \sum_{e \in E} T_e \, |\xi_{ur}(e) - \xi_{vr}(e)| && (3.22)\\
&= \sum_{e \in E} T_e \, |\xi_{ua}(e) - \xi_{va}(e)| && (3.23)\\
&= \sum_{e \in E} T_e (\xi_{ua}(e) + \xi_{va}(e)) && (3.24)
\end{aligned}$$

Encontrada a equação que calcula a distância entre os pontos $f(u)$ e $f(v)$ na norma ℓ_1, dar-se-á continuidade à prova com o cálculo da distância entre u e v na árvore T. Assim como mostrado na prova do Lema 3.1, a distância entre dois vértices u e v quaisquer na árvore T é dada pela soma das distâncias entre u e a e entre v e a, sendo a o primeiro vértice ancestral comum de u e v. A equação 3.25 representa esse cálculo, sendo que o termo P_{ua} corresponde ao conjunto formado pelas arestas que descrevem o caminho entre u e a na árvore T. Na equação 3.26 foi feita uma generalização em ambos os somatórios

e, por essa causa, adicionou-se uma multiplicação pela função ξ para eliminar da soma os elementos que não pertencem a P_{ua} e P_{va}. A equação 3.27 é possível de ser obtida apenas com manipulação algébrica da equação 3.26. A equação 3.28 pode ser obtida por substituição direta da equação 3.24, provando assim o teorema em questão.

$$
\begin{aligned}
T_{uv} &= \sum_{e \in P_{ua}} T_e + \sum_{e \in P_{va}} T_e && (3.25)\\
&= \sum_{e \in E} (T_e \cdot \xi_{ua}(e)) + \sum_{e \in E} (T_e \cdot \xi_{va}(e)) && (3.26)\\
&= \sum_{e \in E} (T_e \cdot \xi_{ua}(e) + T_e \cdot \xi_{va}(e)) \\
&= \sum_{e \in E} T_e (\xi_{ua}(e) + \xi_{va}(e)) && (3.27)\\
&= \|f(u) - f(v)\|_1 && (3.28)
\end{aligned}
$$

□

É possível visualizar que o procedimento realizado para imergir uma métrica em árvore (V, T) em um espaço métrico definido pela norma ℓ_1 executa em tempo polinomial e pode ser utilizado em algoritmos de aproximação. Essa imersão é realizada no algoritmo apresentado na seção 4.3.

CAPÍTULO 4

ALGORITMOS DE APROXIMAÇÃO

Muitos problemas existentes na literatura podem ser modelados com o uso de programação matemática. Existem vários tipos de programação matemática, entre elas destacam-se a programação linear e a programação vetorial. Essas ferramentas se mostram eficientes na área de análise de algoritmos de aproximação por serem compactas e por proporcionarem uma abordagem matemática para formular provas de teoremas.

Existem problemas NP-difíceis que também podem ser modelados por um tipo específico de programa linear: o inteiro. Apesar dessa ferramenta ser capaz de encontrar a solução ótima, ela ainda não possui algoritmo que a soluciona em tempo polinomial. De fato, o próprio problema de solucionar programação linear inteira é um problema NP-difícil.

Analisando de forma superficial, não existe grande diferença entre resolver um problema NP-difícil modelado em um programa linear inteiro ou resolvê-lo com algoritmos exponenciais especializados. Porém muitos autores apresentam técnicas que tomam decisões baseadas na resposta de um programa matemático comum, o qual possui algoritmo polinomial que o soluciona. É possível modelar problemas NP-difíceis em programas matemáticos que resultam em valores de resposta contínuos e que podem ser utilizados para solucionar o problema como um todo. Algoritmos de aproximação também podem ser obtidos dessa forma.

Uma solução criada por meio de programas matemáticos possivelmente possui uma garantia de aproximação de uma solução ótima. Pelo fato de se utilizar programas matemáticos, existe a possibilidade de se determinar matematicamente esse fator com mais facilidade e precisão. Hochbaum [20] apresentou uma série de algoritmos que realizam arredondamento de programas lineares, entre eles a técnica de arredondamento de dual. Exitem também técnicas que fazem uso de conceitos de programação linear, porém são

algoritmos especializados que não realizam arredondamento de soluções de programas lineares, como o método primal-dual apresentado por Bar-Yehuda e Even [6].

São apresentados nesse trabalho alguns algoritmos que fazem uso de técnicas baseadas em respostas de programas matemáticos. Primeiramente será mostrado um método que encontra uma solução factível para o problema do corte máximo baseado na resposta de um programa vetorial. Logo após são apresentados dois algoritmos que solucionam o problema do corte multisseparador mínimo. Por último será apresentado um algoritmo que soluciona o problema do corte mais disperso por meio da resolução de um programa linear seguido de uma imersão métrica.

4.1 Algoritmo para o corte máximo

Será apresentada uma forma de utilizar programação vetorial no desenvolvimento de um algoritmo de aproximação aleatorizado capaz de resolver o problema do corte máximo. Relembrando, no problema do corte máximo tem-se como entrada um grafo não direcionado com pesos não negativos em cada aresta. O objetivo é separar o conjunto de vértices em dois subconjuntos U e W maximizando a soma dos pesos das arestas que separam as partes. Uma melhor explanação sobre o problema pode ser encontrada na seção 2.1.2. O algoritmo foi apresentado por Goemans e Williamson [17] e possui o valor esperado de garantia de aproximação de 0,878, ou seja, o algoritmo é 0,878-aproximado.

Inicialmente será apresentado o programa não linear inteiro que modela o problema do corte máximo e sua respectiva explicação. Em seguida será mostrado um outro programa vetorial que modela o mesmo problema, mas admite valores contínuos como resposta. Logo após, será apresentado o algoritmo de arredondamento aleatorizado com as suas respectivas provas de corretude e aproximação. O programa não linear inteiro a seguir modela o problema do corte máximo, onde as variáveis y_i são iguais a -1 se o vértice $i \in U$ ou são iguais a $+1$ se o vértice $i \in W$. Uma solução ótima para esse programa não linear é também uma solução ótima para o problema do corte máximo. Isso significa que o valor da solução ótima dele é igual ao valor da solução ótima do problema, denotado no presente trabalho como OPT.

$$\begin{aligned} \text{maximizar} \quad & \tfrac{1}{2} \sum_{\{i,j\} \in E} w_{ij}(1 - y_i y_j) \\ \text{sujeito a} \quad & y_i \in \{-1, +1\}, \qquad\qquad i = 1, \ldots, n. \end{aligned} \tag{4.1}$$

Lema 4.1. *O programa 4.1 modela o problema do corte máximo.*

Demonstração. Como $U = \{i : y_i = -1\}$ e $W = \{i : y_i = +1\}$, então quando uma aresta $\{i, j\}$ pertence ao conjunto de corte, o valor de $y_i y_j$ é igual a -1. Quando uma aresta não pertence ao conjunto de corte, então $y_i y_j$ é igual a $+1$. Na função objetivo, quando $\{i, j\}$ pertence ao conjunto de corte, o peso da aresta é multiplicado por 2 e acumulado com a mesma multiplicação realizada também com as outras arestas que fazem parte do conjunto de corte. Como todas as arestas que estão no conjunto de corte têm o peso multiplicado por 2, então o resultado do somatório é duas vezes a soma dos pesos das arestas que estão no conjunto de corte. A multiplicação por $1/2$ ao final resulta no valor exato do corte. □

O programa não linear inteiro 4.1 pode ser reescrito como um programa não linear que admite valores contínuos como resposta. Esse programa não linear é obtido substituindo o termo $y_i y_j$ pelo termo $v_i \cdot v_j$ na função objetivo e substituindo o termo $y_i \in \{-1, +1\}$ pelo termo $v_i \cdot v_i = 1$ no único conjunto de restrições. Para finalizar, é adicionado um novo conjunto de restrições que permitem que as variáveis assumam valores pertencentes ao conjunto dos números reais, resultando, assim, no programa vetorial 4.2.

$$\begin{aligned} \text{maximizar} \quad & \tfrac{1}{2} \sum_{\{i,j\} \in E} w_{ij}(1 - v_i \cdot v_j) \\ \text{sujeito a} \quad & v_i \cdot v_i = 1, \qquad\qquad i = 1, \ldots, n, \\ & v_i \in \mathbb{R}^n, \qquad\qquad i = 1, \ldots, n. \end{aligned} \tag{4.2}$$

O programa vetorial 4.2 é considerado uma "folga" do programa não linear inteiro 4.1 e, claramente, admite valores fracionários de coordenadas para os vetores variáveis v_i.

Pelo fato do programa 4.2 ser modelado por variáveis que assumem valores fracionários, uma solução ótima para esse programa possivelmente não é uma solução factível para o problema original. O programa 4.2 é chamado de *programa vetorial relaxado.*

Existem algumas propriedades importantes sobre esses dois programas não lineares. Toda solução factível para o programa não linear inteiro 4.1 corresponde a uma solução factível para o programa vetorial relaxado 4.2, mas o contrário não é verdade. O valor de qualquer solução factível para o programa não linear inteiro 4.1 é igual ao valor da mesma solução para o programa vetorial relaxado 4.2. Qualquer solução factível para o programa não linear inteiro 4.1 satisfará as restrições do programa vetorial relaxado 4.2. As funções objetivo de ambos os programas matemáticos são equivalentes.

Como o programa vetorial relaxado aceita valores fracionários para as coordenadas dos vetores v_i, acrescido do fato que o problema do corte máximo é um problema de maximização, então o valor da solução ótima do programa vetorial relaxado é maior ou igual ao valor da solução ótima do programa não linear inteiro 4.1. Assim sendo, pode ser considerado que o valor da solução ótima do programa vetorial relaxado determina um limitante superior para o valor da solução ótima do programa não linear inteiro. A prova da corretude dessa relaxação é apresentada a seguir.

Teorema 4.1. *Dada uma solução factível y para o programa não linear 4.1, existe uma solução factível v para o programa vetorial 4.2 com o mesmo valor.*

Demonstração. Seja v_i um vetor de dimensão n tal que $v_i = (y_i, 0, \ldots, 0)$, para todo $i = 1, \ldots, n$. Como $y_i \in \{-1, +1\}$, então $v_i \cdot v_i = 1$ é verdadeiro para todo i, estando em conformidade com o conjunto de restrições do programa vetorial 4.2. Da mesma forma, para qualquer solução do programa inteiro 4.1, $v_i \cdot v_j = y_i y_j$ sempre será verdadeiro, de forma que o resultado das funções objetivos dos dois programas sempre resultem nos mesmos valores. □

O Teorema 4.1 prova que para qualquer solução factível do programa não linear inteiro 4.1 existe uma solução factível equivalente para o programa vetorial 4.2. O contrário, porém, não é necessariamente verdade. O programa vetorial 4.2 assume um número

maior de soluções factíveis em comparação ao programa não linear inteiro 4.1. Como para toda solução factível para 4.1 existe uma solução factível para 4.2 de mesmo valor e como 4.2 assume um conjunto maior de soluções, então 4.2 é uma relaxação de 4.1.

É possível visualizar que o programa vetorial 4.2 possui um conjunto maior de soluções factíveis pelo fato das componentes dos vetores v_i poderem ser valores fracionários contidos no intervalo $[-1, 1]$. As restrições $v_i \cdot v_i = 1$ apenas garantem que o módulo de cada vetor v_i seja igual a 1 e, consequentemente, garantem que as suas componentes estejam entre os limites do intervalo $[-1, 1]$. Assim, além das soluções onde os vetores v_i são iguais a $(-1, 0, \ldots, 0)$ ou $(1, 0, \ldots, 0)$, o programa vetorial 4.2 também admite soluções factíveis cujas primeiras componentes de cada v_i são diferentes de -1 e 1.

Como mostrado, o programa 4.2 admite um conjunto de soluções que contém todas soluções de 4.1. Assim, sabe-se que $Z^*_{PV} \geq OPT$ é verdadeiro por se tratar de um problema de maximização, sendo Z^*_{PV} o valor da solução ótima de 4.2 e OPT o valor da solução ótima de 4.1. Pelo fato de existirem algoritmos polinomiais que solucionam programas semidefinidos, então é possível obter um algoritmo de aproximação que soluciona o problema do corte máximo realizando um processamento sobre a solução do programa 4.2. O Algoritmo 2 mostra os passos desse processamento.

Seja v^* uma solução ótima do programa vetorial relaxado 4.2;
para $i \leftarrow 1$ **até** n **faça**
    Faça r_i um valor aleatório amostrado de uma distribuição normal com média 0 e variância 1;
fim
$r = r / \|r\|$;
$U \leftarrow \emptyset$;
$W \leftarrow \emptyset$;
para $i \leftarrow 1$ **até** n **faça**
    se $v_i^* \cdot r \geq 0$ **então**
        $U \leftarrow U \cup \{i\}$;
    fim
    senão
        $W \leftarrow W \cup \{i\}$;
    fim
fim
retorna U e W;

Algoritmo 2: Algoritmo de aproximação para o problema do corte máximo

Inicialmente o programa vetorial é solucionado, obtendo-se os vetores v_i^*. Como foi visto, um programa vetorial pode ser solucionado em tempo polinomial por se tratar de um programa semidefinido relaxado. Logo após, é gerado um vetor $r = (r_1, \ldots, r_n)$ de forma que cada componente seja adquirida aleatoriamente de uma distribuição normal com média 0 e variância 1. Após o vetor r ter sido gerado, é realizada a normalização dele fazendo com que o seu módulo seja igual a 1. Finalmente a solução é obtida da seguinte maneira: se $v_i^* \cdot r \geq 0$ então $i \in U$ ou $i \in W$ caso contrário.

Pode-se interpretar geometricamente o algoritmo aleatorizado da seguinte maneira: ele faz uso de lógica algébrica de vetores em um espaço n-dimensional. Imagine todos os n vetores unitários (são unitários porque $v_i \cdot v_i = 1$) encontrados na solução do programa 4.2 com suas origens posicionadas no centro de uma esfera. Essa esfera é partida ao meio por um hiperplano posicionado exatamente sobre o seu centro e com o vetor normal igual ao vetor r. Os vetores que estão internos em uma partição da esfera são inclusos em U e os vetores resultantes, que estão internos na outra partição, são inclusos em W. A Figura 4.1 ilustra a esfera partida ao meio por um hiperplano aleatório.

