Universidade Federal do ABC
Centro de Matemática, Computação e Cognição
Relatório de Pesquisa referente ao Edital 01/2024

Problemas de cortes em grafos

Aluno:
Gabriel Ângelo Sembenelli
gabriel.sembenelli@aluno.ufabc.edu.br
Tel: (11) 91413-4646

Orientador:
Prof. Dra. Carla Negri Lintzmayer
carla.negri@ufabc.edu.br
Tel: (11) 98826-8273

Instituição de vínculo:
Universidade Federal do ABC
Tel: (11) 3356-7000
Av. dos Estados, 5001 - Bairro Bangu
Santo André, 09280-560

Instituição da bolsa:
Conselho Nacional de Desenvolvimento
Científico e Tecnológico
Processo: 139831/2024-0
Tel: (61) 3211-4000
Setor de Autarquias Sul (SAUS)
Quadra 01 lote 1 e 6
Ed. Telemundi II - Asa Sul, Brasília - DF

Resumo

Este relatório apresenta um levantamento da teoria e das técnicas de abordagem para os problemas de Corte Mínimo e Corte Máximo em grafos. O Problema do Corte Mínimo admite algoritmos exatos polinomiais, conforme explorado nas técnicas da dualidade com o Fluxo Máximo, contração de arestas e empacotamento de árvores. Em contrapartida, o Problema do Corte Máximo é NP-difícil, exigindo o uso de algoritmos de aproximação. Nesse contexto, dedica-se atenção especial ao estudo da relaxação semidefinida proposta por Goemans e Williamson. Além disso, discutem-se limites de inaproximabilidade do problema no caso geral, bem como outros resultados para famílias restritas de grafos.

1 Introdução

Problemas de cortes em grafos consistem em particionar o conjunto de vértices de um grafo, buscando minimizar ou maximizar o peso das arestas entre partes distintas, obedecendo a critérios definidos pelo problema. Eles têm grande relevância prática e teórica na Ciência da Computação, como no processamento de imagens — mais especificamente em segmentação [16, 17, 19], reconstrução [67, 91], remoção de ruídos [6] e rastreamento de objetos [75, 77, 82] —, em confiabilidade de redes [7, 50, 55], tópico relevante em telecomunicações, transporte urbano e redes de distribuição de energia, e no design de grandes circuitos eletrônicos para chips [52, 88]. Resultados envolvendo cortes também aparecem na solução de problemas teóricos, como a Conjectura 1-2-3 resolvida por Keusch [61].

Dado um grafo $G = (V, E)$, dois problemas clássicos se destacam. O Problema do Corte Mínimo (*Min-Cut*) busca a partição $\{S, T\}$ de V que minimiza o número de arestas entre S e T, enquanto o Problema do Corte Máximo (*Max-Cut*), naturalmente, visa maximizar essa quantidade. As versões desses problemas para grafos com pesos nas arestas visam otimizar o peso total das arestas, ao invés de sua quantidade.

Apesar de definições semelhantes, esses problemas apresentam grandes diferenças computacionais. Mais especificamente, o Problema do Corte Mínimo pode ser resolvido de forma eficiente, isto é, via algoritmos polinomiais. Por outro lado, o Problema do Corte Máximo é NP-difícil [38, 39, 60], ou seja, a menos que $\mathsf{P} = \mathsf{NP}$, não se espera que existam algoritmos exatos em tempo polinomial para toda instância desse problema. Assim, uma boa forma de abordá-lo é por meio de algoritmos de aproximação, que oferecem soluções eficientes subótimas cujos valores são limitados por algum fator do valor de uma solução ótima.

O presente trabalho tem como objetivo explorar o estado da arte, bem como comparar as diferentes abordagens para esses dois problemas. Além disso, busca-se complementar a formação do aluno, por meio da introdução à pesquisa científica e do contato com tópicos mais avançados da Ciência da Computação que, em geral, não são abordados durante o bacharelado, como fluxo em redes, árvores de Gomory-Hu, algoritmos aleatorizados, algoritmos de aproximação e programação semidefinida.

O restante deste relatório está estruturado da seguinte forma. A Seção 2 apresenta um resumo da pesquisa realizada. A Seção 3 define a notação e os conceitos essenciais envolvendo grafos, problemas de otimização e algoritmos de aproximação. A Seção 4 apresenta o Problema do Corte Mínimo e aborda as principais técnicas utilizadas para sua resolução, incluindo algoritmos e resultados clássicos. A Seção 5 introduz o Problema do Corte Máximo e estabelece sua complexidade. A Seção 5.1 detalha a aproximação de Goemans e Williamson para o problema do Corte Máximo, incluindo a formulação do programa semidefinido, o procedimento de arredondamento e a análise do fator de aproximação. A Seção 5.2 reúne outros resultados, que envolvem a análise da qualidade do fator de Goemans-Williamson, limites de inaproximabilidade para o Problema do Corte Máximo e desdobramentos para o problema restrito a famílias específicas de grafos. A Seção 6 traz as considerações finais sobre o trabalho desenvolvido, levando em conta os objetivos estabelecidos. Por limitações de espaço, partes do conteúdo foram abreviadas e são apresentadas em apêndice.

2 Visão geral

A primeira parte do desenvolvimento deste projeto de pesquisa focou no Problema do Corte Mínimo. O estudo se aprofundou em três técnicas fundamentais para a resolução deste problema em tempo polinomial: a forte dualidade com o Problema do Fluxo Máximo, acompanhada de algoritmos, aplicações didáticas de fluxo e resultados teóricos adicionais; a técnica de contração de arestas, com atenção especial ao algoritmo aleatorizado de Karger-Stein [58] e extensões; e o método de empacotamento de árvores, com foco no estudo do algoritmo de Karger [56].

A segunda metade da pesquisa iniciou com um estudo sobre técnicas fundamentais para o projeto de algoritmos de aproximação, que foram essenciais para construir um repertório teórico, ainda que não restritas a problemas de cortes. Nessa etapa, foram estudados os Capítulos 1, 3 e 4 do texto de Vazirani [94], bem como o Capítulo 1 e a Seção 5.1 do texto de Williamson e Shmoys [97], nos quais são apresentados problemas clássicos de otimização, que servem de base para a exemplificação de diferentes métodos de projeto e análise de algoritmos de aproximação. Entre os problemas estudados, encontram-se Cobertura por Vértices (*Vertex Cover*), Árvore de Steiner Métrica, Caixeiro Viajante Métrico, Corte *Multiway*, k-Corte Mínimo e Cobertura por Conjuntos (*Set Cover*). Quanto às técnicas de aproximação, foram exploradas relaxações de programas lineares seguidas de arredondamento determinístico ou aleatorizado, dualidade de programação linear, o método primal-dual, *dual fitting* e o uso de propriedades estruturais dos grafos, como emparelhamentos, árvores geradoras, cortes isolantes e Árvores de Gomory-Hu.

Por fim, a pesquisa se dedicou ao Problema do Corte Máximo. Partiu-se da prova de sua complexidade no caso geral e, na sequência, estudaram-se os principais algoritmos de aproximação para o problema. Em seguida, realizou-se um levantamento bibliográfico, onde foram examinados resultados sobre inaproximabilidade e outros avanços para casos mais restritos do problema.

3 Fundamentação Teórica

Notações sobre grafos que não forem definidas nesse texto estão seguindo o livro de Bondy e Murty [14]. Em um grafo $G = (V, E)$ não-direcionado, dados quaisquer subconjuntos $X, Y \subseteq V$, definimos $E[X, Y] := \{xy \in E : x \in X, y \in Y\}$, ou seja, o conjunto de todas as arestas de G que possuem uma extremidade em X e outra em Y. Quando $Y = V \setminus X$, denotamos $E[X, Y]$ simplesmente por $\partial(X)$ e chamamos esse conjunto de *corte de G associado a X*. Por conveniência, dado $v \in V$, usaremos $\partial(v)$ para abreviar $\partial(\{v\})$. Além disso, abreviamos $E[X, X]$ por $E[X]$ quando estivermos nos referindo às arestas com as duas extremidades em X.

Analogamente, para um grafo direcionado $D = (V, A)$, dados $X, Y \subseteq V$, definimos $A(X, Y) := \{xy \in A : x \in X, y \in Y\}$, ou seja, o conjunto dos arcos com cauda em X e cabeça em Y. Quando $Y = V \setminus X$, usamos $\partial^+(X) := A(X, Y)$ e $\partial^-(X) := A(Y, X)$ para denotar, respectivamente, o *corte de saída* e o *corte de entrada* de D associados a X. De modo análogo ao caso não-direcionado, é conveniente usar $\partial^\pm(v)$ como abreviação de $\partial^\pm(\{v\})$, para qualquer $v \in V$.

Um grafo é dito *conexo* se, para todo $X \subseteq V$, $X \neq \emptyset$, vale que $\partial(X) \neq \emptyset$. Analogamente, um digrafo é *fortemente conexo* se $\partial^+(X) \neq \emptyset$ para todo $X \subset V$ não-vazio.

Dado um digrafo $D = (V, A)$, vértices distintos $s, t \in V$ e uma função de pesos $w : A \to \mathbb{R}_{\geq 0}$, se tomarmos $S \subset V$ com $s \in S$ e $t \in T = V \setminus S$, então o conjunto $\partial^+(S)$ é chamado de *st-corte*.

O *valor* desse *st*-corte é definido como $\|S\| := \sum_{a \in \partial^+(S)} w(a)$. O peso total de todos os arcos do digrafo será denotado por $W_{tot} := \sum_{a \in A} w(a)$. As definições para o caso não-direcionado são análogas e usam $\partial(S)$ no lugar de $\partial^+(S)$.

Para uma função genérica $f : Y \to \mathbb{R}$, definida sobre algum conjunto Y, dado um subconjunto finito $X \subseteq Y$, faremos um pequeno abuso de notação usando $f(X)$ para abreviar $\sum_{x \in X} f(x)$.

No que se segue, as notações para algoritmos de aproximação e classes de complexidade estão usando como base os textos de Vazirani [94] e de Williamson e Shmoys [97]. Um *problema de otimização* se trata de um problema de minimização ou maximização. Cada instância I desse problema possui um conjunto de soluções viáveis, e a cada solução viável é associado um valor. É possível computar o valor de uma solução viável em tempo polinomial no tamanho $|I|$ da instância I. Dada uma instância I de um problema de maximização (minimização), uma *solução ótima* é uma solução viável que possui o maior (menor) valor. Além disso, o valor associado a uma solução ótima é chamado de *valor ótimo* e denotado por $\mathrm{OPT}(I)$. Frequentemente, abreviaremos $\mathrm{OPT}(I)$ por OPT quando o contexto permitir.

Dado um problema de maximização, um *algoritmo de aproximação* para este problema devolve, para cada instância I, uma solução viável cujo valor é limitado por algum fator do valor ótimo, em tempo polinomial em $|I|$. Em outras palavras, dada uma instância I, se $\mathcal{A}(I)$ é o valor da solução devolvida pelo algoritmo, então $\alpha \cdot \mathrm{OPT}(I) \leq \mathcal{A}(I) \leq \mathrm{OPT}(I)$, onde $\alpha \in (0, 1]$ é o *fator de aproximação*, que pode ou não depender de $|I|$. Para um problema de minimização, o que muda é que o algoritmo aproxima a solução ótima por cima, ou seja, $\mathrm{OPT}(I) \leq \mathcal{A}(I) \leq \alpha \cdot \mathrm{OPT}(I)$, onde $\alpha \geq 1$. Um algoritmo de aproximação com fator α também é chamado de *α-aproximação*.

Também será útil ter uma definição para algoritmos aleatorizados. Nesse caso, o valor da solução devolvida é uma variável aleatória. Assim, dado um problema de maximização, um *algoritmo de aproximação aleatorizado* com fator de aproximação $\alpha \in (0, 1]$ devolve uma solução de valor $\mathcal{A}(I)$, em tempo polinomial, tal que $\mathbb{P}(\mathcal{A}(I) \geq \alpha \cdot \mathrm{OPT}(I)) \geq 1/2$. A definição para problemas de minimização é análoga.

Alguns problemas de otimização admitem aproximações tão boas quanto se queira. Formalmente, dado um problema de maximização, um *esquema de aproximação de tempo polinomial* (PTAS, do inglês *polynomial-time approximation scheme*) é um algoritmo que, fixado qualquer $\varepsilon > 0$, devolve uma solução viável de valor $\mathcal{A}(I) \geq (1 - \varepsilon)\mathrm{OPT}(I)$ em tempo polinomial em $|I|$. Observe que, pela definição, o tempo de execução pode depender de ε de forma arbitrária.

4 O Problema do Corte Mínimo

A partir dos conceitos e da notação estabelecida para cortes, definimos o Problema 4.1, que busca por um *st*-corte de valor mínimo. Sua generalização, apresentada no Problema 4.2, procura um corte mínimo sem especificar os vértices s e t.

Problema 4.1 (*st*-Corte Mínimo)**.** *Dados $D = (V, A)$, $s, t \in V$ tais que $s \neq t$ e uma função $w : A \to \mathbb{R}_{\geq 0}$, determinar $S \subset V$ que resulte em um st-corte de valor mínimo. Em outras palavras,* $\min_{S \subset V,\, s \in S,\, t \notin S} \{\|S\|\}$.

Problema 4.2 (Corte Mínimo Global)**.** *Dados $D = (V, A)$ e $w : A \to \mathbb{R}_{\geq 0}$, determinar $X \subset V$ não-vazio que minimize $\sum_{a \in \partial^+(X)} w(a)$.*

Note que, em um grafo sem pesos, podemos considerar todas as arestas com pesos unitários. Dessa forma, o valor do corte mínimo global corresponde à *conectividade por arestas* do grafo. Além disso, qualquer grafo não-direcionado pode ser reduzido a um digrafo, onde cada aresta é substituída por um par de arcos opostos, cada um com o mesmo peso da aresta original. Segue-se que essa redução preserva as soluções dos Problemas 4.1 e 4.2, o que justifica a formulação para digrafos ponderados como sendo a mais geral. Nas três seções a seguir, aprofundamos as técnicas fundamentais para o estudo do Problema do Corte Mínimo, seguindo uma divisão proposta por Gawrychowski, Mozes e Weimann [40].

4.1 O Problema do *st*-Fluxo Máximo

Seja $D = (V, A)$ um digrafo e fixe $s, t \in V$ distintos. Defina uma função $c : V^2 \to \mathbb{R}_{\geq 0}$ de capacidade sobre os arcos de D, de modo que, por conveniência, atribuímos $c(uv) = 0$ sempre que $uv \notin A$. Dizemos que um *st-fluxo* é uma função $f : V^2 \to \mathbb{R}$ que satisfaz a chamada *restrição de conservação*: $\forall v \in V \setminus \{s, t\}, \quad \sum_{u \in V} f(uv) = \sum_{u \in V} f(vu)$.

De modo informal, a restrição diz que, para todo vértice, exceto possivelmente s e t, o fluxo total que entra no vértice deve igualar o que sai. Além disso, da mesma forma que a função c, adotamos $f(uv) = 0$ quando $uv \notin A$.

Um *st*-fluxo é *viável* se $0 \leq f(a) \leq c(a)$ para todo $a \in A$. No caso em que $f(a) = 0$, dizemos que f *evita* o arco a. Por outro lado, se $f(a) = c(a)$, então f *satura* o arco a.

Vamos definir o *valor* do *st*-fluxo f, denotado $\|f\|$, como a diferença entre o fluxo total que sai e o que entra em s, ou seja, $\|f\| := \sum_{u \in V} f(su) - \sum_{u \in V} f(us) \left(= \sum_{a \in \partial^+(s)} f(a) - \sum_{a \in \partial^-(s)} f(a) \right)$.

Problema 4.3 (*st*-Fluxo Máximo)**.** *Dados $D = (V, A)$, $s, t \in V$ tais que $s \neq t$ e uma função $c : V^2 \to \mathbb{R}_{\geq 0}$, determinar um st-fluxo viável de maior valor em D.*

Comecemos a elucidar a relação entre os Problemas 4.1 e 4.3. Mais especificamente, o Lema 4.4 estabelece a dualidade fraca entre o *st*-Corte Mínimo e o *st*-Fluxo Máximo. Sua demonstração foi adaptada de Erickson [29]. Dada a função $c : V^2 \to \mathbb{R}_{\geq 0}$, lembre-se de que o valor de um *st*-corte $\partial^+(S)$ é $\|S\| := \sum_{a \in \partial^+(S)} c(a)$.

Lema 4.4. *Para qualquer st-fluxo viável f e qualquer st-corte $\partial^+(S)$, vale que $\|f\| \leq \|S\|$. Além disso, $\|f\| = \|S\|$ se e somente se f satura todo arco de S a $V \setminus S$ e evita todo arco de $V \setminus S$ a S.*

Demonstração. Segue direto das definições:

$$
\begin{aligned}
\|f\| &= \sum_{u \in V} f(su) - \sum_{u \in V} f(us) && \text{definição de } \|f\| \\
&= \sum_{v \in S} \Big(\sum_{u \in V} f(vu) - \sum_{u \in V} f(uv) \Big) && \text{restrição de conservação em } S \\
&= \sum_{v \in S} \sum_{u \in V} f(vu) - \sum_{v \in S} \sum_{u \in V} f(uv) && \text{linearidade da soma} \\
&= \sum_{v \in S} \sum_{u \in V \setminus S} f(vu) - \sum_{v \in S} \sum_{u \in V \setminus S} f(uv) && \text{cancelando os arcos dentro de } S \\
&= \sum_{a \in \partial^+(S)} f(a) - \sum_{a \in \partial^-(S)} f(a) && \text{definições de } \partial^{\pm}(S) \\
&\leq \sum_{a \in \partial^+(S)} f(a) \leq \sum_{a \in \partial^+(S)} c(a) && 0 \leq f(a) \leq c(a) \text{ sempre, pois } f \text{ é viável} \\
&= \|S\| && \text{definição de } \|S\|.
\end{aligned}
$$

Além disso, a primeira desigualdade é justa se, e somente se, $f(a) = 0$ para todo $a \in \partial^-(S)$, ou seja, f evita todo arco de $V \setminus S$ a S. Similarmente, a segunda desigualdade é justa se, e somente se, $f(a) = c(a)$ para todo $a \in \partial^+(S)$, ou seja, f satura todo arco de S a $V \setminus S$. □

Corolário 4.5. *Se $\|f\| = \|S\|$ para algum st-fluxo f e algum st-corte $\partial^+(S)$, então f é um st-fluxo máximo e $\partial^+(S)$ é um st-corte mínimo.*

O Teorema 4.6 enuncia a dualidade forte entre o *st*-Corte Mínimo e o *st*-Fluxo Máximo, estabelecida em 1956 por Ford e Fulkerson [35].

Teorema 4.6 (Ford e Fulkerson 1956)**.** *Para todo digrafo com vértices s, t especificados e uma função de capacidade c, os valores de um st-fluxo máximo e de um st-corte mínimo são iguais.*

Uma demonstração didática pode ser encontrada no texto de Erickson [29]. Aqui, nosso foco será discutir o algoritmo que se segue imediatamente dela, também usando o texto de Erickson como referência principal.

4.1.1 O Algoritmo de Ford-Fulkerson

A ideia intuitiva do algoritmo é buscar, repetidamente, caminhos de s até t que ainda não estejam saturados, para transferir mais fluxo, até que isso não seja mais possível. Além disso, ao longo da execução, pode ser necessário redirecionar parte do fluxo já transferido, para que mais caminhos válidos apareçam. Essa ideia motiva a definição que vem a seguir.

Entretanto, antes de apresentá-la, seria conveniente que o digrafo tivesse, no máximo, um arco entre cada par de vértices, então fazemos uma redução. Para cada arco uw, adicione um novo vértice v e substitua o arco uw pelos arcos uv e vw, cada um com a mesma capacidade de uw. Observe que a solução ótima é preservada após essa redução. Além disso, o digrafo original não será mais necessário a partir daqui, de modo que usaremos $D = (V, A)$ para denotar o digrafo transformado.

A função $c_f : V^2 \to \mathbb{R}$, denominada *função de capacidade residual*, é definida como

$$c_f(uv) := \begin{cases} c(uv) - f(uv) & \text{se } uv \in A \\ f(vu) & \text{se } vu \in A \\ 0 & \text{caso contrário.} \end{cases}$$

Por exemplo, se um arco uv tem capacidade $c(uv) = 20$ e fluxo $f(uv) = 8$, então as capacidades residuais são $c_f(uv) = 12$ e $c_f(vu) = 8$. De modo informal, elas indicam a quantidade de fluxo que ainda pode ser transferida ao longo de uv e a quantidade de fluxo que pode ser revertida (ou cancelada) ao longo de uv, respectivamente.

O *digrafo residual*, denotado por $D_f = (V, A_f)$, é construído de modo a ter os mesmos vértices de D, porém seus arcos são apenas os pares de vértices com capacidade residual positiva, ou seja, $A_f := \{uv \in V^2 : c_f(uv) > 0\}$. Um *caminho aumentador* no digrafo residual é uma sequência de vértices distintos, digamos $P = (v_0, v_1, \ldots, v_k)$, tal que $v_0 = s$, $v_k = t$ e $v_{i-1}v_i \in A_f$ para todo $i \in \{1, \ldots, k\}$. Representamos o conjunto de arcos no caminho por $A(P)$, de modo que o *comprimento* de P é dado por $|A(P)| = k$.