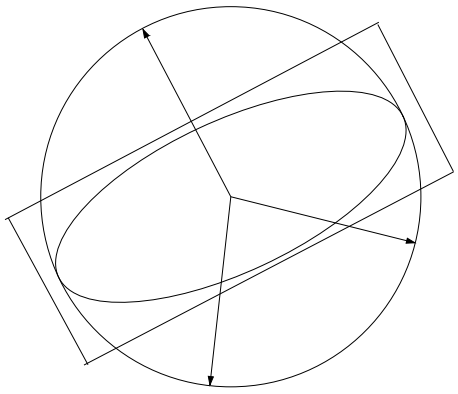


Figura 4.1: Esfera partida ao meio por um hiperplano aleatório
[37]

Como a função objetivo do programa vetorial busca maximizar o seu valor, então a solução do programa se resume em um conjunto de vetores cujo produto escalar entre eles seja minimizado, ou seja, com o produto escalar o mais próximo possível do valor -1. Assim sendo, a resposta tenderá a ser formada por vetores que em parte possuem uma direção e parte possuem a direção oposta. Os pesos das arestas w_{ij} definirão quais

vetores possuirão uma direção e quais possuirão a outra direção. Quanto maior o peso de uma aresta, maior será a tendência dos vetores que representam os seus vértices estarem dispostos em sentidos opostos. Quando a esfera é partida por um hiperplano aleatório, como os vetores tendem a estar separados em dois conjuntos que apontam para sentidos opostos, então a probabilidade desse hiperplano separar esses dois conjuntos é grande, resultando em um corte próximo do ótimo.

Essas afirmações precisam ser matematicamente comprovadas. Mensurar o desempenho de algoritmos aleatorizados não é uma tarefa simples e necessita de cálculos que envolvem conceitos de probabilidade. Para isso, inicialmente é necessário provar que o hiperplano é gerado de forma aleatória e uniforme. Goemans e Williamson [17] citam dois fatos para comprovar que o vetor r é linearmente distribuído em uma esfera unitária n-dimensional. O Fato 4.1, apresentado inicialmente por Rényi [33], afirma que se dois vetores e_1 e e_2 são ortogonais então as componentes de um vetor r projetado sobre e_1 e e_2 são distribuídas pela distribuição normal com média 0 e variância 1. Como as componentes de r foram geradas por essa mesma distribuição e, fazendo uso do Fato 4.2, apresentado inicialmente por Knuth [25], então conclui-se que o vetor r, após a sua normalização, está uniformemente distribuído em uma esfera n-dimensional.

Fato 4.1. *As projeções do vetor r sobre dois vetores unitários e_1 e e_2 são independentes e distribuídas pela distribuição normal com média 0 e variância 1 se e somente se e_1 e e_2 são ortogonais.*

Fato 4.2. *A normalização do vetor r, $r = r/\|r\|$, é uniformemente distribuída sobre uma esfera de raio unitário n-dimensional.*

Provada a geração aleatória e uniforme do hiperplano que parte a esfera n-dimensional ao meio, basta calcular o quão próxima da resposta ótima está a resposta devolvida pelo algoritmo. Os autores apresentam o Corolário 4.1, que também faz uso dos Fatos 4.1 e 4.2, para calcular a probabilidade de uma aresta qualquer pertencer ao conjunto de corte. Esse cálculo é mostrado no Lema 4.2.

Corolário 4.1. *Seja r' a projeção de r sobre um plano bidimensional. Então a norma-*

lização de r', $r' = r'/\|r'\|$, é distribuída de acordo com a distribuição normal em um círculo unitário no plano.

Lema 4.2. *A probabilidade de uma aresta $\{i, j\}$ pertencer ao conjunto de corte é igual a $\frac{1}{\pi}\arccos(v_i \cdot v_j)$.*

Demonstração. Seja v_i o vetor que representa o vértice i e v_j o vetor que representa o vértice j. Seja r' o vetor resultante da projeção do vetor r no plano definido pelos vetores v_i e v_j. Como r' é uma projeção de r, então existe um vetor r'' tal que $r = r' + r''$. Para que seja mantida essa igualdade, o vetor r'' é ortogonal a ambos os vetores v_i e v_j. Dessa forma, $v_i \cdot r = v_i \cdot (r' + r'') = v_i \cdot r'$ e, similarmente, $v_j \cdot r = v_j \cdot r'$.

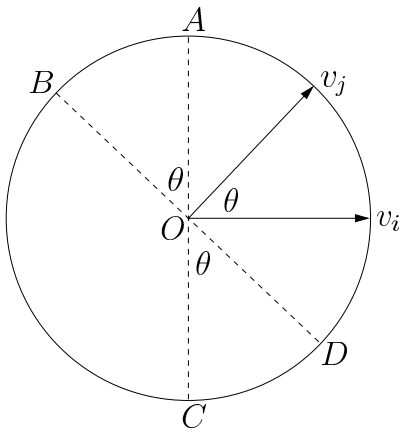


Figura 4.2: Ângulos entre vetores no hiperplano
[37]

Na Figura 4.2, a linha formada pelos pontos AC é perpendicular ao vetor v_i e a linha formada por BD é perpendicular ao vetor v_j. Ao posicionar os vetores v_i e r' na origem O, é possível afirmar com base no Corolário 4.1 que o ângulo α entre os vetores v_i e r' está uniformemente distribuído em $[0, 2\pi)$. Se o vetor r' está a direita da linha AC, então o produto escalar entre r' e v_i é positivo, ou seja, o ângulo entre os vetores r' e v_i é menor do que 90°. O mesmo acontece entre os vetores v_j e r' na linha BD. A aresta $\{i, j\}$ pertence ao conjunto de corte se $i \in W$ e $j \in U$ ou se $i \in U$ e $j \in W$. Ou seja, $\{i, j\}$ pertence ao conjunto de corte se $v_i \cdot r' > 0$ e $v_j \cdot r' < 0$ ou se $v_i \cdot r' < 0$ e $v_j \cdot r' > 0$. Em outras palavras, a aresta $\{i, j\}$ pertence ao conjunto de corte se r' estiver em uma das fatias $\angle AOB$ ou $\angle COD$. Se o ângulo formado entre os vetores v_i e v_j é igual a θ, então os

ângulos das duas fatias também são iguais a θ. Como α está uniformemente distribuído em $[0, 2\pi)$, então a fração desse evento acontecer com base nesse intervalo é $\frac{2\theta}{2\pi}$.

A probabilidade de uma aresta $\{i, j\}$ pertencer ao conjunto de corte é igual a $\frac{\theta}{\pi}$. Utilizando a fórmula do produto escalar, temos $v_i \cdot v_j = \|v_i\| \, \|v_j\| \cos\theta$ [35]. Sabendo que v_i e v_j são unitários, devido a um conjunto de restrições do programa vetorial 4.2, então $v_i \cdot v_j = \cos\theta$. Isolando a variável referente ao ângulo, temos $\theta = \arccos(v_i \cdot v_j)$. Dessa forma, a probabilidade de uma aresta $\{i, j\}$ pertencer ao conjunto de corte é $\frac{1}{\pi}\arccos(v_i \cdot v_j)$. É importante mostrar que o valor dessa probabilidade sempre estará no intervalo $[0, 1]$ pelo fato da função arccos retornar valores de arco em radianos que estão no intervalo $[0, \pi]$. □

Os autores criaram o Lema 4.3 para realizar uma comparação entre o valor da função objetivo calculado por meio da resposta obtida pelo algoritmo e o valor da função objetivo da resposta ótima do problema.

Lema 4.3. *Para* $x \in [-1, 1]$, $\frac{1}{\pi}\arccos(x) \geq 0,878 \cdot \frac{1}{2}(1 - x)$ *é verdadeiro.*

Esboço. É possível limitar inferiormente a função $\frac{1}{\pi}\arccos(x)$ pela função $c \cdot \frac{1}{2}(1 - x)$ no intervalo $[-1, 1]$. Para isso, é necessário encontrar o valor de c que torna tal afirmação verdadeira. Nesse caso, o valor de c é igual ao ponto de mínimo da função de taxa $\frac{\frac{1}{\pi}\arccos(x)}{\frac{1}{2}(1-x)}$ no intervalo [-1,1]. Para encontrar o ponto de mínimo, calcula-se a derivada da função de taxa e iguala-se a função derivada a 0. Solucionando essa equação, encontra-se os valores de x que são pontos críticos. Com os pontos críticos calculados, verifica-se qual deles resulta no ponto de mínimo no intervalo [-1,1]. A constante c é igual ao valor encontrado após solucionar a função de taxa substituindo o valor x que resulta no ponto de mínimo. A Figura 4.3 ilustra os gráficos das funções $\frac{1}{\pi}\arccos(x)$, $\frac{1}{2}(1 - x)$ e $\frac{\frac{1}{\pi}\arccos(x)}{\frac{1}{2}(1-x)}$. É possível visualizar que o valor da constante c é igual a 0,878.

□

Finalmente, o Teorema 4.2 comprova a garantia de aproximação do algoritmo mencionado.

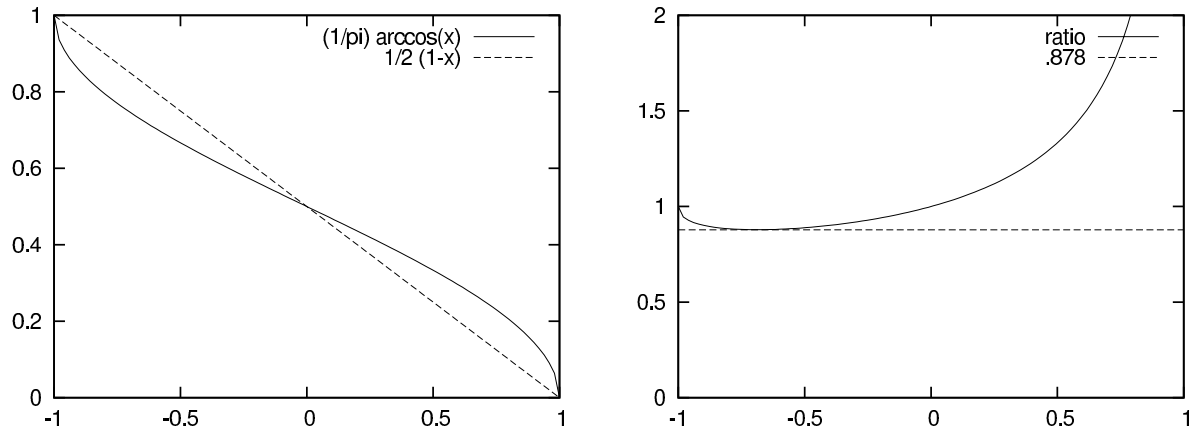


Figura 4.3: Gráficos referentes ao Lema 4.3
[37]

Teorema 4.2. *O Algoritmo 2 é um algoritmo aleatorizado 0,878-aproximado para o problema do corte máximo.*

Demonstração. É possível visualizar que o algoritmo apresentado é polinomial durante todo o processo, desde onde o programa vetorial 4.2 é resolvido até o seu término. Para demonstrar o presente teorema, basta provar qual é o valor da garantia de aproximação.

Seja X_{ij} uma variável aleatória de forma que $X_{ij} = 1$ se a aresta $\{i, j\}$ pertence ao conjunto de corte e 0 caso contrário. Seja W uma variável aleatória que expressa o peso total do corte, ou seja, $W = \sum_{\{i,j\} \in E} w_{ij} X_{ij}$.

Inicia-se a prova com a equação 4.3, que determina o valor esperado de W. Sendo X_{ij} uma variável aleatória que só pode ser igual a 1 ou 0, então o valor esperado de X_{ij} é igual a probabilidade dela ser igual a 1. A equação 4.4 pode ser obtida com a aplicação do Lema 4.2. A inequação 4.5 é obtida com a aplicação direta do Lema 4.3. A equação 4.6 é obtida com manipulação algébrica sobre a inequação 4.5. Percebe-se que a função objetivo do programa vetorial está presente na equação 4.6, gerando, assim, a equação 4.7 por substituição. Como o valor da solução do programa vetorial Z^*_{PV} é maior do que o valor da solução ótima do problema OPT, então conclui-se pela inequação 4.8 que o valor esperado da solução é no mínimo $0,878 \cdot OPT$. Assim, prova-se que o valor esperado da solução devolvida pelo algoritmo é pelo menos 0,878 vezes o valor da solução ótima do problema.

$$
\begin{aligned}
\mathsf{E}[W] &= \sum_{\{i,j\}\in E} w_{ij} \cdot \Pr[X_{ij} = 1] && (4.3)\\
&= \sum_{\{i,j\}\in E} w_{ij} \cdot \frac{1}{\pi} \arccos(v_i \cdot v_j) && (4.4)\\
&\geq \sum_{\{i,j\}\in E} w_{ij} \cdot 0,878 \cdot \frac{1}{2}(1 - v_i \cdot v_j) && (4.5)\\
&= 0,878 \cdot \frac{1}{2} \sum_{\{i,j\}\in E} w_{ij}(1 - v_i \cdot v_j) && (4.6)\\
&= 0,878 \cdot Z^*_{PV} && (4.7)\\
&\geq 0,878 \cdot OPT && (4.8)
\end{aligned}
$$

□

É possível perceber na prova do Teorema 4.2 que algoritmos aleatorizados possuem a garantia de aproximação determinada sobre o valor esperado da solução devolvida pelo algoritmo. Dessa forma, existem soluções cujos valores não condizem com a garantia de aproximação, pelo fato de ter sido calculado um valor esperado. Porém, esse valor é mais próximo da média de todas as respostas possíveis de serem devolvidas pelo algoritmo, diferente de quando se busca limitar pelo pior caso.

Erdős apresentou um dos primeiros algoritmos aleatorizados para o problema do corte máximo cujo valor esperado da solução é pelo menos metade da soma dos pesos das arestas [13]. Sahni e Gonzalez [34] apresentaram um algoritmo com garantia de aproximação igual a $\frac{1}{2}$ e que pode ser visto como uma versão determinística do algoritmo apresentado por Erdős. Apesar da garantia de aproximação de ambos os algoritmos serem consideráveis, o algoritmo apresentado nessa seção possui a garantia de aproximação melhor. Goemans e Williamson desconfiam que não existe algoritmo com garantia de aproximação superior, a não ser que $\mathsf{P} = \mathsf{NP}$. O Teorema 4.3 apresentado por Håstad [19] mostra que existe uma grande possibilidade disso ser verdade. Além disso, Khot, Kindler, Mossel e O'Donnell [24] provaram, com base na conjectura dos jogos únicos, a inexistência de algoritmos de aproximação que solucionam o problema do corte máximo com garantia de aproximação

maiores do que 0,878, detalhado no Teorema 4.4.