Note que, dado um caminho aumentador P em D_f, então $\varphi := \min_{uv \in A(P)}\{c_f(uv)\}$ é a quantidade máxima de fluxo que pode ser transferida ao longo de P mantendo viabilidade. Assim, podemos obter um novo fluxo $f' : V^2 \to \mathbb{R}$ em D da seguinte forma:

$$f'(uv) = \begin{cases} f(uv) + \varphi & \text{se } uv \in A(P) \\ f(uv) - \varphi & \text{se } vu \in A(P) \\ f(uv) & \text{caso contrário.} \end{cases}$$

Observe que o valor do fluxo aumentou por φ, isto é, $\|f'\| = \|f\| + \varphi$. Então, em linhas gerais, o Algoritmo 1 repete o processo de buscar por caminhos aumentadores em D_f e aumentar o fluxo ao longo deles. O procedimento termina quando tais caminhos não existem mais.

Algoritmo 1 Algoritmo de Ford-Fulkerson para o st-Fluxo Máximo

1: **Função** FordFulkerson(D, s, t, c)
2: Seja $f : V^2 \to \mathbb{R}$ uma função identicamente nula
3: Seja $c_f : V^2 \to \mathbb{R}$ tal que $c_f = c$
4: Seja $D_f := (V, \{uv \in V^2 : c_f(uv) > 0\})$
5: **Enquanto** existe caminho aumentador P em D_f **faça**
6: $\varphi \leftarrow \min_{uv \in A(P)}\{c_f(uv)\}$
7: Atualize f ao longo de P com incremento φ
8: Atualize D_f e c_f
9: **Devolva** $\|f\|$

Além disso, caso seja necessário obter $S \subset V$ correspondente ao corte mínimo, basta lembrar que, ao final, f satura todo arco de S a $V \setminus S$ e evita todo arco de $V \setminus S$ a S. Logo, S pode ser obtido ao identificar todos os vértices alcançáveis a partir de s na versão final do digrafo residual.

Agora suponha, por simplicidade, que as capacidades iniciais do digrafo D são inteiros não-negativos. Seja m a quantidade de arcos nesse grafo e F o valor do fluxo máximo. Nesse caso, a cada iteração, o fluxo aumenta por uma quantidade inteira, estritamente positiva, de modo que o algoritmo realiza no máximo F iterações. Além disso, como em cada iteração o algoritmo reconstrói o digrafo residual e busca por um caminho aumentador — operações que custam $O(m)$ com as sub-rotinas de busca usuais —, o tempo total do algoritmo é $O(mF)$.

Vale notar que, da forma como está, sem especificar o tipo de busca para o caminho aumentador, o tempo de $O(mF)$ é justo, como pode ser visto na Figura 1. Além disso, se permitirmos capacidades reais não-negativas, é possível construir instâncias com capacidades irracionais nas quais o algoritmo pode nunca parar e sequer convergir ao valor correto [21, 29, 71, 99].

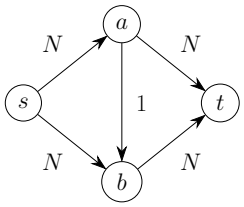

Figura 1: Exemplo de pior caso para o Algoritmo 1, onde N é um inteiro positivo. Os números representam a capacidade sobre cada arco. Se o algoritmo encontrar apenas os caminhos aumentadores (s, a, b, t) e (s, b, a, t) alternadamente, então $\varphi = 1$ em toda iteração, ou seja, o valor do fluxo aumenta de unidade em unidade até atingir $F = 2N$.

4.1.2 O Algoritmo de Edmonds-Karp

Em linhas gerais, este algoritmo pode ser entendido como uma versão do algoritmo de Ford-Fulkerson que usa Busca em Largura (BFS, do inglês *Breadth-First Search*), como sub-rotina para procurar caminhos aumentadores. Essa escolha traz consigo a vantagem de que o tempo total do algoritmo se torna polinomial no tamanho do grafo. Isso é o que verificamos a seguir.

Dado um digrafo residual D_f, seja $d_f(u, v)$ a *distância* entre os vértices u e v nesse grafo, *i.e.*, o comprimento do menor caminho existente entre u e v, desconsiderando as funções c e f sobre os arcos. Além disso, vamos usar $d_f(v)$ para abreviar $d_f(s, v)$.

O Lema 4.7 estabelece a monotonicidade de $d_f(v)$, que é uma propriedade importante para analisar o consumo de tempo do algoritmo. Sua demonstração foi adaptada de Cormen, Leiserson, Rivest e Stein [21].

Lema 4.7. *Para todo $v \in V \setminus \{s, t\}$, temos que $d_f(v)$ não diminui a cada iteração do algoritmo de Edmonds-Karp.*

Demonstração. Suponha, por absurdo, que uma iteração do algoritmo diminui a distância de s a v, para algum $v \in V \setminus \{s, t\}$. Ou seja, se D_f é o digrafo residual antes da iteração e $D_{f'}$ é o digrafo residual após a iteração, então estamos supondo que $d_f(v) > d_{f'}(v)$. Entre todos os vértices com essa propriedade, escolha v para ser aquele com o menor valor de $d_{f'}(v)$.

Seja $P' = (s, \ldots, u, v)$ um caminho mínimo entre s e v em $D_{f'}$. Logo, vale que (i) $d_{f'}(v) = d_{f'}(u) + 1$ (por construção de P') e (ii) $d_f(u) \leq d_{f'}(u)$ (pois, caso contrário, v não teria sido escolhido corretamente).

Além disso, $uv \notin A_f$, pois caso contrário, teríamos

$$\begin{aligned} d_f(v) &\leq d_f(u) + d_f(u, v) && \text{(desigualdade triangular)} \\ &= d_f(u) + 1 && \text{(se } uv \in A_f\text{)} \\ &\leq d_{f'}(u) + 1 = d_{f'}(v) && \text{(por (ii) e (i))} \end{aligned}$$

contradizendo a suposição inicial.

Até aqui, temos que $uv \notin A_f$, mas $uv \in A_{f'}$, implicando que uma quantidade positiva de fluxo foi transferida ao longo de vu. Ou seja, se P foi o caminho aumentador encontrado em D_f, então $vu \in A(P)$. Como P foi encontrado a partir de uma BFS, então P é um caminho mínimo de s a t e, como tal, seus subcaminhos também são mínimos. Logo, $d_f(v) = d_f(u) - 1 \leq d_{f'}(u) - 1 = d_{f'}(v) - 2 < d_{f'}(v)$, contradizendo a suposição inicial. Logo, um vértice como v não pode existir. □

O Lema 4.7 implica o Teorema 4.8, que estabelece o número de iterações do algoritmo. Sua demonstração também foi adaptada de Cormen *et al.* [21]. A partir disso, o consumo de tempo total do algoritmo é obtido imediatamente como o Corolário 4.9.

Teorema 4.8. *O total de iterações realizadas pelo algoritmo de Edmonds-Karp em um digrafo com n vértices e m arcos é $O(nm)$.*

Demonstração. Sejam $u, v \in V$ vértices conectados por um arco em A (pode ser uv ou vu).

Novamente, como o algoritmo usa BFS como sub-rotina de busca, os caminhos aumentadores serão caminhos mínimos, de modo que, se uv participa de um caminho aumentador em D_f, então $d_f(v) = d_f(u) + 1$.

Além disso, se uv é *crítico*, isto é, $c_f(uv) = \min_{a \in A(P)}\{c_f(a)\}$, então uv não estará presente nos próximos digrafos residuais, a menos que o fluxo de u a v seja diminuído, o que acontece quando vu participa de um caminho aumentador. Seja $D_{f'}$ o digrafo residual onde isso ocorre, então teremos $d_{f'}(u) = d_{f'}(v) + 1$. Como $d_f(v) \leq d_{f'}(v)$ pelo Lema 4.7, então temos $d_f(u) = d_f(v) - 1 \leq d_{f'}(v) - 1 = d_{f'}(u) - 2 < d_{f'}(u)$. Ou seja, a cada vez que uv puder se tornar crítico novamente, a distância de s a u no digrafo residual terá aumentado estritamente. Como qualquer caminho de s a u tem comprimento menor que n, então o arco uv pode se tornar crítico no máximo n vezes.

Para concluir, observe que qualquer caminho aumentador, por ser um conjunto finito, tem ao menos um arco crítico. Além disso, o digrafo residual tem, no máximo, $2m$ arcos. Logo, o algoritmo deve parar após $O(nm)$ iterações. □

Corolário 4.9. *O tempo de execução do algoritmo de Edmonds-Karp é* $O(nm^2)$.

Demonstração. A cada iteração, é necessário realizar um número constante de buscas em largura para construir o digrafo residual e buscar um caminho aumentador. Logo, cada iteração custa $O(m)$. Usando o Teorema 4.8, o resultado segue. □

Vale observar que, ao longo das demonstrações, foi utilizada a versão reduzida do digrafo que, embora conveniente para definir a função de capacidade residual, pode conter muito mais vértices que o digrafo original. Entretanto, numa implementação prática[1] essa redução não é necessária, de modo que o tempo do algoritmo permanece $O(nm^2)$ para um digrafo não reduzido com n vértices e m arcos.

4.1.3 O Algoritmo de Dinitz

Este algoritmo foi proposto com a intenção de melhorar o tempo de execução do algoritmo de Ford-Fulkerson. Ou seja, suas bases ainda residem na ideia de buscar caminhos aumentadores, mas sua motivação está em fazer isso de modo mais rápido, usando uma estrutura de dados mais eficiente. Vale ressaltar que a versão brevemente descrita a seguir é, na verdade, uma reinterpretação feita por Shimon Even e Alon Itai, a qual se tornou mais popular no Ocidente e, reconhecidamente, a mais elegante. Mais detalhes sobre a história e o desenvolvimento do algoritmo podem ser encontrados nos textos de Dinitz [26] e Even [30].

A primeira observação é que, em uma implementação prática do Algoritmo 1, não é necessário reconstruir todo o digrafo residual D_f a cada iteração, pois, ao aumentar o fluxo ao longo de um caminho aumentador P, apenas os arcos em P são alterados. Todos os outros arcos em D_f continuam intactos. Isso é vantajoso, pois como P possui, no máximo, n vértices e $n - 1$ arcos, atualizar f, c_f e D_f ao longo de P custa $O(n)$, enquanto construir D_f do zero custaria $O(m)$, que é $O(n^2)$ no pior caso.

[1] Exemplo de implementação para o Algoritmo 1 (Edmonds-Karp) disponível em: github.com/Gabriel-Sembenelli/cp/blob/master/graphs/MaxFlow-EdmondsKarp.cpp.

Assim, Dinitz percebeu que a operação mais custosa em uma iteração é a busca por um caminho aumentador em D_f. Com as sub-rotinas usuais (BFS ou DFS), essa busca custa $O(m)$. Após isso, as atualizações ao longo de P custam $O(n)$, como já mencionado. Ou seja, o gargalo de fato é a procura por P.