Teorema 4.3. *Se existe um algoritmo α-aproximado para o problema do corte máximo com $\alpha > \frac{16}{17} \approx 0,941$, então $\mathsf{P} = \mathsf{NP}$.*

Teorema 4.4. *Dada a conjectura dos jogos únicos, não existe algoritmo α-aproximado para o problema do corte máximo com constante*

$$\alpha > \min_{-1 \leq x \leq 1} \frac{\frac{1}{\pi} \arccos(x)}{\frac{1}{2}(1-x)} \geq 0,878.$$

4.2 Algoritmos para o corte multisseparador mínimo

O problema que será abordado nessa seção é conhecido como o problema do corte multisseparador mínimo. Nesse problema é dado como entrada um grafo não direcionado $G = (V, E)$, custos $c_e \geq 0$ para todas as arestas $e \in E$ e k vértices selecionados $s_1, \ldots, s_k \in V$. O objetivo é particionar o conjunto de vértices V em k subconjuntos disjuntos C_i de forma que cada vértice selecionado s_i pertença a um subconjunto disjunto. Esse problema é formalmente definido na seção 2.1.3.

Primeiramente será mostrado um algoritmo combinatório 2-aproximado que soluciona o problema do corte multisseparador mínimo. Logo após, será apresentado um algoritmo aleatorizado $\frac{3}{2}$-aproximado que soluciona o mesmo problema com a relaxação de um programa linear.

4.2.1 Solução com algoritmo combinatório

Os autores Dahlhaus, Johnson, Papadimitriou, Seymour e Yannakakis [11] apresentaram um algoritmo combinatório de aproximação por meio de uma análise sobre toda solução factível para o problema. Dada uma solução factível com o conjunto de corte F, seja C_i o conjunto de vértices alcançáveis no grafo G após a remoção das arestas pertencentes ao conjunto de corte F partindo de um vértice selecionado s_i. Seja $F_i = \delta(C_i)$ para todo

vértice selecionado s_i, onde $\delta(C_i)$ é o conjunto de todas as arestas que possuem somente um vértice em C_i. É possível notar que cada conjunto F_i é um conjunto de corte que isola o componente conexo que contém o vértice s_i dos outros vértices selecionados. Como isso acontece, cada conjunto de corte F_i também é chamado de *conjunto de corte de isolamento*. Poderão existir arestas que fazem parte de mais de um conjunto de corte de isolamento. Por exemplo, um vértice de uma aresta e pode se encontrar em um componente conexo C_i e o outro vértice de e se encontrar em outro componente conexo C_j sendo $i \neq j$, ou seja, $e \in F_i$ e $e \in F_j$.

$S \leftarrow \emptyset$;
para $i \leftarrow 1$ **até** k **faça** $S \leftarrow S \cup \{s_i\}$;
para $i \leftarrow 1$ **até** k **faça**
    $G' \leftarrow$ INCLUIR-VÉRTICE-T$(G, S - \{s_i\})$;
    $F_i \leftarrow$ CORTE-MÍNIMO(G', s_i, t);
fim
retorna $\bigcup_{i=1}^{k} F_i$;

Algoritmo 3: Algoritmo combinatório para o corte multisseparador mínimo

O Algoritmo 3 encontra uma solução para o problema do corte multisseparador mínimo utilizando como sub-rotina um algoritmo polinomial que soluciona o problema do corte s-t mínimo, esse último melhor abordado na seção 2.1.1. Primeiramente é computado o conjunto S, que contém todos os vértices selecionados s_i. Logo após é computado cada conjunto de corte de isolamento F_i correspondente a cada vértice selecionado s_i. Para encontrar cada conjunto F_i, é inicialmente criado um grafo G' por meio da sub-rotina INCLUIR-VÉRTICE-T, que adiciona um vértice t ao grafo G e interliga todos os outros vértices selecionados $s_1, \ldots, s_{i-1}, s_{i+1}, \ldots, s_k$ com t por meio de arestas com custo infinito. Os conjuntos F_i são encontrados com a execução da sub-rotina CORTE-MÍNIMO que computa o conjunto de corte referente ao corte s-t mínimo entre os vértices s_i e t. Após cada F_i ter sido encontrado, o algoritmo devolve como solução a união entre todos os conjuntos F_i, ou seja, $\bigcup_{i=1}^{k} F_i$.

A execução da sub-rotina INCLUIR-VÉRTICE-T adiciona um novo vértice t no grafo G com o principal intuito de encontrar o corte mínimo que isola um vértice selecionado

dos outros vértices selecionados. Apesar do algoritmo que soluciona o problema do corte s-t mínimo encontrar o corte mínimo que separa dois vértices, é possível utilizá-lo como sub-rotina para encontrar o corte mínimo que separa um conjunto de vértices U de outro conjunto de vértices W. Para isso, inicialmente adiciona-se no grafo G dois novos vértices s e t. Logo após, interliga-se o vértice s com todos os elementos de U e interliga-se o vértice t com todos os elementos de W. Os vértices s e t são interligados com os elementos dos conjuntos U e W, respectivamente, por meio de arestas com custos infinitos. Ao executar o algoritmo do corte s-t mínimo que separa os vértices s e t, como esse é um problema de minimização, as arestas adicionadas com custo infinito não são escolhidas como pertencentes ao conjunto de corte. Dessa forma, separa-se os conjuntos de vértices U e W por um conjunto de corte que possui apenas as arestas que já faziam parte do grafo original G. Como o Algoritmo 3 separa apenas um vértice selecionado dos outros, então não é necessário adicionar um novo vértice s.

A corretude, a complexidade e o fator de aproximação do algoritmo são abordados no teorema a seguir.

Teorema 4.5. *O Algoritmo 3 é um algoritmo 2-aproximado para o problema do corte multisseparador mínimo.*

Demonstração. Como k define a quantidade de vértices selecionados s_i, então o algoritmo computa um total de k conjuntos de corte F_i. Cada conjunto de corte é encontrado por meio da execução de um algoritmo polinomial que resolve o problema do corte s-t mínimo. Ao final, o algoritmo faz a união de k conjuntos e a devolve como resposta, resultando em um algoritmo que executa em tempo polinomial no tamanho da entrada.

A corretude do algoritmo foi provada pela análise acima, onde cada solução factível para o problema do corte multisseparador mínimo pode ser interpretada como uma união de cortes que isolam cada vértice selecionado s_i. Como o algoritmo encontra uma união de cortes que isolam cada vértice selecionado, então o algoritmo retorna uma solução factível.

Para encontrar a garantia de aproximação do algoritmo será realizada uma comparação entre o custo da união dos conjuntos F_i e o custo de uma solução ótima F^*. Seja F_i^* um

conjunto de corte de isolamento do vértice selecionado s_i na solução ótima F^*. Como F_i é obrigatoriamente o conjunto de corte referente ao corte mínimo que isola s_i dos outros vértices selecionados, então sabemos que o custo de F_i é menor ou igual ao custo de F_i^*, ou seja, $c(F_i) \leq c(F_i^*)$. Adaptando a inequação para a soma dos custos de todos os conjuntos de corte de isolamento, tem-se $\sum_{i=1}^{k} c(F_i) \leq \sum_{i=1}^{k} c(F_i^*)$. Assim,

$$\begin{aligned} \sum_{i=1}^{k} c(F_i) &\leq \sum_{i=1}^{k} c(F_i^*) \\ &\leq 2 \cdot c(F^*) \qquad (4.9) \\ &= 2 \cdot OPT, \qquad (4.10) \end{aligned}$$

sendo a inequação 4.9 válida pelo fato de que cada aresta pertencente ao conjunto de corte referente ao corte multisseparador mínimo ótimo F^* pode pertencer a no máximo dois conjuntos de corte de isolamento F_i^* e F_j^*. Isso acontece quando uma aresta $\{u, v\}$ pertencente ao conjunto de corte F^* possui seu vértice $u \in F_i^*$ e seu outro vértice $v \in F_j^*$. Na equação 4.10 apenas foi substituído o custo do corte ótimo multisseparador mínimo pela sua variável correspondente OPT, provando-se que o algoritmo é 2-aproximado. □

Ainda é possível melhorar um pouco mais a garantia de aproximação calculada. Sem perder generalidade, seja F_k o conjunto de corte de isolamento com maior custo dentre os conjuntos de corte de isolamento. Note que a união dos outros $k-1$ conjuntos de corte de isolamento é também um conjunto corte que isola o vértice s_k de todos os outros vértices selecionados. Dessa forma, segue o Teorema 4.6 que prova a garantia de aproximação desse algoritmo melhorado.

Teorema 4.6. *O algoritmo que retorna a união dos $k-1$ conjuntos de corte de isolamento com menor custo é um algoritmo $(2 - \frac{2}{k})$-aproximado que resolve o problema do corte multisseparador mínimo.*

Demonstração. Como a união dos $k - 1$ conjuntos de corte de isolamento com menor custo também é um conjunto de corte que isola o vértice selecionado s_k, então o algoritmo melhorado devolve um conjunto de corte que isola cada vértice s_i. Nesse algoritmo

melhorado, apenas é adicionada uma sub-rotina que ordena os k conjuntos de corte de isolamento com base nos seus custos antes de realizar a união, resultando também em um algoritmo polinomial.

Como F_k é o conjunto de corte de isolamento com maior custo, então as inequações $c(F_i) \leq c(F_k)$ e $\sum_{i=1}^{k} c(F_i) \leq \sum_{i=1}^{k} c(F_k)$ são verdadeiras. Ou seja, $\sum_{i=1}^{k} c(F_i) \leq k \cdot c(F_k)$ e $\frac{1}{k}\sum_{i=1}^{k} c(F_i) \leq c(F_k)$ também são verdadeiras. Dessa forma, a soma dos custos dos $k-1$ conjuntos de corte de isolamento com menor custo é:

$$\begin{aligned}
\sum_{i=1}^{k-1} c(F_i) &= \sum_{i=1}^{k} c(F_i) - c(F_k) \\
&\leq \sum_{i=1}^{k} c(F_i) - \frac{1}{k}\sum_{i=1}^{k} c(F_i) \\
&\leq (1 - \frac{1}{k})\sum_{i=1}^{k} c(F_i) \\
&\leq (1 - \frac{1}{k})\sum_{i=1}^{k} c(F_i^*) \\
&= 2 \cdot (1 - \frac{1}{k}) \cdot OPT.
\end{aligned}$$

□

Os Teoremas 4.5 e 4.6 foram apresentados por Williamson e Shmoys [37], tomando como base o algoritmo e provas mostradas por Dahlhaus, Johnson, Papadimitriou, Seymour e Yannakakis [11]. Será visto na próxima seção um algoritmo com um valor melhor de garantia de aproximação e que resolve o mesmo problema fazendo uso das propriedades de métricas.

4.2.2 Solução com programação linear

A seguir será apresentado um outro ponto de vista sobre o problema do corte multisseparador mínimo com o objetivo de encontrar o programa matemático inteiro que o modela. São dados um grafo $G = (V, E)$, custos $c_e \geq 0$ para todas as arestas $e \in E$ e k vértices selecionados $s_1, \ldots, s_k \in V$. O objetivo é particionar o conjunto de vértices V em sub-

conjuntos C_i, de forma que cada vértice selecionado $s_i \in C_i$ para todo i e a soma dos custos das arestas em $F = \bigcup_{i=1}^{k} \delta(C_i)$ seja minimizada. Para relembrar, $\delta(S)$ é o conjunto de todas as arestas pertencentes a E que possuem somente um vértice em S. O algoritmo abordado nessa seção foi apresentado por Călinescu, Karloff e Rabani [10].

No programa matemático inteiro que modela o problema, tem-se k variáveis diferentes $x_u^1, \ldots, x_u^k$ para cada vértice $u \in V$. Uma variável x_u^i é igual a 1 se e somente se o vértice u for atribuído ao subconjunto C_i na solução, ou seja $u \in C_i$, e 0 caso contrário. No mesmo programa existem também k variáveis $z_e^1, \ldots, z_e^k$ para cada aresta $e \in E$. Uma variável z_e^i é igual a 1 se e somente se a aresta e faz parte do conjunto de corte F, ou seja $e \in \delta(C_i)$, e 0 caso contrário. Como visto, cada aresta pertencente ao conjunto de corte está obrigatoriamente em algum $\delta(C_i)$ e em algum $\delta(C_j)$ com $i \neq j$, de forma que $\sum_{i=1}^{k} z_e^i = 2$ para toda aresta e pertencente ao conjunto de corte. Assim sendo, a função objetivo do programa matemático inteiro é minimizar $\frac{1}{2} \sum_{e \in E} c_e \sum_{i=1}^{k} z_e^i$. Essa função objetivo resultará exatamente no valor de custo da solução factível $F = \bigcup_{i=1}^{k} \delta(C_i)$.

Agora são criadas as restrições do programa inteiro. Como visto, cada vértice $u \in V$ precisa estar atribuído a exatamente um subconjunto C_i. Para isso, adiciona-se o conjunto de restrições $\sum_{i=1}^{k} x_u^i = 1$ para todo $u \in V$. O próximo conjunto de restrições é criado para garantir que a variável z_e^i seja igual a 1 somente quando a aresta e fizer parte do corte, ou seja, quando $e \in \delta(C_i)$. Para isso, adiciona-se o conjunto de restrições $z_e^i \geq |x_u^i - x_v^i|$ para toda aresta $e = \{u, v\} \in E$, que garante que a aresta e pertença ao conjunto de corte se u e v pertencem a subconjuntos diferentes. Como a função objetivo busca minimizar z_e^i com valores não negativos, então pode-se assumir que $z_e^i = |x_u^i - x_v^i|$. Nesse último conjunto de restrições está presente uma função modular (que não é linear), podendo esse conjunto de restrições ser reescrito por meio de outros dois conjuntos de restrições lineares $z_e^i \geq x_u^i - x_v^i$ e $z_e^i \geq x_v^i - x_u^i$. Finalizando, são adicionados outros dois conjuntos de restrições, um para assegurar que um vértice selecionado s_i pertença ao subconjunto C_i e outro para que os valores das variáveis x_e^i sejam iguais a 0 ou 1. Para isso, adiciona-se respectivamente os conjuntos de restrições $x_{s_i}^i = 1$ e $x_u^i \in \{0, 1\}$. Segue o programa linear

inteiro que modela o problema.

$$
\begin{aligned}
\text{minimizar} \quad & \tfrac{1}{2} \sum_{e \in E} c_e \sum_{i=1}^{k} z_e^i \\
\text{sujeito a} \quad & \sum_{i=1}^{k} x_u^i = 1, && \forall u \in V, \\
& z_e^i \geq x_u^i - x_v^i, && \forall e = \{u, v\} \in E, \\
& z_e^i \geq x_v^i - x_u^i, && \forall e = \{u, v\} \in E, \\
& x_{s_i}^i = 1, && i = 1, \ldots, k, \\
& x_u^i \in \{0, 1\}, && \forall u \in V, i = 1, \ldots, k.
\end{aligned}
$$

Uma das propriedades da solução do programa linear inteiro modelado é que a união de todos os subconjuntos C_i resulta obrigatoriamente no conjunto de vértices V. Relembrando, um conjunto C_i é o conjunto de vértices alcançáveis partindo do vértice selecionado s_i após a remoção das arestas pertencentes ao conjunto de corte. Em um grafo desconexo, por exemplo, a união de todos os subconjuntos C_i pode não resultar no conjunto V. No caso onde se tem um grafo desconexo G particionado em dois componentes conexos G_1 e G_2 e vértices selecionados $s_1, \cdots, s_k$ que se encontram todos em G_1, pode-se perceber que os vértices de G_2 são inalcançáveis partindo de qualquer vértice selecionado. Quando isso acontece, a união de todos os subconjuntos C_i não resulta em V, fazendo com que algumas restrições $\sum_{i=1}^{k} x_u^i = 1$ não sejam satisfeitas. Porém existem respostas factíveis para o programa modelado mesmo quando ocorre casos como o exemplo citado.