Observando propriedades da árvore de BFS, Dinitz notou que essa estrutura poderia ser melhorada para encontrar múltiplos caminhos aumentadores de uma vez, dando origem ao que ele chamou de *BFS estendida* [26]. Lembre-se de que uma BFS a partir de s pode ser usada para computar a distância de s até um outro vértice v, denotada por $d(s, v)$, que é a menor quantidade de arcos em um caminho de s a v. Porém, uma BFS usual gera uma arborescência T na qual, para qualquer vértice v diferente de s, existe apenas um arco uv em T com $d(s, u) = d(s, v) - 1$, *i.e.*, todo vértice, exceto a raiz, tem exatamente um pai.

Dessa forma, se usarmos a BFS clássica para buscar um caminho aumentador em D_f, e supondo que ainda não estamos na iteração final, então teremos uma arborescência T contendo t onde s é a raiz. Após aumentar o fluxo ao longo do único caminho de s a t em T, pelo menos um arco será saturado e deixará de existir em T, desconectando s de t. Ou seja, a busca serviu para encontrar apenas um caminho aumentador — isto é exatamente o que acontece no algoritmo de Edmonds-Karp.

No entanto, para um vértice $v \neq s$, poderiam existir outros arcos uv em D_f tais que $d(s, u) = d(s, v) - 1$ e que não foram incluídos em T. Se incluirmos todos esses arcos para cada v, então a nova estrutura pode conter muitos outros caminhos de s a t (e deixará de ser uma arborescência, mas isso não é um problema). Uma modificação simples na sub-rotina de BFS é suficiente para incluir tais arcos, surgindo assim a *BFS estendida*.

Note que a estrutura gerada pela BFS estendida contém somente arcos uv tais que $d(s, u) = d(s, v) - 1$. Logo, aplicar uma DFS nessa estrutura para buscar um st-caminho é eficiente, pois a cada passo avançamos para vértices estritamente mais distantes de s. Em outras palavras, essa sub-rotina define uma estrutura de níveis (ou camadas) no digrafo, e a DFS sobre essa estrutura só percorre arestas que aumentam o nível. Um exemplo comparando a BFS clássica e a estendida pode ser visto na Figura 2.

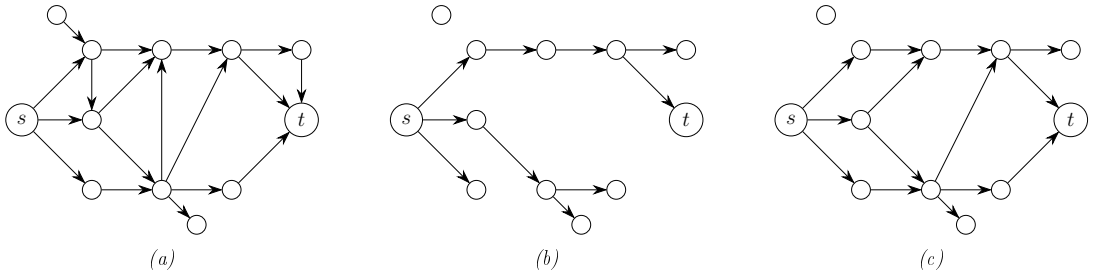

Figura 2: (a) Um digrafo qualquer, com vértices s e t distintos especificados. (b) Estrutura gerada pela BFS usual a partir de s. (c) Estrutura gerada pela BFS estendida a partir de s.

É necessário, no entanto, ter atenção aos possíveis caminhos sem saída, *i.e.*, que não chegam até t. Se tal caminho for encontrado, seus arcos devem ser eliminados no momento de retorno da recursão da DFS, para que não sejam atravessados novamente em futuras iterações. Além disso, arcos saturados após aumentos de fluxo também devem ser excluídos da estrutura. Essas observações são facilmente implementáveis.

Agora é possível estimar o tempo de uma *fase* do algoritmo, *i.e.*, o tempo durante o qual uma

estrutura gerada pela BFS estendida é útil. Fazemos isso contando as operações entre eventos em que arcos são removidos. A cada DFS — seja a inicial ou após a exclusão de um arco —, em $O(n)$ passos temos a ocorrência do próximo evento, pois a DFS alcança a ponta de um caminho sem saída, onde arcos são eliminados no retorno da recursão, ou alcança t, definindo um caminho aumentador que garantidamente satura e exclui algum arco. Como a estrutura tem $O(m)$ arcos e cada arco, após eliminado, não é reinserido, então cada fase executa em tempo $O(nm)$.

Além disso, temos a garantia de que os caminhos aumentadores da próxima fase, se existirem, terão comprimento estritamente maior. Uma demonstração detalhada dessa afirmação pode ser encontrada no texto de Even [30].

Finalmente, como um st-caminho possui no máximo $n-1$ arcos, então não ocorrem mais que $O(n)$ fases. Consequentemente, o tempo total do algoritmo de Dinitz é $O(n^2m)$.

O sucesso do algoritmo de Dinitz se deve a diversos fatores. Um dos mais evidentes é sua elegância ao combinar BFS e DFS para resolver o problema em questão — uma característica da versão desenvolvida por Shimon Even e Alon Itai [26]. Além disso, sua simplicidade de implementação[2] e boa performance prática fazem com que este seja um dos algoritmos de fluxo prediletos em maratonas de programação.

Outras aplicações do st-Fluxo Máximo foram estudadas e encontram-se detalhadas no Apêndice A.0.1. Uma discussão sobre o Problema do Fluxo Máximo Global e outros avanços encontra-se no Apêndice A.0.2.

4.2 Contração de Arestas

Para um grafo $G = (V, E)$ não-direcionado, *contrair* $uv \in E$ consiste em adicionar um novo vértice x e, para cada aresta do tipo uw ou vw, onde $w \in V \setminus \{u, v\}$, adicionar uma aresta xw. Por fim, removem-se os vértices u e v, bem como todas as arestas incidentes a eles. O grafo G com a aresta uv contraída será denotado por G/uv.

Note que permitimos a existência de arestas paralelas, *i.e.*, múltiplas arestas entre um mesmo par de vértices, tanto antes quanto depois de uma contração. Contudo, não permitimos *loops*, *i.e.*, arestas com extremidades idênticas. Grafos com arestas paralelas e, possivelmente, *loops* são chamados de *multigrafos*. Em um multigrafo $G = (V, E)$, temos que E é um multiconjunto, ou seja, pode conter elementos repetidos.

Observe que contrair arestas que não pertencem ao corte mínimo não impacta na solução do Problema do Corte Mínimo. Em outras palavras, dado $S \subset V$ tal que $\partial(S)$ é o corte mínimo desejado, contrair uma aresta e com extremidades ambas em S ou ambas em $V \setminus S$ dá origem a um novo grafo com a mesma solução. De fato, o valor do corte mínimo não pode aumentar, pois $\partial(S)$ está intacto, logo, ainda é um corte em G/e. Também não pode diminuir, pois todo corte em G/e tem um correspondente em G, com as extremidades de e do mesmo lado da partição.

Dessa forma, é possível contrair todas as arestas internas a S e a $V \setminus S$, até restarem dois *metavértices* e as arestas entre eles. Em linhas gerais, essa é a base de muitos algoritmos que aplicam a técnica de contração de arestas.

[2]Exemplo de implementação para o Algoritmo de Dinitz disponível em: github.com/Gabriel-Sembenelli/cp/blob/master/graphs/MaxFlow-Dinitz.cpp.

Em 1992, Nagamochi e Ibaraki [78, 79] introduziram um algoritmo determinístico que, dado um multigrafo não-direcionado, sem pesos nas arestas, com n vértices e m arestas, encontra o corte mínimo em tempo $O(m + \min\{cn^2, pn + n^2 \log n\})$, onde c é o valor do corte mínimo e $p\,(\leq m)$ é a quantidade de pares de vértices conectados por pelo menos uma aresta. Além disso, a versão desse algoritmo estendida para grafos não-direcionados e ponderados nas arestas resolve o problema em tempo $O(nm + n^2 \log n)$.

No ano seguinte, Karger e Stein [54, 57, 58] propuseram uma abordagem aleatorizada para a contração de arestas, inicialmente baseada em ideias e algoritmos simples. A versão final do artigo, publicada em 1996, trouxe resultados relevantes, que são discutidos nas Seções 4.2.1 e 4.2.2. Já em 1997, Stoer e Wagner [90] aplicaram contração de arestas de uma forma um pouco diferente, apresentando um algoritmo determinístico surpreendentemente simples que encontra o corte mínimo em tempo $O(nm + n^2 \log n)$, considerando um grafo não-direcionado ponderado nas arestas com n vértices e m arestas.

4.2.1 O Algoritmo de Karger-Stein

Considere um multigrafo G não-direcionado com n vértices, m arestas e, a princípio, sem pesos nas arestas. Embora este caso pareça simples, o algoritmo proposto por Karger e Stein [54, 57, 58] oferece resultados interessantes e foi posteriormente estendido para casos mais gerais.

Para este primeiro caso, o algoritmo em questão simplesmente contrai uma aresta escolhida aleatoriamente e com probabilidade uniforme, entre aquelas que ainda existem no grafo, até que restem apenas dois vértices. O Algoritmo 2 resume essa ideia.

Algoritmo 2 Algoritmo Probabilístico de Karger-Stein para o Corte Mínimo

```
1: Função KargerStein(G)
2:     Enquanto G tem mais do que 2 vértices faça
3:         sorteie e ∈ E com probabilidade uniforme
4:         G ← G/e
5:     Devolva G
```

A análise da probabilidade de sucesso desse algoritmo decorre do Teorema 4.10, cuja demonstração foi adaptada de Mitzenmacher e Upfal [76].

Teorema 4.10. *O Algoritmo 2 devolve um corte mínimo específico de G com probabilidade pelo menos $\binom{n}{2}^{-1} \in \Omega(n^{-2})$.*

Demonstração. Observe que podem existir vários cortes em G com o mesmo valor. Fixemos um corte mínimo específico $C \subseteq E$, onde $c := |C|$. Para cada iteração do algoritmo, contando a partir de 1, denote por $G_i = (V_i, E_i)$ o grafo obtido após a i-ésima iteração. Definimos os seguintes eventos:

- A_i : a aresta contraída na i-ésima iteração não está em C;
- B_i : nenhuma aresta de C foi contraída nas primeiras i iterações.

Por definição, $B_i = \bigcap_{j=1}^{i} A_j$. Além disso, como o algoritmo itera até que sobrem dois vértices, o objetivo é calcular $\mathbb{P}(B_{n-2})$.

Note que todo vértice deve ter ao menos c vizinhos, caso contrário, seria possível isolar um

vértice removendo menos do que c arestas, contrariando a minimalidade de C. Assim, denotando $\delta(G) := \min_{v \in V}\{|\partial(v)|\}$, temos $\delta(G) \geq c$. Consequentemente, pelo Lema do Aperto de Mãos, $2m = \sum_{v \in V} |\partial(v)| \geq \sum_{v \in V} \delta(G) \geq \sum_{v \in V} c = nc$, de onde $m \geq \frac{nc}{2}$.

Como o algoritmo sorteia arestas uniformemente, a probabilidade de contrair uma aresta em C na primeira iteração (isto é, a probabilidade de acontecer o complemento de A_1) será $\mathbb{P}(\overline{A_1}) = \frac{c}{m} \leq \frac{c}{nc/2} = \frac{2}{n}$. Logo, $\mathbb{P}(B_1) = \mathbb{P}(A_1) = 1 - \mathbb{P}(\overline{A_1}) \geq 1 - 2/n = (n-2)/n$.

Supondo que B_1 ocorreu, ou seja, que a primeira iteração não contraiu uma aresta em C, então o grafo G_1 tem $n-1$ vértices e corte mínimo com o mesmo valor c. Como antes, $\delta(G_1) \geq c$, o que implica $2|E_1| \geq |V_1|c = (n-1)c$. Portanto $\mathbb{P}(\overline{A_2} \mid B_1) = \frac{c}{|E_1|} \leq \frac{2}{n-1}$, de onde $\mathbb{P}(A_2 \mid B_1) \geq 1 - \frac{2}{n-1} = \frac{(n-1)-2}{n-1}$.

De modo geral, para todo inteiro $1 < i \leq n-2$, temos $\mathbb{P}(A_i \mid B_{i-1}) \geq 1 - \frac{2}{n-i+1} = \frac{n-i-1}{n-i+1}$.

Como $\mathbb{P}(B_i) = \mathbb{P}(B_{i-1} \cap A_i)$, obtemos a recorrência $\mathbb{P}(B_i) = \mathbb{P}(A_i \mid B_{i-1}) \cdot \mathbb{P}(B_{i-1})$, com base $\mathbb{P}(B_1) \geq (n-2)/n$. Expandindo a partir de B_{n-2}, temos:

$$\begin{aligned}
\mathbb{P}(B_{n-2}) &= \mathbb{P}(A_{n-2} \mid B_{n-3}) \cdot \mathbb{P}(B_{n-3}) \\
&= \mathbb{P}(A_{n-2} \mid B_{n-3}) \cdot \mathbb{P}(A_{n-3} \mid B_{n-4}) \cdot \mathbb{P}(B_{n-4}) \\
&= \ldots = \mathbb{P}(B_1) \cdot \prod_{i=2}^{n-2} \mathbb{P}(A_i \mid B_{i-1}) \\
&\geq \frac{n-2}{n} \cdot \frac{n-3}{n-1} \cdot \frac{n-4}{n-2} \cdots \frac{3}{5} \cdot \frac{2}{4} \cdot \frac{1}{3} = \frac{2}{n(n-1)},
\end{aligned}$$

o que completa a prova. □

O teorema estabelece uma cota inferior para a probabilidade de encontrar um corte mínimo específico, o que significa que a probabilidade de sucesso, *i.e.*, de obter qualquer corte mínimo, é pelo menos $\binom{n}{2}^{-1}$. Como essa probabilidade é pequena, podemos executar o algoritmo muitas vezes a fim de aumentá-la. Mais especificamente, sabendo que a desigualdade $1+x \leq e^x$ vale para todo $x \in \mathbb{R}$, se repetirmos o algoritmo $n(n-1)\ln n$ vezes e devolvermos o menor corte obtido, então a probabilidade de erro será, no máximo, $\left(1 - \frac{2}{(n-1)n}\right)^{n(n-1)\ln n} \leq \left(e^{\frac{-2}{(n-1)n}}\right)^{n(n-1)\ln n} = n^{-2}$. Esse valor vai a zero conforme n cresce, o que significa que um corte mínimo será encontrado com alta probabilidade.

Karger e Stein [58] mostram como o Algoritmo 2 pode ser implementado de modo a consumir tempo polinomial. Mais especificamente, utilizando uma matriz de adjacência e um vetor com os valores de $|\partial(v)|$ para cada $v \in V$, é possível simular em $O(n)$ a contração de uma aresta já sorteada. Também é mostrado como obter um algoritmo de tempo $O(n)$ para sortear uma aresta. Apesar de ser uma análise necessária para provar o tempo total do algoritmo, essa sub-rotina dificilmente seria implementada na prática, pois os geradores de números aleatórios e as ferramentas de arredondamento, integrados às linguagens de programação ou ao próprio sistema, geralmente são suficientes. Dessa forma, o Algoritmo 2 executa em tempo $O(n^2)$. Logo, em $n(n-1)\ln n$ tentativas, encontramos um corte mínimo em tempo total $O(n^4 \log n)$ com alta probabilidade.

4.2.2 Karger-Stein: Melhorias e Extensões

Embora $O(n^4 \log n)$ seja polinomial, não é um tempo de execução competitivo quando comparado aos algoritmos baseados em fluxo disponíveis à época. Dessa forma, Karger e Stein [58] apresentam uma versão aprimorada, baseada em recursão, que utiliza o Algoritmo 2 como subrotina. Após as melhorias, dado um grafo não-direcionado, ponderado arbitrariamente, com n vértices e m arestas, o algoritmo encontra todos os cortes mínimos em tempo $O(n^2 \log^3 n)$ e espaço $O(m \log(n^2/m))$.

Para compreender os resultados seguintes, definimos a classe NC (do inglês, *Nick's Class*), que contém os problemas de decisão resolvíveis deterministicamente em tempo polilogarítmico, usando uma quantidade polinomial de processadores. Mais precisamente, existem constantes positivas k e c tais que, para uma entrada de tamanho n, o algoritmo executa em tempo $O(\log^k n)$ utilizando $O(n^c)$ processadores. A classe RNC é a versão randomizada de NC.

No mesmo artigo, Karger e Stein [58] também mostram que o algoritmo pode ser paralelizado, executando em tempo polilogarítmico com n^2 processadores. Esse foi o primeiro resultado que provou que o Problema do Corte Mínimo está em RNC.

Além disso, foram dados resultados em complexidade e estruturas de dados sobre cortes próximos ao mínimo, *i.e.*, de valor no máximo α vezes o valor do corte mínimo, e sobre o Problema do Corte *Multiway* Mínimo, que consiste em determinar um corte de peso mínimo que desconecta um conjunto de k terminais entre si.

4.3 Empacotamento de Árvores

Uma *arborescência* é a versão direcionada de uma árvore enraizada, ou seja, é um grafo direcionado acíclico no qual um vértice r, chamado de raiz, tem grau de entrada nulo e todos os outros vértices têm grau de entrada igual a um. Uma arborescência com raiz r pode ser chamada de *r-arborescência*. Um *empacotamento de árvores* em um grafo não-direcionado é um conjunto de árvores geradoras arestas-disjuntas. A definição de *empacotamento de arborescências* para o caso direcionado é análoga. O *valor* do empacotamento é a quantidade de árvores (ou arborescências) no conjunto.

Em um digrafo sem pesos, Gabow [36, 37] explora uma espécie de dualidade entre empacotamentos de r-arborescências e r-cortes, onde r-corte é um conjunto $\partial^+(S)$ com $S \neq \emptyset$ sendo um subconjunto próprio de vértices e $r \in S$. Essa dualidade está expressa no Teorema 4.11.

Teorema 4.11 (Edmonds 1972)**.** *Em um grafo direcionado, o valor máximo de um empacotamento de r-arborescências é igual ao valor mínimo de um r-corte.*

Identificando que o empacotamento de r-arborescências satisfaz uma estrutura de matroides, Gabow desenvolve um algoritmo para construir um empacotamento máximo e, consequentemente, computar a conectividade por arestas, λ, em tempo $O(\lambda m \log(n^2/m))$ para digrafos e $O(m + \lambda^2 n \log(n/\lambda))$ para grafos não-direcionados.

Em 2000, Karger [56] abordou o caso não-direcionado de uma forma ligeiramente diferente. Nessa situação, não vale exatamente o teorema de Edmonds, mas uma versão similar, dada pelo Teorema 4.12, de Nash-Williams [80].

Teorema 4.12 (Nash-Williams 1961). *Todo grafo não-direcionado com corte mínimo c tem um empacotamento de árvores com valor pelo menos $c/2$.*

Note que, para cada corte e cada árvore geradora, pelo menos uma aresta do grafo deve ser compartilhada entre essas duas estruturas. Assim, considerando um corte mínimo qualquer, se um empacotamento máximo possui pelo menos $c/2$ árvores geradoras, então a média de arestas do corte por árvore é, no máximo, 2. Logo, existe ao menos uma árvore no empacotamento que compartilha no máximo duas arestas com o corte.

Dada uma árvore geradora T, é útil saber quais arestas ela compartilha com o corte mínimo, pois essas arestas determinam univocamente a partição induzida pelo corte. De fato, dados vértices u e v, eles estarão do mesmo lado do corte se, e somente se, o caminho entre eles ao longo de T tem um número par de arestas compartilhadas com o corte. Assim, é possível percorrer a árvore e determinar, facilmente, o lado de cada vértice na partição.

Daqui em diante, vamos utilizar uma nomenclatura conveniente. Seja T uma árvore geradora de G. Um corte de G *k-respeita* T se ele compartilha no máximo k arestas com T. Da mesma forma, dizemos que T *k-restringe* o corte.

Agora estendemos a noção de empacotamento para grafos não-direcionados com pesos nas arestas. Um *empacotamento de árvores ponderado* é um conjunto de árvores geradoras, onde cada uma possui um valor associado e, para cada aresta do grafo, a soma dos valores das árvores que a contêm não excede o peso da aresta. O *valor* do empacotamento é a soma dos valores das árvores no conjunto.

Observe que, nesse contexto, não exigimos mais que as árvores sejam arestas-disjuntas. A restrição é sobre o valor total atribuído às árvores que passam por cada aresta do grafo. Assim, o Lema 4.13 e o Corolário 4.14 se aplicam para um grafo não-direcionado, ponderado nas arestas e com corte mínimo de valor c, onde as frações consideradas estão se referindo ao peso total do empacotamento. Observe que o Corolário 4.14 segue do Lema 4.13, utilizando $\alpha = 1$ e o Teorema 4.12, que garante $\beta \geq 1/2$.