Será mostrado que existe uma resposta factível para o programa inteiro modelado mesmo quando ocorre situações onde existem vértices inalcançáveis partindo de qualquer vértice selecionado. Continuando o raciocínio sobre o evento supracitado, seja S o conjunto de todos os vértices que são inalcançáveis partindo de qualquer vértice selecionado s_i no grafo desconexo G. Aplique um corte multisseparador mínimo no grafo G. Após o corte, seja C_i o conjunto dos vértices alcançáveis partindo do vértice selecionado s_i, para todo i. Para um i qualquer faça $C_i' = C_i \cup S$ e para os outros subconjuntos restantes faça $C_j' = C_j$, para todo $j \neq i$. Faça as soluções $F = \bigcup_{i=1}^{k} \delta(C_i)$ e $F' = \bigcup_{i=1}^{k} \delta(C_i')$. Percebe-se que a união do subconjunto S com qualquer C_i não muda o corte, sendo $F = F'$, e a união de todos os conjuntos C_i' resulta no conjunto V. Dessa forma, o programa inteiro modelado também devolve respostas factíveis para grafos desconexos.

O relaxamento do programa linear inteiro apresentado tem grande relação com o cálculo da distância na norma ℓ_1. A distância na norma ℓ_1 pode ser calculada pela equação 2.7, apresentada na seção 2.4.3. Não obstante, o relaxamento do programa linear apresentado também faz uso do conceito de n-simplexo. Um n-simplexo é um conjunto infinito de pontos que determina um politopo regular de $n+1$ dimensões. A Definição 4.1 [37] formaliza o conceito apresentado.

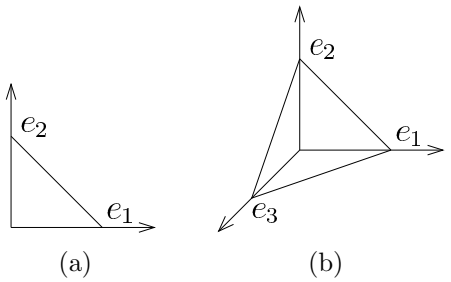
Definição 4.1. *Um n-simplexo é um subconjunto de pontos do* $\mathbb{R}^{n+1}$ *dados por:*

$$\Delta_n = \left\{ (x^1, \ldots, x^{n+1}) \in \mathbb{R}^{n+1} \left| \sum_{i=1}^{n+1} x^i = 1 \text{ e } x^i \geq 0 \text{ para todo } i \right. \right\}.$$

O politopo definido por um n-simplexo possui um total de $n+1$ vértices (pontos extremais) definidos de tal forma que:

$$\begin{aligned} e_1 &= (1, 0, \ldots, 0), \\ &\vdots \\ e_{n+1} &= (0, 0, \ldots, 1). \end{aligned}$$

Por exemplo, o 1-simplexo, que possui dois pontos extremais, é representado por uma reta no plano e o 2-simplexo, que possui três pontos extremais, é representado por um polígono. O 1-simplexo e o 2-simplexo são ilustrados na Figura 4.4(a) e na Figura 4.4(b), respectivamente.



(a) (b)

Figura 4.4: Representações do 1-simplexo e do 2-simplexo

Dadas as devidas definições, será obtido um programa linear relaxado por meio do pro-

grama linear inteiro apresentado. O conjunto de restrições $z_e^i = |x_u^i - x_v^i|$, representado pelos conjuntos de restrições $z_e^i \geq x_u^i - x_v^i$ e $z_e^i \geq x_v^i - x_u^i$, pode ser diretamente substituído na função objetivo por se tratarem de igualdades. Após essa substituição, a função objetivo passa a ser: $\frac{1}{2} \sum_{e=\{u,v\} \in E} c_e \sum_{i=1}^{k} |x_u^i - y_v^i|$. Fazendo uso da equação 2.7, determina-se a função objetivo do programa linear relaxado como $\frac{1}{2} \sum_{e=\{u,v\} \in E} c_e \left\| x_u - x_v \right\|_1$.

Modelando as restrições do programa linear relaxado e fazendo uso da Definição 4.1, o conjunto de restrições $\sum_{i=1}^{k} x_u^i = 1$ passa a ser $x_u \in \Delta_{k-1}$. O conjunto de restrições $x_{s_i}^i = 1$ passa a ser $x_{s_i} = e_i$, fazendo com que cada ponto selecionado s_i seja posicionado em um dos pontos extremais do politopo formado pelo $(k-1)$-simplexo. Finalmente, o conjunto de restrições $x_u^i \in \{0, 1\}$ é substituído pela relaxação $x_u^i \geq 0$. Segue o programa linear relaxado.

$$
\begin{array}{lll}
\text{minimizar} & \frac{1}{2} \sum_{\{u,v\} \in E} c_{uv} \left\| x_u - x_v \right\|_1 & \\
\text{sujeito a} & x_u \in \Delta_{k-1}, & \forall u \in V, \\
& x_{s_i} = e_i, & i = 1, \ldots, k, \\
& x_u^i \geq 0, & \forall u \in V, i = 1, \ldots, k.
\end{array}
\tag{4.11}
$$

Esse programa linear relaxado faz uso da norma ℓ_1 e encontra vetores que possuem suas caudas posicionadas na origem do sistema de coordenadas e o seu ponto final sobre o $(k-1)$-simplexo. Dessa forma, os vetores retornados como resposta do programa linear também podem ser visualizados como pontos sobre o $(k-1)$-simplexo. Sendo o $(k-1)$-simplexo um conjunto infinito de pontos, o programa relaxado passa a admitir um conjunto maior de soluções factíveis em comparação ao programa linear inteiro e, claramente, a sua solução precisa de um tratamento especial para que seja obtida uma solução factível para o problema do corte multisseparador mínimo. A Figura 4.5 ilustra a representação dos vetores por pontos sobre um 2-simplexo.

A ideia geral para encontrar um corte baseado na solução do programa linear relaxado consiste em atribuir ao subconjunto C_i os vértices representados pelos pontos mais

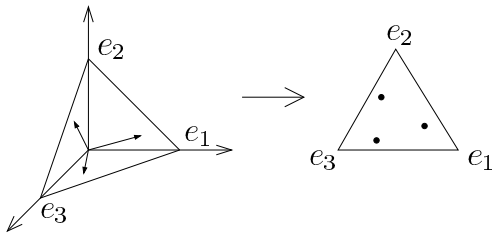


Figura 4.5: Vetores representados por pontos sobre um 2-simplexo

próximos do ponto extremal que representa o vértice s_i no $(k-1)$-simplexo. Para isso, será necessário redefinir o conceito de bola que será utilizado no algoritmo apresentado nessa seção. Seja $B(s_i, r)$ o conjunto de vértices representados pelos pontos x_u internos a uma bola de raio r na norma ℓ_1 posicionada sobre o vértice e_i no $(k-1)$-simplexo. Como a maior distância na norma ℓ_1 entre dois pontos sobre o politopo formado pelo $(k-1)$-simplexo é igual a 2, e como o raio r da bola posicionada sobre os vértices é limitado ao valor 1 pelo algoritmo que veremos a seguir, então B é definido como:

$$B(s_i, r) = \left\{ u \in V : \frac{1}{2} \left\| e_i - x_u \right\|_1 \leq r \right\}.$$

Pelo fato de ser utilizada a distância na norma ℓ_1 como métrica para a criação dos conjuntos B, então a bola, que no uso mais comum da palavra possui limites arredondados, possui como limite um plano. Todo ponto encontrado na solução do programa linear obrigatoriamente pertence ao $(k-1)$-simplexo, de forma que o politopo formado por ele determina os limites do espaço onde esses pontos estarão posicionados. Como visto, existe um conjunto de restrições no programa linear relaxado que impõe isso e outro que garante que cada vértice selecionado é representado por um dos pontos extremais do politopo. É possível visualizar na Figura 4.6 que os pontos estão obrigatoriamente sobre ao politopo formado pelo $(k-1)$-simplexo e que os limites das bolas posicionadas em cada vértice s_i são, na verdade, planos.

A sequência de execução do algoritmo aleatorizado começa com a determinação do valor do raio r, que é um valor aleatório amostrado de uma distribuição uniforme que varia entre 0 e 1. Logo após, determina-se aleatoriamente uma permutação π referente aos índices dos k subconjuntos. Para cada índice $\pi(i)$, com i iniciando em 1 e finalizando em

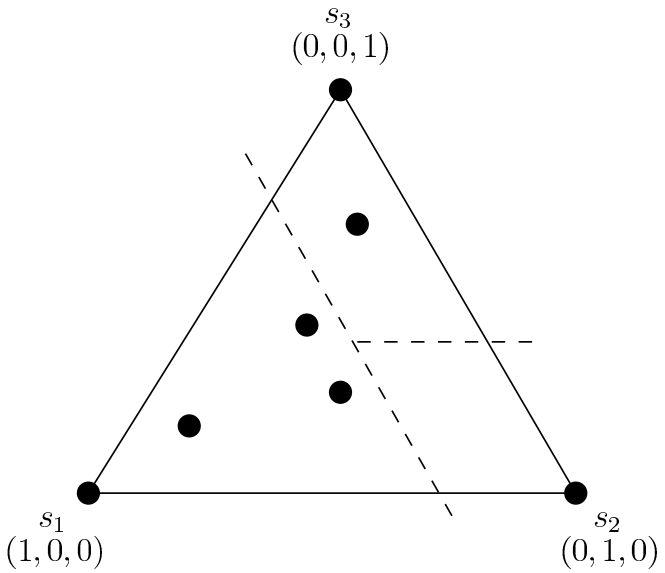


Figura 4.6: Representação geométrica dos pontos em um 2-simplexo
[37]

$k-1$, o algoritmo atribui a $C_{\pi(i)}$ todos os vértices representados pelos pontos pertencentes a $B(s_{\pi(i)}, r)$ que ainda não foram atribuídos a nenhum subconjunto anteriormente. Ao final, para que todos os vértices em V sejam atribuídos a um subconjunto, os vértices restantes, que não foram atribuídos a algum subconjunto, são atribuídos a $C_{\pi(k)}$. O Algoritmo 4 [37] define formalmente essa sequência de execução.

Seja x uma solução do programa linear relaxado 4.11;
para $i \leftarrow 1$ **até** k **faça** $C_i \leftarrow \emptyset$;
Escolha $r \in (0, 1)$ aleatoriamente de uma distribuição uniforme;
Escolha uma permutação aleatória π de $\{1, \ldots, k\}$;
$X \leftarrow \emptyset$;
para $i \leftarrow 1$ **até** $k - 1$ **faça**
    $C_{\pi(i)} \leftarrow B(s_{\pi(i)}, r) - X$;
    $X \leftarrow X \cup C_{\pi(i)}$;
fim
$C_{\pi(k)} \leftarrow V - X$;
retorna $F = \bigcup_{i=1}^{k} \delta(C_i)$

Algoritmo 4: Algoritmo aleatorizado para o corte multisseparador mínimo

A Figura 4.6 ilustra a execução do Algoritmo 4. Nesse exemplo, o algoritmo executou com a constante $k = 3$, formando um 2-simplexo, e com a permutação aleatória $\pi = (1, 3, 2)$. Na primeira execução do segundo *para*, quando $i = 1$, foram atribuídos a C_1 os

três vértices representados pelos pontos na bola centralizada em s_1. Na segunda execução do mesmo laço, foi atribuído a C_3 o outro vértice representado pelo ponto pertencente a bola centralizada em s_3. Percebe-se também que existe um outro ponto que também pertence a $B(s_3, r)$, porém o vértice representado por ele não foi atribuído a C_3 pelo fato dele já pertencer a C_1. Ao final, o algoritmo devolve como solução a união de todos os conjuntos $\delta(C_i)$, esses formados pelas arestas que possuem exatamente um vértice em C_i.

O início da análise do algoritmo apresentado se dá sobre a própria solução devolvida por dele. Para que seja possível o cálculo do valor esperado da garantia aproximação, faz-se necessário primeiramente analisar quando uma aresta qualquer $\{u, v\}$ pertence a algum $\delta(C_i)$. Durante a execução do algoritmo, uma aresta $\{u, v\}$ é dita *escolhida* pelo subgrafo induzido que contém s_i se e somente se i é o primeiro índice na permutação aleatória que possui pelo menos um vértice de $\{u, v\}$ pertencente a $B(s_i, r)$. O evento de um vértice s_i *escolher* a aresta $\{u, v\}$ será tratado pela variável aleatória S_i. Quando o vértice selecionado s_i *escolher* a aresta $\{u, v\}$, então S_i será igual a 1, senão S_i será igual a 0.

A variável aleatória S_i diz respeito a todas as arestas que estão ligadas em algum vértice de C_i por uma ponta ou por ambas as pontas. Porém busca-se definir o evento onde as arestas apareçam antes na permutação aleatória e têm somente um vértice em C_i, ou seja, quando uma aresta $\{u, v\}$ é *escolhida* por s_i e também quando somente um vértice de $\{u, v\}$ está em C_i. Quando esses dois eventos acontecem simultaneamente, então acontece o caso de $\{u, v\} \in \delta(C_i)$ durante a execução do algoritmo. O evento que acontece quando somente um vértice de uma aresta $\{u, v\}$ pertence a C_i será referenciado como o evento que o vértice s_i *corta* a aresta $\{u, v\}$. Esse evento será tratado pela variável aleatória X_i, sendo que X_i é igual a 1 quando somente um vértice de $\{u, v\}$ pertence a C_i ou X_i é igual a 0 caso contrário. Assim sendo, a probabilidade de uma aresta $\{u, v\}$ pertencer a $\delta(C_i)$ é denotada como $\Pr[S_i \wedge X_i]$.

Levando em consideração as variáveis aleatórias referentes aos eventos apresentados, a probabilidade de uma aresta $\{u, v\}$ pertencer ao conjunto de corte é calculada pela equação $\Pr[\{u, v\} \text{ estar no corte}] = \sum_{i=1}^{k} \Pr[S_i \wedge X_i]$. Será encontrado qual é o limitante

superior de $\sum_{i=1}^{k} \Pr[S_i \wedge X_i]$. Os próximos lemas são importantes para encontrar esse valor. Intuitivamente, se x_u^i é próximo de 1 então $u \in C_i$, ou seja, $u \in B(s_i, r)$. Se x_u^i é próximo de 0 então $u \notin C_i$, ou seja, $u \notin B(s_i, r)$. O Lema 4.4 busca relacionar x_u^i com o fato de u pertencer ou não a $B(s_i, r)$.

Lema 4.4. *Um vértice $u \in B(s_i, r)$ se e somente se $1 - x_u^i \leq r$.*

Demonstração. Pela definição de B, sabe-se que um vértice $u \in B(s_i, r)$ se e somente se a metade da distância na norma ℓ_1 entre o vértice u e o vértice s_i é menor ou igual ao raio r. Sabe-se também que o vértice s_i está posicionado sobre e_i, propriedade essa imposta por uma restrição do programa linear relaxado. A equivalência 4.12 é obtida por meio da aplicação da fórmula que calcula a distância na norma ℓ_1, mostrada na equação 2.7. Levando em consideração que e_i possui somente a coordenada i igual a 1 e as outras coordenadas zeradas, e que x_u^l sempre é positivo segundo a Definição 4.1, então a equivalência 4.13 pode ser obtida. Novamente com o uso da mesma definição, a equivalência 4.14 pode ser obtida pelo fato de $\sum_{l \neq i} x_u^l$ ser um valor positivo igual a $(1 - x_u^i)$. O resto da prova é obtido somente por manipulação algébrica.

$$\begin{aligned}
\frac{1}{2} \left\| e_i - x_u \right\|_1 \leq r &\equiv \frac{1}{2} \sum_{l=1}^{k} \left| e_i^l - x_u^l \right| \leq r && (4.12)\\
&\equiv \frac{1}{2} \left(\sum_{l \neq i} x_u^l + (1 - x_u^i) \right) \leq r && (4.13)\\
&\equiv \frac{1}{2}(2(1 - x_u^i)) \leq r && (4.14)\\
&\equiv 1 - x_u^i \leq r
\end{aligned}$$

□

Um vértice s_i *corta* uma aresta $\{u, v\}$ quando um dos vértices da aresta está na bola $B(s_i, r)$ e o outro não está. Sem perder generalidade, suponha que $u \in B(s_i, r)$ e $v \notin B(s_i, r)$. Como u pertence a $B(s_i, r)$, então $\frac{1}{2} \left\| e_i - x_u \right\|_1 \leq r$ e como v não está em $B(s_i, r)$, então $\frac{1}{2} \left\| e_i - x_v \right\|_1 > r$. Dessa forma, quando um vértice s_i *corta* uma aresta

$\{u, v\}$, o raio r tem o valor obrigatoriamente entre $\frac{1}{2}\|e_i - x_u\|_1$ e $\frac{1}{2}\|e_i - x_v\|_1$. Fazendo uso dessa afirmação, é calculada no Lema 4.5 a probabilidade de um vértice s_i *cortar* uma aresta $\{u, v\}$.

Lema 4.5. *A probabilidade de um vértice* s_i cortar *uma aresta* $\{u, v\}$ *é calculada por:* $\Pr[X_i] = |x_u^i - x_v^i|$.

Demonstração. Um vértice s_i *corta* uma aresta $\{u, v\}$ quando o raio r é um valor entre $\frac{1}{2}\|e_i - x_u\|_1$ e $\frac{1}{2}\|e_i - x_v\|_1$. Sendo as inequações mostradas na prova do Lema 4.4 equivalentes, então $\frac{1}{2}\|e_i - x_u\|_1 = 1 - x_u^i$ e $\frac{1}{2}\|e_i - x_v\|_1 = 1 - x_v^i$. É possível notar que durante o cálculo independe se x_v^i é maior ou menor do que x_u^i pelo fato de se utilizar uma função modular.

$$\begin{aligned}
\Pr[X_i] &= \Pr[\min(1 - x_u^i, 1 - x_v^i) \leq r < \max(1 - x_u^i, 1 - x_v^i)] \\
&= \left|1 - x_v^i - (1 - x_u^i)\right| \\
&= \left|x_u^i - x_v^i\right|
\end{aligned}$$

□

O Lema 4.6 é importante para calcular o limitante superior de $\sum_{i=1}^{k} \Pr[S_i \wedge X_i]$.

Lema 4.6. *Para qualquer índice* l *e dois vértices* u *e* $v \in V$, $\left|x_u^l - x_v^l\right| \leq \frac{1}{2}\|x_u - x_v\|_1$ *é verdade.*

Demonstração. Sem perder generalidade, assume-se que $x_u^l \geq x_v^l$. Com o uso da Definição 4.1, uma coordenada de um ponto pode ser encontrada pela diferença entre o valor 1 e o somatório das outras coordenadas. Com manipulações algébricas é possível chegar na prova do lema.

$$\left|x_u^l - x_v^l\right| = x_u^l - x_v^l$$

$$
\begin{aligned}
&= (1 - \sum_{j \neq l} x_u^j) - (1 - \sum_{j \neq l} x_v^j) \\
&= \sum_{j \neq l} (x_v^j - x_u^j) \\
&\leq \sum_{j \neq l} \left| x_u^j - x_v^j \right|
\end{aligned}
$$

Somando $\left| x_u^l - x_v^l \right|$ nos dois lados da inequação, tem-se $2 \left| x_u^l - x_v^l \right| \leq \|x_u - x_v\|_1$. $\square$

Para limitar superiormente $\sum_{i=1}^{k} \Pr[S_i \wedge X_i]$, será levado em consideração o ponto extremal do $(k-1)$-simplexo, representado pelo vértice selecionado s_ℓ, mais próximo de um dos vértices da aresta $\{u, v\}$. Será inicialmente calculado o limitante superior da probabilidade $\Pr[S_i \wedge X_i]$ para todo vértice s_i de forma que $i \neq \ell$. Ou seja, o limitante superior das probabilidades da aresta $\{u, v\}$ ser *escolhida* e *cortada* por um vértice s_i cujo ponto extremal não é o mais próximo de $\{u, v\}$ no $(k-1)$-simplexo. Logo após será calculado o limitante superior da probabilidade $\Pr[S_\ell \wedge X_\ell]$. Ao final são somados todos esses limites para definir o limitante superior de $\sum_{i=1}^{k} \Pr[S_i \wedge X_i]$. Esses cálculos são mostrados no Lema 4.7.

Lema 4.7. *O limitante superior* $\sum_{i=1}^{k} \Pr[S_i \wedge X_i] \leq \frac{3}{4} \|x_u - x_v\|_1$ *é verdade.*

Demonstração. Seja ℓ o índice do vértice selecionado representado pelo ponto extremal mais próximo da aresta $\{u, v\}$. O limitante superior da probabilidade $\Pr[S_i \wedge X_i]$ para todo vértice s_i sendo $i \neq \ell$ será calculado levando em consideração a possibilidade de ℓ ser posicionado antes e depois de i na permutação π. Dessa forma, sendo $i \neq \ell$:

$$
\begin{aligned}
\Pr[S_i \wedge X_i] &= \Pr[S_i \wedge X_i | \ell \text{ está antes de } i] \cdot \Pr[\ell \text{ está antes de } i] + \\
&\quad \Pr[S_i \wedge X_i | \ell \text{ está depois de } i] \cdot \Pr[\ell \text{ está depois de } i].
\end{aligned}
$$

Se $i \neq \ell$ e ℓ estiver antes de i na permutação π, então o vértice s_i nunca *escolherá* a aresta $\{u, v\}$. Isso acontece porque se $u \in B(s_i, r)$ ou $v \in B(s_i, r)$, então $u \in B(s_\ell, r)$ ou $v \in B(s_\ell, r)$ também é verdade. Como ℓ está antes de i na permutação, então a aresta $\{u, v\}$ será *escolhida* por s_ℓ antes. Sendo que o vértice s_i nunca *escolherá* a aresta $\{u, v\}$ se ℓ estiver antes de i na permutação π, então s_i nunca *escolherá*

e *cortará* a aresta $\{u, v\}$, ou seja, $\Pr[S_i \wedge X_i | \ell \text{ está antes de } i] = 0$. A probabilidade de ℓ estiver depois de i na permutação π é $\Pr[\ell \text{ está depois de } i] = \frac{1}{2}$. Sabendo que a probabilidade do evento X_i acontecer é maior ou igual a probabilidade dos eventos S_i e X_i acontecerem simultaneamente, então a inequação $\Pr[S_i \wedge X_i | \ell \text{ está depois de } i] \leq \Pr[X_i | \ell \text{ está depois de } i]$ é verdadeira. Realizando as devidas substituições, tem-se $\Pr[S_i \wedge X_i] \leq \Pr[X_i | \ell \text{ está depois de } i] \cdot \frac{1}{2} + 0$. Como o evento X_i é independente da ordem da permutação π, então $\Pr[X_i | \ell \text{ está depois de } i] = \Pr[X_i]$. Fazendo uso do Lema 4.5, quando $i \neq \ell$, tem-se:

$$\begin{aligned} \Pr[S_i \wedge X_i] &\leq \frac{1}{2} \cdot \Pr[X_i | \ell \text{ está depois de } i] \\ &= \frac{1}{2} \cdot \Pr[X_i] \\ &= \frac{1}{2} \left| x_u^i - x_v^i \right|. \end{aligned}$$

Calculado o limitante superior de $\Pr[S_i \wedge X_i]$ para $i \neq \ell$, parte-se para o caso onde $i = \ell$. Com o uso do Lema 4.5, o limitante superior da probabilidade do vértice s_ℓ *escolher* e *cortar* uma aresta $\{u, v\}$ é $\Pr[S_\ell \wedge X_\ell] \leq \Pr[X_\ell] = \left| x_u^\ell - x_v^\ell \right|$. Calculados os limites superiores de $\Pr[S_i \wedge X_i]$, sendo $i \neq \ell$, e de $\Pr[S_\ell \wedge X_\ell]$, então segue o cálculo do limitante superior de $\sum_{i=1}^{k} \Pr[S_i \wedge X_i]$. A inequação 4.15 é praticamente obtida por substituição direta de variáveis. O termo ausente é adicionado no somatório, gerando a equação 4.16. A equação 4.17 é obtida pela aplicação direta da equação 2.7. A inequação 4.18 foi obtida por meio do Lema 4.6 e o resto dos cálculos seguem somente com manipulações algébricas.

$$\begin{aligned} \sum_{i=1}^{k} \Pr[S_i \wedge X_i] &= \Pr[S_\ell \wedge X_\ell] + \sum_{i \neq \ell} \Pr[S_i \wedge X_i] \\ &\leq \left| x_u^\ell - x_v^\ell \right| + \frac{1}{2} \sum_{i \neq \ell} \left| x_u^i - x_v^i \right| && (4.15) \\ &= \frac{1}{2} \left| x_u^\ell - x_v^\ell \right| + \frac{1}{2} \sum_{i=0}^{k} \left| x_u^i - x_v^i \right| && (4.16) \\ &= \frac{1}{2} \left| x_u^\ell - x_v^\ell \right| + \frac{1}{2} \left\| x_u - x_v \right\|_1 && (4.17) \end{aligned}$$

$$\begin{aligned} &\leq \frac{1}{2} \cdot \frac{1}{2} \|x_u - x_v\|_1 + \frac{1}{2} \|x_u - x_v\|_1 \qquad (4.18) \\ &= \frac{3}{4} \|x_u - x_v\|_1 \end{aligned}$$

□

Encontrado o limitante superior de $\sum_{i=1}^{k} \Pr[S_i \wedge X_i]$, automaticamente foi encontrado também o limitante superior de $\Pr[\{u, v\}$ estar no corte$]$. Esse limite é importante para que seja calculada a garantia de aproximação do algoritmo apresentado e para que seja provado o Teorema 4.7.

Teorema 4.7. *O Algoritmo 4 é um algoritmo aleatorizado $\frac{3}{2}$-aproximado para o problema do corte multisseparador mínimo.*

Demonstração. Como mostrado no começo da seção, o programa linear modela o problema do corte multisseparador mínimo. Ao decidir a qual subconjunto C_i pertencerá cada vértice em V, o algoritmo processa $k-1$ subconjuntos C_i diferentes, sendo que cada vez são comparados no máximo $O(|V|)$ pontos. Ao final, o algoritmo realiza a união dos k conjuntos de corte $\delta(C_i)$. Dessa forma é devolvido um corte factível em tempo polinomial no tamanho da entrada.