Lema 4.13 (Karger 2000). *Dado qualquer empacotamento de árvores ponderado com valor βc e qualquer corte de valor αc, uma fração de pelo menos $(3-\alpha/\beta)/2$ do peso das árvores 2-restringe o corte.*

Corolário 4.14 (Karger 2000). *Em qualquer empacotamento ponderado máximo, pelo menos metade do peso das árvores 2-restringe o corte mínimo.*

Como encontrar empacotamentos máximos é custoso, Karger [56] utiliza um dos resultados de Gabow [36, 37] para apenas computar um empacotamento de valor $c/2$. No entanto, o algoritmo tem tempo dependente de c, então emprega-se também uma técnica de amostragem aleatória para obter um novo grafo, chamado *esqueleto*, que é mais esparso, mas que preserva propriedades estruturais importantes do grafo original. Assim, obtém-se o Teorema 4.15.

Teorema 4.15 (Karger 2000). *Dado qualquer grafo ponderado não-direcionado com n vértices e m arestas, em tempo $O(m + n\log^3 n)$ é possível construir um conjunto de $O(\log n)$ árvores geradoras tal que o corte mínimo 2-respeita $1/3$ delas, com alta probabilidade.*

Sorteando uma árvore desse conjunto com probabilidade proporcional ao seu valor, a probabilidade de que ela 2-restringe o corte mínimo é constante. Portanto, ao sortear $\Theta(\log n)$ árvores

dessa forma, obtém-se, com alta probabilidade, pelo menos uma que 2-restringe o corte mínimo. Essa observação conduz ao Lema 4.16.

Lema 4.16 (Karger 2000). *Dado um grafo com n vértices e m arestas, se o menor corte que 2-respeita uma dada árvore geradora pode ser encontrado em tempo $T(n,m)$, então o corte mínimo global do grafo pode ser computado em tempo $T(n,m)+O(m+n\log^3 n)$ com probabilidade constante, e em tempo $O(T(n,m)\log n+m+n\log^3 n)$ com alta probabilidade.*

Karger [56] apresenta cuidadosamente uma série de algoritmos complementares para computar o menor corte que 2-respeita uma árvore, utilizando programação dinâmica. Primeiro, resolve o problema para cortes que 1-respeitam um caminho, depois para cortes que 1-respeitam uma árvore, e por fim para cortes que compartilham exatamente 2 arestas com a árvore.

O algoritmo resultante encontra o menor corte que 2-respeita a árvore em tempo $O(n^2)$. Aplicando-o às $\Theta(\log n)$ árvores construídas, pelo Lema 4.16, obtém-se um algoritmo que encontra, com alta probabilidade, todos os cortes mínimos globais em tempo $O(n^2\log n)$.

Além disso, o autor desenvolveu outra série de algoritmos, utilizando estruturas de *dynamic trees*, para encontrar o menor corte que 2-respeita a árvore em tempo $O(m\log^2 n)$. Quando aplicado ao Lema 4.16 no lugar de $T(n,m)$, conclui-se que o corte mínimo global pode ser encontrado, com alta probabilidade, em tempo $O(m\log^3 n)$.

O mesmo artigo ainda inclui pequenas melhorias para esses algoritmos, resultados sobre paralelização, um limite assintótico para o número de cortes próximos ao mínimo, bem como uma estrutura de dados conveniente para representar tais cortes.

5 O Problema do Corte Máximo

Com as definições de cortes em mãos e motivados por aplicações práticas, é natural nos perguntarmos qual subconjunto de vértices de um grafo define um corte de valor máximo. A formalização dessa pergunta está expressa no Problema 5.1.

Problema 5.1 (Corte Máximo). *Dado um grafo $G=(V,E)$ e uma função $w: E\to\mathbb{R}_{\geq 0}$, determinar um subconjunto $S\subset V$ não-vazio que resulte em um corte de valor máximo. Em outras palavras, determinar* $\max_{\emptyset\subset S\subset V}\{\|S\|\}$.

A versão não-ponderada desse problema (toda aresta com peso 1) também aparece na literatura como Problema do Corte Máximo Simples (*Simple Max-Cut*) ou Problema do Corte Máximo em Cardinalidade (*Cardinality Max-Cut*, ou *Maximum Cardinality Cut*).

Na versão de decisão, dados um grafo, uma função de pesos nas arestas e um valor real k, a questão é decidir se existe um corte de valor maior ou igual a k. Esse problema é NP-completo, conforme estabelecido por Karp [60], por meio de uma redução partindo de um problema sabidamente NP-completo para o Problema do Corte Máximo. Mais especificamente, a redução foi a partir do Problema da Partição.

Problema 5.2 (Partição). *Dados $c_1,\ldots,c_n\in\mathbb{Z}$, decidir se existe $S\subseteq\{1,\ldots,n\}$ tal que $\sum_{i\in S}c_i=\sum_{j\notin S}c_j$.*

Vale ressaltar que, originalmente, Karp [60] provou a redução considerando o Problema do Corte Máximo com pesos inteiros quaisquer, ou seja, incluindo negativos. Dessa forma, esse resultado não necessariamente implica que o Problema 5.1 é NP-completo. Mesmo assim, vamos detalhar a redução original, sem modificações, apenas para ilustrar a técnica de demonstração. Essa decisão se justifica, pois o resultado que mostraremos na sequência implica imediatamente que o Problema 5.1 é NP-completo.

Teorema 5.3 (Karp 1972)**.** *O Problema do Corte Máximo com pesos em $\mathbb{Z}$ é NP-completo.*

Demonstração. Considere o Problema do Corte Máximo com pesos inteiros. O primeiro passo para estabelecer que este problema é NP-completo é verificar que ele está em NP. De fato, dada uma instância do problema e um $k \in \mathbb{Z}$, um certificado é um corte válido com valor maior ou igual a k, se tal corte existir. O segundo passo é consiste em verificar que este problema é NP-difícil, ou seja, que todos os problemas em NP se reduzem a este. Uma forma de fazer isso é por meio de uma redução a partir de outro problema sabidamente NP-completo, neste caso, o Problema da Partição. Dada uma instância $c_1, \ldots, c_n \in \mathbb{Z}$ do Problema 5.2, seja $G := (V, E)$, onde $V := \{1, \ldots, n\}$, $E := \{ij : i, j \in V, i \neq j\}$, seja $w : E \to \mathbb{Z}$ tal que $w(ij) := c_i c_j$ e tome $k := \left\lceil \frac{1}{4} \left(\sum_{i \in V} c_i\right)^2 \right\rceil$. Note que essa construção é polinomial no tamanho da instância original.

Por fim, temos que verificar que a resposta para a instância do Problema da Partição é `SIM` se, e somente se, a resposta para a instância correspondente para o Problema do Corte Máximo é `SIM`. Suponha primeiro que existe $S \subseteq V$ favorável para a instância do Problema da Partição, ou seja, tal que $\sum_{i \in S} c_i = \sum_{j \notin S} c_j$. Como $\sum_{i \in S} c_i + \sum_{j \notin S} c_j = \sum_{i \in V} c_i$, temos as igualdades $\sum_{i \in S} c_i = \sum_{j \notin S} c_j = \frac{1}{2} \sum_{i \in V} c_i$, donde

$$
\begin{aligned}
\|S\| = \sum_{i \in S} \sum_{j \notin S} w(ij) = \sum_{i \in S} \sum_{j \notin S} c_i c_j = \sum_{i \in S} \left(c_i \cdot \sum_{j \notin S} c_j \right) && c_i \text{ em evidência} \\
= \left(\sum_{i \in S} c_i \right) \cdot \left(\sum_{j \notin S} c_j \right) && \sum_{j \notin S} c_j \text{ em evidência} \\
= \left(\frac{1}{2} \sum_{i \in V} c_i \right) \cdot \left(\frac{1}{2} \sum_{i \in V} c_i \right) = \frac{1}{4} \left(\sum_{i \in V} c_i \right)^2 = \left\lceil \frac{1}{4} \left(\sum_{i \in V} c_i \right)^2 \right\rceil .
\end{aligned}
$$

A última igualdade vem do fato que a quantidade $\frac{1}{4} \left(\sum_{i \in V} c_i\right)^2$ é inteira, pois é igual à soma inicial de inteiros $\sum_{i \in S} \sum_{j \notin S} c_i c_j$.

Suponha agora que não existe $S \subseteq V$ viável para a instância do Problema da Partição. Vamos provar que qualquer $S \subseteq V$ tem valor menor do que $\left\lceil \frac{1}{4} \left(\sum_{i \in V} c_i\right)^2 \right\rceil$. De fato, dado $S \subseteq V$ qualquer, defina $s := \sum_{i \in S} c_i$ e $t := \sum_{i \in V} c_i$, de modo que $\sum_{j \notin S} c_j = t - s$. Por hipótese, temos $s \neq t - s$, donde $s \neq \frac{t}{2}$. Agora note que a função $f : \mathbb{R} \to \mathbb{R}$, definida por $f(x) = x \cdot (t - x)$, é uma parábola côncava com raízes em 0 e t. Logo, o único máximo de f acontece quando $x = t/2$. Desse modo, $\|S\| = \sum_{i \in S} \sum_{j \notin S} w(ij) = \left(\sum_{i \in S} c_i\right) \cdot \left(\sum_{j \notin S} c_j\right) = s \cdot (t - s)$
$= f(s) < f\left(\frac{t}{2}\right) = \frac{t^2}{4} \leq \left\lceil \frac{t^2}{4} \right\rceil = \left\lceil \frac{1}{4} \left(\sum_{i \in V} c_i\right)^2 \right\rceil$. □

Além disso, em 1976, Garey, Johnson e Stockmeyer [39] provaram que a versão sem pesos (equivalentemente, com pesos unitários) do Problema do Corte Máximo também é NP-completa.

A demonstração encadeia duas reduções, partindo do Problema da Satisfatibilidade com até 3 literais por cláusula para o Problema da Satisfatibilidade Máxima com até 2 literais por cláusula, e deste para a versão não-ponderada do Problema do Corte Máximo. Como essa versão é um caso particular da versão ponderada com pesos reais não-negativos, segue-se que, a menos que $\mathsf{P} = \mathsf{NP}$, não existem algoritmos exatos em tempo polinomial para toda instância do Problema 5.1. Consequentemente, uma das abordagens ao problema é por meio de algoritmos de aproximação.

Na literatura, diversas técnicas têm sido estudadas para o Problema do Corte Máximo. Muitas delas garantem fator de aproximação 1/2 e algumas dessas são apresentadas no Apêndice B. Na seção a seguir, apresentamos o primeiro algoritmo que melhorou esse fator de aproximação.

5.1 O Algoritmo de Goemans-Williamson

A escrita desta seção foi baseada nos textos de Goemans e Williamson [41], Vazirani [94] e de Williamson e Shmoys [97].

Em um artigo de 1995, Goemans e Williamson [41] propuseram uma abordagem baseada em programação semidefinida, combinada com arredondamento por hiperplano aleatório, para o Problema do Corte Máximo. O trabalho desses autores marcou a introdução de uma nova técnica de aproximação, bem como a primeira melhoria significativa além do fator 1/2 em quase duas décadas, garantindo fator de aproximação $\alpha_{GW} \approx 0.878$.