Seja W a variável aleatória que denota o custo do corte. Seja Z_{uv} a variável aleatória que é igual a 1 se e somente se u e v estão em partições diferentes de V ou é igual a 0 caso contrário. Dado o limitante superior da probabilidade de uma aresta $\{u, v\}$ pertencer no conjunto de corte, calculado no Lema 4.7, o cálculo do valor esperado da garantia de aproximação segue.

$$\begin{aligned} \mathsf{E}[W] &= \mathsf{E}\left[\sum_{e=\{u,v\}\in E} c_e Z_{uv}\right] \\ &= \sum_{e=\{u,v\}\in E} c_e \mathsf{E}[Z_{uv}] \qquad (4.19) \\ &= \sum_{e=\{u,v\}\in E} c_e \Pr[\{u, v\} \text{ estar no corte}] \qquad (4.20) \end{aligned}$$

$$\leq \sum_{e=\{u,v\}\in E} c_e \frac{3}{4} \|x_u - x_v\|_1 \tag{4.21}$$

$$= \frac{3}{2} \cdot \frac{1}{2} \sum_{e=\{u,v\}\in E} c_e \|x_u - x_v\|_1 \tag{4.22}$$

$$\leq \frac{3}{2} \cdot OPT$$

A equação 4.19 é obtida com a aplicação da propriedade da linearidade da esperança. Sabe-se também que o valor esperado de constantes é igual ao valor da própria constante. Como a variável aleatória Z_{uv} pode assumir somente os valores 0 e 1, então o seu valor esperado é igual a probabilidade de Z_{uv} ser igual a 1. Essa probabilidade corresponde a probabilidade da aresta $\{u, v\}$ pertencer ao conjunto de corte, como mostra a equação 4.20. A inequação 4.21 e a equação 4.22 são obtidas com base no Lema 4.7 e por manipulação algébrica, respectivamente. Percebe-se a presença da função objetivo do programa linear relaxado na equação 4.22, que é substituída pelo seu limitante superior OPT, comprovando o valor esperado da garantia de aproximação. □

O Algoritmo 4, os Lemas 4.4, 4.5, 4.6 e 4.7 e o Teorema 4.7 foram todos apresentados por Williamson e Shmoys [37], tomando como base o algoritmo e provas mostradas por Călinescu, Karloff e Rabani [10]. O uso de relaxamento de programação linear e de aleatoriedade no desenvolvimento da solução do problema propiciou o cálculo de uma garantia de aproximação melhor em relação ao algoritmo apresentado na seção 4.2.1.

Karger, Klein, Stein, Thorup e Young [22] apresentaram um algoritmo que possui uma garantia de aproximação melhor em comparação ao algoritmo que utiliza programação linear apresentado por Călinescu, Karloff e Rabani. Foi provado nesse trabalho que, para um corte multisseparador mínimo com 3 subconjuntos disjuntos, o algoritmo possui garantia de aproximação igual a $\frac{12}{11}$. Karger, Klein, Stein, Thorup e Young acreditam que essa garantia de aproximação é a melhor possível para algoritmos que fazem uso da resposta do programa linear relaxado.

4.3 Algoritmo para o corte mais disperso

A métrica em árvore, apresentada na seção 3.2.2.1, é também uma ferramenta poderosa que pode ser utilizada na elaboração de algoritmos de aproximação para uma variedade de problemas. Será visto que a imersão de espaços métricos finitos gerais em métricas em árvores pode facilitar o desenvolvimento e análise de algoritmos. Será apresentado nesta seção um algoritmo aleatorizado de aproximação que soluciona o problema do corte mais disperso com uso de imersão de espaços métricos finitos gerais em métricas em árvores.

Relembrando, no problema do corte mais disperso, é dado como entrada um grafo não direcionado $G = (V, E)$, capacidades não negativas c_e para cada aresta $e \in E$ e k pares de vértices selecionados (s_i, t_i) com demandas não negativas associadas d_i, sendo s_i e $t_i \in V$. O objetivo é encontrar o corte $S \subseteq V$ e $V - S$ que minimiza a razão $\rho(S) = \frac{\sum_{e \in \delta(S)} c_e}{\sum_{i:|S \cap \{s_i, t_i\}|=1} d_i}$. Uma melhor explicação sobre o problema é dada na seção 2.1.4.

Esse problema pode ser modelado pelo programa linear inteiro 4.23. Nesse programa linear, a variável x_e representa a presença da aresta e no conjunto de corte e a variável y_i representa se o par (s_i, t_i) é separado pelo corte. A variável x_e é igual 1 se a aresta e pertence ao conjunto de corte $\delta(S)$ ou é igual a 0 caso contrário, enquanto que a variável y_i é igual 1 se o par (s_i, t_i) é separado pelo corte S ou é igual a 0 caso contrário. Dessa forma, a função objetivo do programa linear é a razão da soma das capacidades das arestas pertencentes ao conjunto de corte pela soma das demandas separadas por ele.

$$
\begin{aligned}
\text{minimizar} \quad & \frac{\sum\limits_{e \in E} c_e x_e}{\sum\limits_{i=1}^{k} d_i y_i} & & \\
\text{sujeito a} \quad & \sum_{e \in P} x_e \geq y_i, & \forall P \in \mathcal{P}_i, 1 \leq i \leq k, & \\
& x_e \in \{0, 1\}, & \forall e \in E, & \\
& y_i \in \{0, 1\}, & i = 1, \ldots, k. &
\end{aligned}
\tag{4.23}
$$

O primeiro conjunto de restrições do programa linear é responsável por identificar se a demanda i é separada pelo corte. O conjunto P contém as arestas que formam um

caminho possível para enviar a demanda i no grafo G, partindo do vértice s_i com destino ao vértice t_i. Ainda no mesmo conjunto de restrições, um conjunto $\mathcal{P}_i$ contém os conjuntos que formam todos os caminhos possíveis de interligar s_i a t_i no grafo G. Assim sendo, existe uma restrição para cada caminho possível de ser realizado de um vértice s_i até um vértice t_i, para cada demanda i. Se existe um caminho P que não tem arestas no corte, então $\sum_{e \in P} x_e = 0$, obrigando, pelo primeiro conjunto de restrições, que $y_i = 0$. Caso contrário, se todos os caminhos entre s_i e t_i tiverem pelo menos uma aresta no corte, então y_i assume o valor 1, pois a função objetivo busca maximizar y_i.

Um ponto importante sobre o programa linear 4.23 é que, mesmo se ele for relaxado, ainda existirá um número exponencial de restrições a serem criadas para representar cada um dos caminhos possíveis de serem gerados para todas as demandas. Williamson e Shmoys apresentam uma forma de solucionar o programa linear relaxado em tempo polinomial por meio do método dos elipsoides [37]. A solução adotada no presente trabalho é a mesma apresentada por Aumann e Rabani [4], obtendo-se um programa linear equivalente com um número polinomial de restrições por meio de uma relaxação do programa linear 4.23 seguido de uma imersão no espaço ℓ_∞^k. O valor k representa a quantidade de pares (s_i, t_i) e não será explicada a imersão realizada pelos autores nem a prova que indica que o programa linear 4.24 é uma relaxação do programa 4.23, podendo essa última ser encontrada no trabalho citado. O programa linear relaxado 4.24 será tomado como base para o problema a partir de então.

$$
\begin{aligned}
\text{minimizar} \quad & \sum_{e \in E} c_e x_e \\
\text{sujeito a} \quad & \sum_{i=1}^{k} d_i y_i = 1, \\
& x_e \geq p_i^u - p_i^v, \quad \forall e = \{u, v\} \in E, i = 1, \ldots, k, \\
& x_e \geq p_i^v - p_i^u, \quad \forall e = \{u, v\} \in E, i = 1, \ldots, k, \\
& y_i \leq p_i^{t_i} - p_i^{s_i} \quad i = 1, \ldots, k, \\
& p_i^{t_i} \geq 0, \qquad\quad i = 1, \ldots, k, \\
& p_i^{s_i} \leq 0, \qquad\quad i = 1, \ldots, k.
\end{aligned}
\tag{4.24}
$$

Nesse programa linear, as variáveis p_i^u são as coordenadas do ponto p^u, que representa o vértice u no espaço métrico ℓ_∞^k, ou seja, $p^u = (p_1^u, \ldots, p_k^u)$. Apesar do programa linear estar em função de p, os valores de y e x também são encontrados caso ele seja resolvido. Após solucionar o programa linear 4.24, somente os valores de x são utilizados no algoritmo em questão. A variável x descreve uma métrica, onde x_e é igual a 1 se a aresta e pertence ao conjunto de corte ou é igual a 0 caso contrário, como explicado detalhadamente na seção 2.4.1. Porém, como o programa linear é uma relaxação, então os valores x_e não são obrigatoriamente iguais a 0 ou 1. Nesse caso, têm-se um espaço métrico finito geral (V, x) definido pela métrica x sobre os vértices contidos em V.

Aumann e Rabani apresentaram um algoritmo polinomial que encontra o melhor corte caso o problema seja modelado por um espaço métrico definido pela norma ℓ_1. Como esse não é necessariamente o caso, para que o problema do corte mais disperso seja solucionado, basta realizar uma imersão do espaço métrico finito geral (V, x) em um espaço métrico finito que seja ℓ_1-imersível e submeter o espaço métrico resultante ao algoritmo apresentado pelos autores. Como o algoritmo encontra a resposta ótima, o fator de aproximação α será simplesmente igual a distorção da imersão de (V, x) em um espaço métrico finito ℓ_1-imersível. O próximo teorema mostra a existência de um procedimento polinomial de imersão de espaços métricos finitos gerais em um espaço métrico definido pela norma ℓ_1.

Apesar do fator de aproximação do próximo algoritmo ser $O(\log n)$, devido ao uso de imersão em métricas em árvores, sabe-se que existem algoritmos com resultados melhores. Aumann e Rabani apresentaram um processo de imersão melhorado que diminui a taxa de aproximação do algoritmo para $O(\log k)$, sendo k o número de demandas [4]. Os autores apresentam um algoritmo que realiza a imersão do espaço métrico (V, x), esse obtido após a solução do programa linear 4.24, em um espaço métrico definido pela norma ℓ_1 de forma direta e soluciona o problema do corte mais disperso por meio de um procedimento que decompõe um espaço métrico definido pela norma ℓ_1. A solução estudada realiza a imersão de (V, x) em uma métrica em árvore e, posteriormente, realiza a imersão da métrica em árvore em um espaço métrico definido pela norma ℓ_1. A Figura 4.7 ilustra os processos de

imersão e as distorções realizadas por cada um deles. Apesar de se ter uma perda no grau de distorção, o principal intuito de apresentar a análise dessa outra versão do algoritmo é unicamente mostrar uma versão alternativa da demonstração, que foi desenvolvida com base no conteúdo coberto nos estudos realizados até o momento.

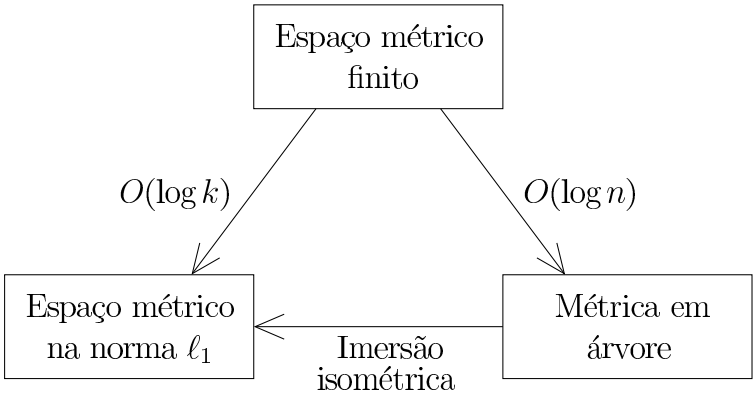


Figura 4.7: Imersões realizadas para solucionar o problema do corte mais disperso

Teorema 4.8. *Existe um procedimento aleatorizado polinomial que realiza a imersão* $f : V \to \mathbb{R}^m$ *de um espaço métrico finito* (V, d) *no espaço métrico definido pela norma* ℓ_1 *com distorção esperada igual a* $O(\log n)$*, ou seja,* $\mathsf{E}[\|f(u) - f(v)\|_1] \leq O(\log n) \cdot d_{uv}$ *e* $d_{uv} \leq \|f(u) - f(v)\|_1$ *sendo* u *e* $v \in V$*.*

Demonstração. Foi apresentado na seção 3.2.2.2 um algoritmo polinomial aleatorizado que realiza a imersão f_1 de espaços métricos finitos gerais (V, d) em métricas em árvores (V', T), ou seja, $f_1 : V \to V'$, sendo $V \subseteq V'$, com um fator de distorção esperado igual a $O(\log n)$. Isso é possível com a execução do Algoritmo 1 e com a prova do Teorema 3.5. Não obstante, foi apresentado no Teorema 3.6 um processo polinomial de imersão isométrica f_2 de métricas em árvores no espaço métrico definido por um conjunto de vetores no $\mathbb{R}^m$ e pela norma ℓ_1, sendo m igual à quantidade de arestas existentes na árvore T, ou seja, $f_2 : V' \to \mathbb{R}^m$. Dessa forma, existe um procedimento polinomial aleatorizado de imersão f de espaços métricos finitos gerais (V, d) no espaço métrico definido pela norma ℓ_1 de forma que $f : V \xrightarrow{f_1} V' \xrightarrow{f_2} \mathbb{R}^m$. Com a aplicação do Teorema 3.5, a imersão f_1 acontece com distorção $\mathsf{E}[T_{uv}] \leq O(\log n) \cdot d_{uv}$ e $d_{uv} \leq T_{uv}$. Como a imersão f_2 é isométrica, então $\mathsf{E}[\|f(u) - f(v)\|_1] \leq O(\log n) \cdot d_{uv}$ e $d_{uv} \leq \|f(u) - f(v)\|_1$ são verdadeiros. □

Aumann e Rabani apresentaram um processo de imersão otimizado de espaços métricos finitos gerais no espaço métrico definido pela norma ℓ_1 com distorção igual $O(\log k)$, sendo k igual ao número de pares de demandas [4]. Porém, optou-se pela criação de uma versão modificada do algoritmo para desenvolver uma demonstração por um caminho alternativo, com base no material estudado até então. Dando continuidade ao raciocínio, a simples imersão do espaço métrico finito geral (V, x) no espaço métrico finito (p, ℓ_1), sendo p o conjunto que contém os pontos que representam cada vértice em V, não resolve o problema como um todo. Note que esse conjunto p não tem relação com o conjunto p existente no programa linear 4.24, sendo que o primeiro contém os pontos do espaço métrico (p, ℓ_1) e o segundo contém pontos no espaço métrico ℓ_∞^k. Na verdade, apenas se obtém com essa imersão um espaço métrico finito conhecido como *métrica de corte*. Uma métrica de corte pode ser decomposta em uma soma de pesos determinados pelos vários cortes possíveis de serem realizados no conjunto V. O próximo lema realiza tal prova e serve como base para a prova da garantia de aproximação do algoritmo.

A decomposição do espaço métrico finito (V, x) em uma soma de pesos de cortes é inicialmente realizada pela imersão dele no espaço métrico finito (p, ℓ_1). Estando (V, x) imerso em (p, ℓ_1), todo o vértice de V tem um ponto que o representa em p. Sendo m o número de dimensões do espaço métrico após a imersão, então é possível decompor o espaço métrico finito (p, ℓ_1) em m dimensões. A disposição dos pontos p no espaço métrico faz com que em cada dimensão m exista uma sequência distinta dos pontos em p, partindo daquele com o menor valor de coordenada com destino ao de maior valor. São gerados consecutivos cortes com a realização de uma varredura em cada dimensão na norma ℓ_1. O Lema 4.8 mostra como é realizada a decomposição de um espaço métrico finito (p, ℓ_1) em cada dimensão e prova que um espaço métrico ℓ_1-imersível (V, d) corresponde a uma soma dos pesos de vários cortes obtidos por meio de varreduras em cada dimensão no espaço (p, ℓ_1). A função $\chi_{\delta(S)}(u, v)$ é igual a 1 se exatamente um vértice entre u e v pertence a S ou é igual a 0 caso contrário.

Lema 4.8. *Seja (V, d) um espaço métrico ℓ_1-imersivo e $f : V \to \mathbb{R}^m$ uma imersão*

associada. Então existe um $\lambda_S \geq 0$ *para todo* $S \subseteq V$*, sendo* u *e* $v \in V$*, de forma que:*

$$\|f(u) - f(v)\|_1 = \sum_{S \subseteq V} \lambda_S \chi_{\delta(S)}(u, v).$$

Além disso, no máximo $m \cdot n$ *dos* λ_S *são não zerados e um* λ_S *pode ser computado em tempo polinomial.*

Demonstração. A imersão f é um mapeamento entre o conjunto de vértices V de (V, d) no conjunto de pontos p de (p, ℓ_1), sendo p_u o ponto correspondente ao vértice $u \in V$ no espaço métrico (p, ℓ_1) e p_u^i o valor da i-ésima coordenada do ponto p_u. É possível visualizar na descrição do lema que existe um valor de λ_S para cada subconjunto de V. Existem, porém, alguns λ_S que são determinantes na decomposição de um espaço métrico finito definido pela norma ℓ_1. Os valores desses λ_S são encontrados por meio de uma varredura em cada dimensão i na norma ℓ_1. Os cálculos dos λ_S são realizados da seguinte forma: ordene os vértices em V em ordem crescente do valor da i-ésima coordenada dos pontos em p, ou seja, p_u^i. Após ordenado, e sem perder generalização, tem-se $p_{u_1}^i \leq \ldots \leq p_{u_n}^i$. Dessa forma, p_{u_1} possui o valor na coordenada i menor do que p_{u_2}, e assim por diante. Geometricamente, isso significa que p_{u_1} está localizado antes de p_{u_2} na dimensão i na norma ℓ_1. O valor de um $\lambda_{\{u_1,\ldots,u_l\}}$ é obtido pela diferença entre os valores da i-ésima coordenada dos pontos p_{u_l} e $p_{u_{l+1}}$, ou seja, $\lambda_{\{u_1,\ldots,u_l\}} = p_{u_{l+1}}^i - p_{u_l}^i$. A Figura 4.8 ilustra a explicação, onde foram encontrados os valores de $\lambda_{\{u_1\}}$, $\lambda_{\{u_1,u_2\}}$ e $\lambda_{\{u_1,u_2,u_3\}}$ na dimensão i no espaço métrico (p, ℓ_1).

São possíveis de calcular $n - 1$ valores diferentes de λ_S para cada dimensão, pelo fato de existir uma única sequência possível após a ordenação dos vértices. Dessa forma, são possíveis de serem gerados n subconjuntos S diferentes para cada dimensão, considerando também o próprio conjunto V ao final, ou seja, λ_V. Após efetuar o mesmo processo para cada uma das m dimensões, é possível de serem gerados no máximo $m \cdot (n - 1)$ valores positivos de λ_S, sendo que os valores de λ_V são iguais a 0 por definição. Para os outros subconjuntos S que não foram gerados durante todo o processo nas m dimensões, os seus respectivos valores de λ_S são também iguais a zero. Assim sendo, prova-se a última parte

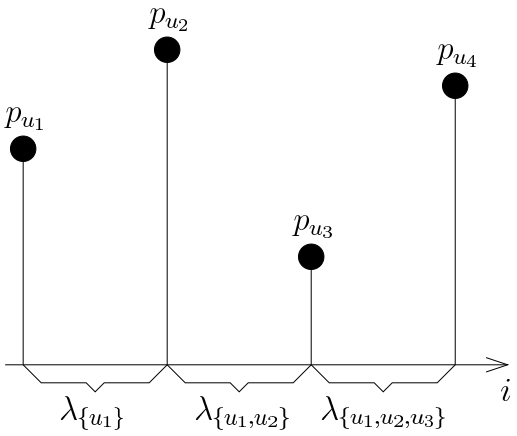


Figura 4.8: Varredura na dimensão i do espaço métrico (p, ℓ_1)

do lema, visto que são gerados um total de n valores de λ_S em m dimensões diferentes, sendo que o cálculo dos λ_S ou corresponde a uma subtração, ou a uma atribuição ao valor zero caso seja λ_V.

Será calculada a distância entre dois pontos p_{u_a} e p_{u_b} tomando como base a explicação dada. Sem perder generalização, seja p_{u_b} localizado antes de p_{u_a} na dimensão i. Sendo a distância entre dois pontos p_{u_a} e p_{u_b}, em uma determinada dimensão, igual a soma de todas as distâncias entre os pontos intermediários a p_{u_a} e p_{u_b} na mesma dimensão, então as igualdades a seguir são verdadeiras.

$$\begin{aligned} \left|p^i_{u_a} - p^i_{u_b}\right| &= p^i_{u_b} - p^i_{u_a} \\ &= \lambda_{\{u_1,\ldots,u_a\}} + \cdots + \lambda_{\{u_1,\ldots,u_{b-1}\}} \\ &= \sum_{l=a}^{b-1} \lambda_{\{u_1,\ldots,u_l\}} \end{aligned}$$

Com isso, segue a prova do teorema por meio do cálculo da distância entre dois vértices quaisquer u e v após a imersão. Sem perder generalização, faça $u = u_a$ e $v = u_b$. Como todos os outros valores de λ_S que não são computados são iguais a zero e, como a função χ é igual a zero caso dois vértices não são separados por S, então o caso pode ser generalizado para um único somatório que considera todos os subconjuntos possíveis de V. Com isso, obtém-se a prova do lema com a equação 4.25.

$$\begin{aligned} \|f(u) - f(v)\|_1 &= \sum_{i=1}^{m} \left| p^i_{u_a} - p^i_{u_b} \right| \\ &= \sum_{i=1}^{m} \sum_{l=a}^{b-1} \lambda_{\{u_1,\ldots,u_l\}} \\ &= \sum_{S \subseteq V} \lambda_S \chi_{\delta(S)(u,v)} \end{aligned} \tag{4.25}$$

□

Foi mostrada na prova do Lema 4.8 a possibilidade de uma métrica de corte (p, ℓ_1) ser decomposta em uma soma de pesos de vários cortes possíveis de serem realizados em cada dimensão. O Algoritmo 5 realiza tal decomposição para solucionar o problema do corte mais disperso. Inicialmente é solucionado o programa linear 4.24 que modela o problema do corte mais disperso. Com a solução do programa linear, tem-se um espaço métrico finito geral (V, x) que define as arestas com maior probabilidade de pertencerem ao conjunto de corte.

Logo após, realiza-se a imersão do espaço métrico (V, x) no espaço métrico finito (p, ℓ_1). Essa imersão é realizada pelo procedimento mostrado no Teorema 4.8. Para que essa imersão ocorra com sucesso, é importante que a métrica x seja escalada de forma que a menor distância entre dois vértices u e $v \in V$ seja maior ou igual a 1.

Após a determinação do espaço métrico (p, ℓ_1), o algoritmo pré-seleciona os subconjuntos S que possuem os menores valores de $\rho(S)$. Isso é feito por meio de uma varredura do espaço métrico (p, ℓ_1) em cada uma das m dimensões, gerando n subconjuntos S diferentes em cada dimensão de acordo com a posição dos pontos p no espaço métrico ℓ_1^m. Ao final do processo de varredura, são criados um total de $m \cdot n$ subconjuntos S diferentes. O algoritmo devolve como solução o subconjunto S que tem o menor valor de $\rho(S)$.

Percebe-se que o algoritmo é polinomial, visto que são gerados e avaliados no máximo $m \cdot n$ subconjuntos S, além dos processos de solução do programa linear e de imersão de (V, x) em (p, ℓ_1) também demandarem tempo polinomial. Antes de apresentar a prova de aproximação do algoritmo, será apresentado um fato algébrico importante.

Seja x uma solução do programa linear relaxado 4.24;
Imergir (V, x) em (p, ℓ_1) conforme o Teorema 4.8, de forma que $f(u) = (p_u^1, \ldots, p_u^m)$;
para $i \leftarrow 1$ **até** m **faça**
    Ordenar V na ordem crescente π de acordo com o valor da coordenada p_u^i;
    $S_1^i \leftarrow \{\pi_1\}$;
    para $j \leftarrow 2$ **até** n **faça** $S_j^i = S_{j-1}^i \cup \{\pi_j\}$;
fim
Obter o conjunto S^* de forma que $\rho(S^*) = \min_{i,j}(\rho(S_j^i))$;
retorna S^*

Algoritmo 5: Algoritmo para o corte mais disperso

Fato 4.3. *Dadas duas sequências de números positivos* $a_1, \cdots, a_k$ *e* $b_1, \cdots, b_k$*, então as seguintes inequações são verdadeiras:*

$$\min_{i=1,\cdots,k} \frac{a_i}{b_i} \leq \frac{\sum_{i=1}^{k} a_i}{\sum_{i=1}^{k} b_i} \leq \max_{i=1,\cdots,k} \frac{a_i}{b_i}.$$

Teorema 4.9. *O Algoritmo 5 é um algoritmo aleatorizado polinomial com garantia de aproximação igual a* $O(\log n)$ *para o problema do corte mais disperso.*

Demonstração. Como foi comentado anteriormente, o algoritmo executa em tempo polinomial no tamanho da entrada. Para provar a aproximação do algoritmo, parte-se do cálculo da razão ρ referente a suposta solução ótima S^*. Sabendo que a resposta ótima é aquela que possui o menor valor de $\rho(S_j^i)$, então inicia-se o cálculo de aproximação. A inequação 4.26 pode ser obtida com a aplicação direta do Fato 4.3. Faz-se necessário incluir a multiplicação por λ_S na mesma inequação pelo fato da generalização realizada. Como vários subconjuntos S não são processados pelo algoritmo, a presença da multiplicação por λ_S faz com que esses subconjuntos sejam desconsiderados no somatório. Isso acontece pelo fato dos valores de λ_S dos subconjuntos não processados serem iguais a 0, como explicado no Lema 4.8. A equação 4.27 pode ser obtida unicamente com a realização de cálculos algébricos. A equação 4.28 pode ser obtida com a aplicação direta do Lema 4.8, que prova que uma métrica de corte corresponde a uma soma dos pesos de vários cortes. A inequação 4.29 pode ser obtida com a aplicação do Teorema 4.8. É importante que não haja confusão entre os termos d_i e d_{uv}, sendo que o primeiro representa a quantidade de

demanda enviada e o segundo representa a distância na métrica d.

$$
\begin{aligned}
\rho(S^*) &= \min_{i,j}(\rho(S_j^i)) \\
&= \min_{S:\lambda_S>0} \rho(S) \\
&= \min_{S:\lambda_S>0} \frac{\sum_{e\in\delta(S)} c_e}{\sum_{i:|S\cap\{s_i,t_i\}|=1} d_i} \\
&= \min_{S:\lambda_S>0} \frac{\sum_{e\in E} c_e \cdot \chi_{\delta(S)}(e)}{\sum_{i=1}^{k} d_i \cdot \chi_{\delta(S)}(s_i,t_i)} \\
&= \min_{S:\lambda_S>0} \frac{\lambda_S \sum_{e\in E} c_e \cdot \chi_{\delta(S)}(e)}{\lambda_S \sum_{i=1}^{k} d_i \cdot \chi_{\delta(S)}(s_i,t_i)} \\
&\leq \frac{\sum_{S\subseteq V} \lambda_S \sum_{e\in E} c_e \cdot \chi_{\delta(S)}(e)}{\sum_{S\subseteq V} \lambda_S \sum_{i=1}^{k} d_i \cdot \chi_{\delta(S)}(s_i,t_i)} && (4.26) \\
&= \frac{\sum_{e\in E} c_e \sum_{S\subseteq V} \lambda_S \cdot \chi_{\delta(S)}(e)}{\sum_{i=1}^{k} d_i \sum_{S\subseteq V} \lambda_S \cdot \chi_{\delta(S)}(s_i,t_i)} && (4.27) \\
&= \frac{\sum_{e=\{u,v\}\in E} c_e \cdot \|f(u)-f(v)\|_1}{\sum_{i=1}^{k} d_i \cdot \|f(s_i)-f(t_i)\|_1} && (4.28) \\
&\leq \frac{\sum_{e=\{u,v\}\in E} c_e \cdot \|f(u)-f(v)\|_1}{\sum_{i=1}^{k} d_i \cdot d_{s_it_i}} && (4.29)
\end{aligned}
$$

Dando continuidade ao cálculo, a inequação 4.30 pode ser obtida com a aplicação da fórmula da esperança. Isso ocorre pelo fato de somente o termo $\|f(u)-f(v)\|_1$ possuir valor esperado e também devido à aplicação da linearidade da esperança que faz com que os outros termos se mantenham intactos. A inequação 4.31 pode ser obtida com a imersão realizada no Teorema 4.8. No dividendo, foi substituído o termo $\mathsf{E}[\|f(u)-f(v)\|_1]$ por $O(\log n)\cdot d_{uv}$ pelo fato do valor esperado de $\|f(u)-f(v)\|_1$ ser $O(\log n)\cdot d_{uv}$. A inequação 4.32 pode ser obtida apenas com a realização de cálculos algébricos.

$$
\begin{aligned}
\mathsf{E}[\rho(S^*)] &\leq \frac{\sum_{e=\{u,v\}\in E} c_e \cdot \mathsf{E}[\|f(u)-f(v)\|_1]}{\sum_{i=1}^{k} d_i \cdot d_{s_it_i}} && (4.30) \\
&\leq \frac{\sum_{e=\{u,v\}\in E} c_e \cdot O(\log n)\cdot d_{uv}}{\sum_{i=1}^{k} d_i \cdot d_{s_it_i}} && (4.31) \\
&\leq O(\log n)\cdot \frac{\sum_{e=\{u,v\}\in E} c_e \cdot d_{uv}}{\sum_{i=1}^{k} d_i \cdot d_{s_it_i}} && (4.32)
\end{aligned}
$$