O ponto de partida é um *programa quadrático (quadratic program)*, ou seja, um problema definido sobre variáveis inteiras, sujeitas a restrições quadráticas, que visa otimizar uma função objetivo quadrática nessas variáveis. Mais especificamente, dado um grafo $G = (V, E)$ com n vértices e uma função de pesos $w : E \to \mathbb{R}_{\geq 0}$, escreva o conjunto de vértices como $V := \{v_1, v_2, \ldots, v_n\}$ e defina, para quaisquer $i, j \in \{1, \ldots, n\}$, o valor w_{ij} como sendo o próprio peso $w(v_i v_j)$, caso a aresta $v_i v_j$ exista, e zero caso contrário.

Dado $S \subseteq V$ qualquer, defina, para cada $v_i \in V$, a variável $y_i \in \{-1, 1\}$, de modo que $y_i = 1$ quando $v_i \in S$ e $y_i = -1$ caso contrário. Assim, dados $v_i, v_j \in V$, o produto das variáveis y_i e y_j vale $y_i y_j = 1$ quando os vértices correspondentes pertencem ao mesmo lado do corte e $y_i y_j = -1$ quando em lados opostos. Além disso, note que uma restrição do tipo $y_i \in \{-1, 1\}$ pode ser reescrita como o par de restrições $y_i^2 = 1$, $y_i \in \mathbb{Z}$. Com essa intuição, o Problema do Corte Máximo pode ser escrito como o seguinte programa quadrático:

$$\begin{aligned} \text{maximizar} \quad & \frac{1}{2} \sum_{1 \leq i < j \leq n} w_{ij}(1 - y_i y_j) & & (1) \\ \text{sujeito a} \quad & y_i^2 = 1, & \forall\, v_i \in V, \\ & y_i \in \mathbb{Z}, & \forall\, v_i \in V. \end{aligned}$$

Usando a discussão anterior, note que cada corte $\partial(S)$ no grafo corresponde a uma solução viável deste programa com valor equivalente, e vice-versa. Assim, a solução ótima deste programa, denotada por Z_Q^*, vale exatamente OPT.

Vamos obter uma relaxação que pode ser resolvida em tempo polinomial e que dá origem a um algoritmo de aproximação. O primeiro passo é relaxar o Programa Quadrático 1 para um *programa vetorial (vector program)*, *i.e.*, um problema sobre n vetores $\vec{x}_1, \vec{x}_2, ..., \vec{x}_n$ definidos

em $\mathbb{R}^n$, que consiste em otimizar uma função linear sobre os produtos internos desses vetores, sujeito a restrições lineares sobre esses produtos internos.

No nosso caso, cada variável $y_i \in \mathbb{Z}$ será substituída por um vetor $\vec{x}_i \in \mathbb{R}^n$ e cada produto $y_i y_j$ em $\mathbb{Z}$ será substituído pelo produto interno canônico $\vec{x}_i \cdot \vec{x}_j$ de vetores em $\mathbb{R}^n$, de modo a obter o seguinte programa vetorial:

$$\begin{aligned} \text{maximizar} \quad & \frac{1}{2} \sum_{1 \leq i < j \leq n} w_{ij}(1 - \vec{x}_i \cdot \vec{x}_j) & & (2) \\ \text{sujeito a} \quad & \vec{x}_i \cdot \vec{x}_i = 1, & \forall\, v_i \in V, \\ & \vec{x}_i \in \mathbb{R}^n, & \forall\, v_i \in V. \end{aligned}$$

Note que as variáveis deste programa podem ser interpretadas geometricamente como vetores sobre uma hiperesfera em $\mathbb{R}^n$ de raio unitário, centrada na origem. Seja Z_V^* o valor ótimo deste programa e observe que cada solução do Programa Quadrático 1 corresponde a uma solução de mesmo valor no Programa Vetorial 2. Basta associar, para cada variável $y_i \in \mathbb{Z}$, um vetor da forma $(y_i, 0, \ldots, 0) \in \mathbb{R}^n$. Ou seja, o Programa 2 é de fato uma relaxação do Programa 1, de modo que vale a desigualdade $Z_V^* \geq Z_Q^*$.

Para verificar que o Programa 2 pode ser resolvido em tempo polinomial, vamos verificar que ele é equivalente a um outro programa, que pertence a uma classe de programas sabidamente solucionáveis em tempo polinomial. Mas, antes disso, precisamos estabelecer algumas notações e propriedades importantes.

Primeiro, vamos enxergar vetores como matrizes-coluna, ou seja, um vetor $\vec{x} \in \mathbb{R}^n$ será visto como uma matriz de dimensão $n \times 1$ com entradas reais. Além disso, vamos aliviar a notação e usar apenas x no lugar de $\vec{x}$ quando o contexto permitir. Assim, dados vetores $\vec{x}, \vec{y} \in \mathbb{R}^n$, o produto interno canônico $\vec{x} \cdot \vec{y}$ pode ser escrito como $x^\top y$.

Seja $A \in \mathbb{R}^{n \times n}$ uma matriz $n \times n$ tal que a (i,j)-ésima entrada é a_{ij} Nesse caso, escrevemos $A = (a_{ij})$ e dizemos que a matriz A é *simétrica* quando $a_{ij} = a_{ji}$ para quaisquer $i, j \in \{1, \ldots, n\}$, ou seja, quando sua transposta $A^\top$ satisfaz $A^\top = A$. Quando uma matriz $A \in \mathbb{R}^{n \times n}$ é simétrica, são equivalentes [94]:

- $\forall x \in \mathbb{R}^n, x^\top A x \geq 0$;
- Todos os autovalores de A são não-negativos;
- Existe uma matriz $B \in \mathbb{R}^{n \times n}$ tal que $B^\top B = A$.

Nesse caso, dizemos que a matriz A é *positiva semidefinida* e escrevemos $A \succeq 0$.

Em um *programa semidefinido (semidefinite program)*, as variáveis em consideração são as entradas y_{ij} de uma matriz $Y = (y_{ij}) \in \mathbb{R}^{n \times n}$, sujeitas a restrições lineares, além da restrição para Y ser é simétrica e positiva semidefinida. A função objetivo a ser otimizada é uma função linear nas entradas y_{ij}.

Vamos reescrever o Programa Vetorial 2 como um programa semidefinido. Para cada escolha de $i, j \in \{1, \ldots, n\}$, trocamos o produto interno $\vec{x}_i \cdot \vec{x}_j$ pela variável $y_{ij} \in \mathbb{R}$ e impomos que a matriz subjacente $(y_{ij}) \in \mathbb{R}^{n \times n}$ seja simétrica e positiva semidefinida. Dessa forma, obtemos o

programa semidefinido:

$$\begin{aligned} \text{maximizar} \quad & \frac{1}{2} \sum_{1 \leq i < j \leq n} w_{ij}(1 - y_{ij}) && (3) \\ \text{sujeito a} \quad & y_{ii} = 1, && \forall\, v_i \in V, \\ & (y_{ij}) \in \mathbb{R}^{n \times n} && \text{simétrica e positiva semidefinida.} \end{aligned}$$

No que se segue, vamos estabelecer a equivalência entre o Programa Vetorial 2 e o Programa Semidefinido 3. A vantagem de fazer isso é que podemos falar de um ou outro de forma indiferente, além de usar resultados estabelecidos para programas semidefinidos, que são bem estudados. A prova de equivalência foi adaptada do texto de Vazirani [94].

Lema 5.4. *O Programa Vetorial 2 e o Programa Semidefinido 3 são equivalentes.*

Demonstração. Vamos mostrar que cada solução viável do Programa 2 corresponde a uma solução viável com mesmo valor para o Programa 3 e vice-versa.

Por um lado, dada uma solução viável $x_1, \ldots, x_n \in \mathbb{R}^n$ do Programa 2, defina a variável $y_{ij} := x_i^\top x_j$ para cada escolha de índices $i, j \in \{1, \ldots, n\}$. É imediato que o valor da nova solução no Programa 3 é o mesmo. Além disso, $y_{ii} = x_i^\top x_i = 1$ para cada $i \in \{1, \ldots, n\}$ e, pela simetria do produto interno, vale $y_{ij} = y_{ji}$ para quaisquer $i, j \in \{1, \ldots, n\}$. Para ver que a matriz subjacente $(y_{ij}) \in \mathbb{R}^{n \times n}$ é positiva semidefinida, basta notar que ela é o resultado de $X^\top X$, onde $X \in \mathbb{R}^{n \times n}$ é a matriz com colunas $x_1, \ldots, x_n$.

Por outro lado, dada uma solução viável $Y = (y_{ij}) \in \mathbb{R}^{n \times n}$ do Programa 3, então esta matriz é positiva semidefinida, de modo que existe uma matriz $X \in \mathbb{R}^{n \times n}$ tal que $X^\top X = Y$. Assim, para cada $i \in \{1, \ldots, n\}$, seja $x_i \in \mathbb{R}^n$ a i-ésima coluna da matriz X. Consequentemente, $y_{ij} = x_i^\top x_j$ para quaisquer $i, j \in \{1, \ldots, n\}$ e o valor da solução $x_1, \ldots, x_n$ no Programa 2 é o mesmo. Finalmente, $x_i^\top x_i = y_{ii} = 1$ para cada $i \in \{1, \ldots, n\}$. □

Computacionalmente, nem sempre é possível resolver um programa semidefinido de forma exata, pois o valor ótimo pode ser irracional. Todavia, dado qualquer $\varepsilon > 0$, é possível encontrar uma solução viável para o Programa Semidefinido 3 cujo valor é pelo menos $Z^*_{SD} - \varepsilon$, e isso pode ser feito em tempo polinomial em n e em $\log(1/\varepsilon)$ [41, 94, 97]. De forma similar, dada uma matriz positiva semidefinida $Y \in \mathbb{R}^{n \times n}$, nem sempre é possível computar a decomposição exata dessa matriz como $Y = X^\top X$, pois a matriz X pode conter entradas irracionais. Porém, é possível, em tempo polinomial, computar uma aproximação suficientemente próxima [41, 94].

Segue daí que podemos resolver o Programa Semidefinido 3 e, consequentemente, o Programa Vetorial 2 em tempo polinomial, de modo que os pequenos erros provenientes das aproximações podem ser tratados no cálculo do fator de aproximação [41, 94, 97].

Dado um conjunto-solução de vetores $x_1, \ldots, x_n \in \mathbb{R}^n$ para o Programa Vetorial 2, o próximo passo é obter uma solução viável para o Programa Quadrático 1. Goemans e Williamson realizam essa etapa de forma aleatorizada, sorteando-se um vetor $r \in \mathbb{R}^n$ uniformemente distribuído na hiperesfera unitária e definindo-se o conjunto $S := \{v_i \in V : x_i \cdot r \geq 0\}$. A interpretação geométrica é que r define um hiperplano passando pela origem, dividindo o espaço $\mathbb{R}^n$ em duas partes, e os vetores $x_i \in \mathbb{R}^n$ ficam de um lado ou de outro conforme o sinal do produto interno $x_i \cdot r$.