Como é possível visualizar na inequação 4.32, estão presentes no somatório localizado no dividendo todos os valores das distâncias d_{uv} para cada aresta $e \in E$. Dessa forma, as distâncias d_{uv} correspondem ao valor das distâncias x_e, obtidas após a solução do programa 4.24. Assim sendo, a distância entre dois vértices u e v, sendo eles os vértices limitantes de uma aresta e, é igual a x_e, ou seja, $x_e = d_{uv}$ sendo $e = \{u, v\}$. Com isso, determina-se parte do dividendo da inequação 4.33. É possível obter o divisor da mesma inequação realizando a soma de todos os pesos das arestas e existentes no caminho P de s_i até t_i, ou seja, $y_i \leq \sum_{e \in P} x_e = d_{s_i t_i}$. A inequação 4.34 pode ser obtida porque o termo $\sum_{e \in E} c_e x_e$ corresponde ao valor função objetivo do programa relaxado 4.24, que é menor do que o valor da resposta ótima OPT, e pela substituição do termo $\sum_{i=1}^{k} d_i y_i$ por uma das restrições do mesmo programa. Assim, prova-se o valor esperado da garantia de aproximação do Algoritmo 5 para o problema do corte mais disperso.

$$
\begin{aligned}
\mathsf{E}[\rho(S^*)] &\leq O(\log n) \cdot \frac{\sum_{e=\{u,v\} \in E} c_e \cdot d_{uv}}{\sum_{i=1}^{k} d_i \cdot d_{s_i t_i}} \\
&\leq O(\log n) \cdot \frac{\sum_{e \in E} c_e x_e}{\sum_{i=1}^{k} d_i y_i} \qquad (4.33) \\
&\leq O(\log n) \cdot OPT \qquad (4.34)
\end{aligned}
$$

□

A existência de uma imersão de espaços métricos finitos gerais em ℓ_1 com distorção igual a $O(\log n)$ foi inicialmente apresentada por Bourgain [8]. Um algoritmo polinomial que realiza tal imersão foi apresentado por Linial, London e Rabinovich [27]. Os mesmos autores apresentaram também em seu trabalho um algoritmo $O(\log k)$ aproximado para o problema do corte mais disperso. De forma independente, Aumann e Rabani [4] apresentaram o seu algoritmo de aproximação com garantia de aproximação igual a $O(\log k)$. Arora, Rao e Vazirani realizaram uma importante descoberta e apresentaram um algoritmo $O(\sqrt{\log n})$-aproximado para o problema do corte mais disperso uniforme em [2] e [3], sendo essa uma outra versão do problema do corte mais disperso com valores

unitários de demanda para cada par de vértices selecionados. Os mesmos autores publicaram também uma versão modificada do algoritmo que soluciona o problema do corte mais disperso uniforme para solucionar o problema do corte mais disperso, com uma garantia de aproximação melhorada de $O(\sqrt{\log n} \log \log n)$ [1].

CAPÍTULO 5

CONCLUSÃO

Diferentes abordagens para tratar problemas NP-difíceis são constantemente estudadas ao redor do mundo. O presente trabalho apresentou algumas abordagens adotadas para solucionar três problemas NP-difíceis de cortes em grafos. Foram apresentadas análises de algoritmos de aproximação, que devolvem respostas com um certo fator de proximidade da resposta ótima do problema. Algumas técnicas utilizadas podem ser listadas como importantes no desenvolvimento de algoritmos.

Modelar problemas por meio de programação matemática se mostrou presente na maioria dos casos estudados. Como foram abordados problemas de otimização discreta, eles eram modelados por meio de um programa inteiro. Os algoritmos existentes até agora que solucionam programas matemáticos inteiros demandam tempo exponencial, por isso são realizados relaxamentos dos programas inteiros, gerando, assim, um programa similar que adota soluções contínuas. Esse método se mostrou eficaz na grande parte dos problemas, visto que programas com variáveis contínuas possuem algoritmos polinomiais que os solucionam. Em contrapartida, faz-se necessário um tratamento especial da resposta do programa solucionado. Como ele passa a assumir respostas com valores contínuos, a tomada de decisão para determinar o corte a partir da solução relaxada deve ser realizada de forma diferenciada. As soluções adotadas são executadas em tempo polinomial e os seus algoritmos tomam decisões com base em análises geométricas realizadas na norma de Minkowski ℓ_1.

O uso de programação matemática se mostrou importante também durante o cálculo da garantia de aproximação dos diferentes algoritmos. A função objetivo e as restrições dos programas lineares são constantemente utilizadas para realizar comparativos entre uma possível solução ótima e a solução devolvida pelo algoritmo. Foi apresentado também um método na seção 4.2.1 que não fazia uso de programa matemático, no caso, o problema do

corte multisseparador mínimo também foi solucionado por um algoritmo combinatório. A principal ideia era combinar respostas de problemas solucionados em tempo polinomial para resultar numa resposta aproximada de um problema NP-difícil. Esse método não fez uso de programação matemática mas também mostrou o seu valor devido à sua garantia de aproximação.

Vários dos algoritmos apresentados fazem uso da aleatoriedade para tomar decisões. O simples fato de utilizar aleatoriedade faz com que o algoritmo precise ser analisado por meio da teoria da probabilidade. Na verdade, todos os algoritmos estudados no presente trabalho são aleatorizados, com exceção daquele apresentado na seção 4.2.1. Sendo o algoritmo aleatório, a única forma de apresentar uma garantia de aproximação é por meio do valor esperado da resposta devolvida. Dessa forma, existem soluções devolvidas que os seus valores vão além da garantia de aproximação. Mas é importante destacar que a média das respostas estão sempre na faixa do valor esperado de garantia de aproximação.

Outro ponto positivo da aleatoriedade é que ela admite o valor médio de resposta. Na análise de algoritmos determinísticos, os seus resultados sempre são avaliados com base no pior caso. Dessa forma, os algoritmos aleatorizados que possuem uma certa garantia de aproximação possuem, na verdade, o caso esperado mais próximo da resposta ótima. Caso o algoritmo seja aleatorizado, tem-se uma ideia melhor do valor da resposta "média" que será devolvida. Além disso, o valor esperado de aproximação normalmente é melhor do que o valor para o pior caso, possibilitando uma aparente melhora na garantia de aproximação.

Outra ferramenta que mostrou a sua importância no projeto de algoritmos de aproximação foi a métrica. A prova de que uma resposta retornada por um programa matemático segue as propriedades de métricas, fez com que as suas propriedades pudessem ser utilizadas tanto para provar a proximidade do algoritmo como para projetá-lo. Foram utilizados métodos que tomavam decisões com base na norma ℓ_1. Dessa forma, basta realizar uma imersão do espaço métrico finito definido por meio da solução de um programa matemático em um espaço métrico definido pela norma ℓ_1 para que uma resposta aproximada pudesse ser obtida. O estudo de imersões está sendo muito importante no

projeto de algoritmos e o algoritmo apresentado na seção 4.3 foi apenas um exemplo de aplicação.

É proposto como trabalho futuro o estudo completo da solução apresentada por Arora, Rao e Vazirani em [2] e [3]. Os autores propuseram um algoritmo de aproximação que soluciona o problema do corte mais disperso uniforme, uma versão similar do mesmo problema estudado no presente trabalho mas com capacidade unitária nas arestas. A descoberta desse algoritmo foi um grande avanço na área de análise de algoritmos porém o seu conceito é longo e abstrato, fazendo com que o seu estudo seja de difícil assimilação. Outra proposta de trabalho futuro é aplicar os algoritmos estudados nesse trabalho em experimentos e analisar os resultados obtidos. Aplicar os algoritmos em grafos específicos como grafos de mundo pequeno, grafos de *power law* ou hipercubos provê uma boa ideia de trabalho futuro pois pode-se verificar qual algoritmo possui resultados melhores para os problemas estudados e quais os motivos disso acontecer.

BIBLIOGRAFIA

[1] Sanjeev Arora, James R. Lee, e Assaf Naor. Euclidean distortion and the sparsest cut. *In Proceedings of the 37th Annual ACM Symposium on Theory of Computing*, páginas 553–562. ACM Press, 2005.

[2] Sanjeev Arora, Satish Rao, e Umesh Vazirani. Expander flows, geometric embeddings and graph partitioning. *Proceedings of the Thirty-sixth Annual ACM Symposium on Theory of Computing*, STOC '04, páginas 222–231, New York, NY, USA, 2004. ACM.

[3] Sanjeev Arora, Satish Rao, e Umesh Vazirani. Geometry, flows, and graph-partitioning algorithms. *Commun. ACM*, 51(10):96–105, outubro de 2008.

[4] Yonatan Aumann e Yuval Rabani. An O(log k) approximate min-cut max-flow theorem and approximation algorithm. *SIAM J. Comput.*, 27(1):291–301, fevereiro de 1998.

[5] Baruch Awerbuch e Yossi Azar. Buy-at-bulk network design. *Proceedings of the 38th IEEE Symposium on Foundations of Computer Science*, páginas 542–547, 1997.

[6] R Bar-Yehuda e S Even. A linear-time approximation algorithm for the weighted vertex cover problem. *Journal of Algorithms*, 2(2):198 – 203, 1981.

[7] Yair Bartal. Probabilistic approximation of metric spaces and its algorithmic applications. *In 37th Annual Symposium on Foundations of Computer Science*, páginas 184–193, 1996.

[8] J. Bourgain. On lipschitz embedding of finite metric spaces in hilbert space. *Israel Journal of Mathematics*, 52(1-2):46–52, 1985.

[9] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, e Clifford Stein. *Introduction to Algorithms (3. ed.)*. MIT Press, 2009.

[10] Gruia Călinescu, Howard Karloff, e Yuval Rabani. An improved approximation algorithm for MULTIWAY CUT. *Journal of Computer and System Sciences*, 60(3):564 – 574, 2000.

[11] E. Dahlhaus, D. S. Johnson, C. H. Papadimitriou, P. D. Seymour, e M. Yannakakis. The complexity of multiterminal cuts. *SIAM J. Comput.*, 23(4):864–894, agosto de 1994.

[12] P. Elias, A. Feinstein, e C. Shannon. A note on the maximum flow through a network. *Information Theory, IEEE Transactions on*, 2(4):117–119, dezembro de 1956.

[13] P. Erdős. Gráfok páros körüljárású részgráfjairól (On bipartite subgraphs of graphs, em húngaro). *Matematikai Lapok*, 18:283–288, 1967.

[14] Jittat Fakcharoenphol, Satish Rao, e Kunal Talwar. A tight bound on approximating arbitrary metrics by tree metrics. *J. Comput. Syst. Sci.*, 69(3):485–497, novembro de 2004.

[15] L. R. Ford, Jr. e D. R. Fulkerson. Maximal flow through a network. *Canad. J. Math.*, 8:399–404, 1956.

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

[17] Michel X. Goemans e David P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *J. ACM*, 42(6):1115–1145, novembro de 1995.

[18] Martin Grötschel, László Lovász, e Alexander Schrijver. The ellipsoid method and its consequences in combinatorial optimization. *Combinatorica*, 1(2):169–197, 1981.

[19] Johan Håstad. Some optimal inapproximability results. *J. ACM*, 48(4):798–859, julho de 2001.

[20] Dorit S. Hochbaum. Approximation algorithms for the set covering and vertex cover problems. *SIAM J. Comput.*, 11(3):555–556, 1982.

[21] Karin Hogstedt, Doug Kimelman, V T Rajan, Tova Roth, e Mark Wegman. Graph cutting algorithms for distributed applications partitioning. *SIGMETRICS Perform. Eval. Rev.*, 28(4):27–29, março de 2001.

[22] David R. Karger, Philip Klein, Cliff Stein, Mikkel Thorup, e Neal E. Young. Rounding algorithms for a geometric embedding of minimum multiway cut. *Proceedings of the Thirty-first Annual ACM Symposium on Theory of Computing*, STOC '99, páginas 668–678, New York, NY, USA, 1999. ACM.

[23] R. M. Karp. Reducibility Among Combinatorial Problems. R. E. Miller e J. W. Thatcher, editors, *Complexity of Computer Computations*, páginas 85–103. Plenum Press, 1972.

[24] Subhash Khot, Guy Kindler, Elchanan Mossel, e Ryan O'Donnell. Optimal inapproximability results for max-cut and other 2-variable csps? *SIAM J. Comput.*, 37(1):319–357, abril de 2007.

[25] Donald E. Knuth. *The art of computer programming, volume 2 (3rd ed.): seminumerical algorithms*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1997.

[26] E.L. Lima. *Espaços métricos*. Projeto Euclides. Instituto de Matemática Pura e Aplicada, CNPq, 1977.

[27] Nathan Linial, Eran London, e Yuri Rabinovich. The geometry of graphs and some of its algorithmic applications. *Combinatorica*, 15(2):215–245, 1995.

[28] Y. Nesterov e A. Nemirovskii. *Interior Point Polynomial Algorithms in Convex Programming*. SIAM studies in applied and numerical mathematics: Society for Industrial and Applied Mathematics. Society for Industrial and Applied Mathematics, 1994.

[29] James B. Orlin. Max flows in O(nm) time, or better. *Proceedings of the Forty-fifth Annual ACM Symposium on Theory of Computing*, STOC '13, páginas 765–774, New York, NY, USA, 2013. ACM.

[30] Christos H. Papadimitriou e Kenneth Steiglitz. *Combinatorial optimization: algorithms and complexity*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1982.

[31] Yuri Rabinovich e Ran Raz. Lower bounds on the distortion of embedding finite metric spaces in graphs. *Discrete & Computational Geometry*, 19(1):79–94, 1998.

[32] Harald Räcke. CMCS 39600: Theory of Metric Embeddings. Notas de aula, 2006.

[33] A. Rényi. *Probability theory*. North-Holland series in applied mathematics and mechanics. Elsevier, 1970.

[34] Sartaj Sahni e Teofilo Gonzalez. P-complete approximation problems. *J. ACM*, 23(3):555–565, julho de 1976.

[35] A. Steinbruch e P. Winterle. *Geometria Analitica*. McGraw-Hill, 1987.

[36] Vijay V. Vazirani. *Approximation algorithms*. Springer-Verlag New York, Inc., New York, NY, USA, 2001.

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