Para obter o vetor r computacionalmente, pode-se sortear independentemente cada coordenada $r_1, \ldots, r_n \in \mathbb{R}$ a partir da distribuição normal de média zero e desvio padrão unitário. Esse processo pode ser feito em tempo polinomial e resulta, após normalização, no vetor $r \in \mathbb{R}^n$ com as propriedades desejadas [94]. Dessa forma, o algoritmo de Goemans-Williamson pode ser resumido, em pseudocódigo, como o Algoritmo 3.

Algoritmo 3 Algoritmo de aproximação de Goemans-Williamson para o Corte Máximo

1: **Função** GoemansWilliamson(G, w)
2: Construa e resolva o Programa Vetorial 2, obtendo $x_1, \ldots, x_n \in \mathbb{R}^n$
3: Sorteie um vetor unitário $r \in \mathbb{R}^n$ de maneira uniformemente distribuída
4: **Devolva** $S := \{v_i \in V : x_i \cdot r \geq 0\}$

Já foi argumentado que o algoritmo executa em tempo polinomial. Resta verificar que ele garante algum fator de aproximação. Definimos

$$\alpha_{GW} := \min_{0 \leq \theta \leq \pi} \frac{2}{\pi} \frac{\theta}{1 - \cos\theta}. \tag{4}$$

Utilizando ferramentas de cálculo em uma variável e minimizando sobre a variável θ, obtemos o mínimo global quando $\theta \approx 2.331122$, donde $\alpha_{GW} > 0.87856$ [41].

Quando não houver ambiguidade, abreviaremos α_{GW} por α. Assim, o objetivo daqui em diante é mostrar que o Algoritmo 3 é uma α-aproximação para o Problema do Corte Máximo. Na verdade, considerando os erros decorrentes da solução aproximada do Programa Semidefinido 3 e da decomposição aproximada da matriz resultante Y como $X^\top X$, obtemos uma $(\alpha - \varepsilon)$-aproximação, para qualquer $\varepsilon > 0$ fixo. Porém, para simplificar a exposição, desconsideraremos tais erros, pois não afetam o argumento.

Sejam $x_1^*, \ldots, x_n^* \in \mathbb{R}^n$ os vetores unitários obtidos na solução ótima do Programa Vetorial 2. Dados $i, j \in \{1, \ldots, n\}$, vamos usar $\theta_{ij} \in [0, \pi]$ para denotar o ângulo entre os vetores x_i^* e x_j^*. Como $x_i^* \cdot x_j^* = \|x_i^*\| \|x_j^*\| \cos\theta_{ij}$ e $\|x_i^*\| = \|x_j^*\| = 1$, podemos escrever o valor ótimo da função objetivo do Programa 2 como

$$Z_V^* := \frac{1}{2} \sum_{1 \leq i < j \leq n} w_{ij}(1 - \cos\theta_{ij}). \tag{5}$$

O fator de aproximação será obtido a partir da comparação entre o valor esperado do corte devolvido pelo Algoritmo 3, $\mathbb{E}[W]$, e o valor ótimo do Programa Vetorial 2, Z_V^*. Para isso, precisamos de alguns lemas. Começamos calculando a probabilidade de dois vértices quaisquer ficarem em partes distintas do corte.

Lema 5.5. *Considerando o Algoritmo 3, para quaisquer $i, j \in \{1, \ldots, n\}$, seja X_{ij} o evento em que os vértices v_i e v_j terminam em partes distintas do corte. Então $\mathbb{P}(X_{ij}) = \frac{\theta_{ij}}{\pi}$.*

Demonstração. Dados $i, j \in \{1, \ldots, n\}$, considere o vetor $r \in \mathbb{R}^n$ sorteado pelo algoritmo e seja r' sua projeção ortogonal no plano gerado pelos vetores x_i^* e x_j^*. Dessa forma, o vetor r pode ser escrito como a soma de r' com uma segunda componente que é ortogonal a esse plano, donde $x_i^* \cdot r = x_i^* \cdot r'$ e, analogamente, $x_j^* \cdot r = x_j^* \cdot r'$. Assim, o fato de v_i e v_j caírem em partes distintas depende apenas do ângulo de r' nesse plano. Além disso, pela forma como r é sorteado, segue-se

que a projeção r' tem um ângulo aleatório uniformemente distribuído no plano gerado por x_i^* e x_j^*. A Figura 3 indica as regiões de interesse, *i.e.*, os dois setores nos quais o evento X_{ij} ocorre caso r' esteja dentro de algum deles. Cada região tem medida de probabilidade $\theta_{ij}/(2\pi)$, donde $\mathbb{P}(X_{ij}) = 2\theta_{ij}/(2\pi)$ e o resultado segue. □

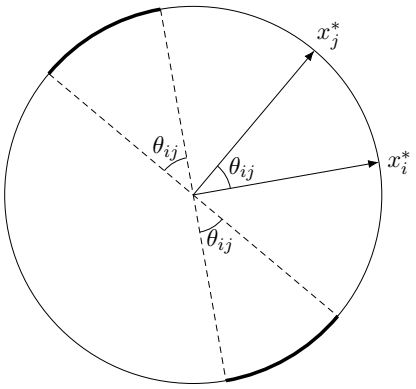

Figura 3: Figura para a demonstração do Lema 5.5. No plano gerado pelos vetores unitários x_i^* e x_j^*, as retas tracejadas representam as direções ortogonais a esses vetores, que possuem um ângulo de θ_{ij} entre si. A projeção r' corresponde a um ângulo sorteado uniformemente nesse plano. Os arcos em negrito delimitam dois setores nos quais $x_i^* \cdot r'$ e $x_j^* \cdot r'$ têm sinais opostos, fazendo com que ocorra X_{ij}. Cada setor está associado a uma medida de probabilidade $\theta_{ij}/(2\pi)$.

O Lema 5.6 estabelece uma relação entre a expressão θ_{ij}/π, obtida no Lema 5.5, e a expressão $(1 - \cos\theta_{ij})/2$, que aparece na Equação 5.

Lema 5.6. *Para α definido pela Equação 4, dado qualquer $\theta \in [0, \pi]$, vale $\frac{\theta}{\pi} \geq \frac{\alpha}{2}(1 - \cos\theta)$.*

Demonstração. Se $\theta = 0$, então o resultado é imediato. Além disso, dado $0 < \theta \leq \pi$, então $-1 \leq \cos\theta < 1$, ou seja, $1 - \cos\theta > 0$, de modo que é válido escrever

$$\begin{aligned}\frac{\theta}{\pi} &= \frac{\theta}{\pi} \cdot \frac{2}{2} \cdot \frac{1-\cos\theta}{1-\cos\theta} = \left(\frac{2}{\pi}\frac{\theta}{1-\cos\theta}\right)\frac{1}{2}(1-\cos\theta) \\ &\geq \left(\min_{0\leq\varphi\leq\pi}\frac{2}{\pi}\frac{\varphi}{1-\cos\varphi}\right)\frac{1}{2}(1-\cos\theta) = \frac{\alpha}{2}(1-\cos\theta).\end{aligned}$$

□

A partir dos Lemas 5.5 e 5.6, obtém-se o resultado principal para o fator de aproximação, expresso no Teorema 5.7. Para finalizar, usamos os resultados $Z_V^* \geq Z_Q^* = \mathrm{OPT}$.

Teorema 5.7. *Sejam α e Z_V^* definidos, respectivamente, pelas equações 4 e 5. Se W representa o valor do corte devolvido pelo Algoritmo 3, então $\mathbb{E}[W] \geq \alpha\, Z_V^*$.*

Demonstração. Seja X_{ij} o evento em que v_i e v_j terminam em partes distintas do corte. Pela linearidade da esperança, tem-se

$$\begin{aligned}\mathbb{E}[W] = \mathbb{E}\left[\sum_{1\leq i<j\leq n} w_{ij}X_{ij}\right] &= \sum_{1\leq i<j\leq n} w_{ij}\mathbb{P}(X_{ij}) = \sum_{1\leq i<j\leq n} w_{ij}\frac{\theta_{ij}}{\pi} && \text{[Lema 5.5]} \\ \geq \sum_{1\leq i<j\leq n} w_{ij}\frac{\alpha}{2}(1-\cos\theta_{ij}) &= \alpha\, Z_V^*. && \text{[Lema 5.6]}\end{aligned}$$

□

Segue daí que $\mathbb{E}[W] \geq \alpha\text{OPT}$. Contudo, essa desigualdade, por si só, não implica que o algoritmo aleatorizado é uma α-aproximação sob a definição que estamos usando. A alternativa de amplificar a probabilidade de sucesso via repetições foi motivada no artigo de Goemans e Williamson [41] e detalhada no texto de Vazirani [94]. Mais especificamente, fixa-se $\varepsilon > 0$ e calcula-se uma constante inteira k, dependente apenas de α e ε. Essa constante é escolhida de modo a garantir que o processo de sortear k hiperplanos e escolher o melhor deles resulte em um corte de valor W' satisfazendo a desigualdade $\mathbb{P}(W' \geq (1-\varepsilon)\mathbb{E}[W]) > 1/2$.

Outra abordagem é desaleatorizar o Algoritmo 3. Esse processo foi realizado com sucesso por Mahajan e Ramesh [74], que corrigiram a primeira tentativa de desaleatorização de Goemans e Williamson. Esse procedimento utiliza o clássico método das esperanças condicionais, embora, neste caso, os cálculos sejam mais técnicos. O resultado é um algoritmo determinístico, polinomial e com o mesmo fator de aproximação α_{GW}.

Por fim, no artigo de Goemans e Williamson [41], além da análise para a versão clássica do Problema do Corte Máximo, os autores apresentam resultados adicionais: demonstram um fator de aproximação melhor que α_{GW} quando o valor ótimo Z_V^* do Programa Vetorial 2 é uma fração suficientemente grande do peso total W_{tot}; estendem a técnica para instâncias em que as arestas podem ter pesos negativos; e propõem uma formulação alternativa do problema como um programa de minimização de autovalores.

5.2 Outros resultados

Há diversos resultados adicionais sobre o Problema do Corte Máximo, além daqueles que foram apresentados até aqui. Todavia, até o momento da redação deste relatório, não se conhece algoritmo de aproximação com fator superior a α_{GW} (definido na Equação 4) para o caso geral do problema. Os resultados explorados ao longo da pesquisa podem ser divididos entre vários tipos: análises que avaliam a qualidade do fator de aproximação α_{GW}; limites de inaproximabilidade para o problema; estudos em classes particulares de grafos; e alguns resultados espectrais, ou seja, envolvendo autovalores.

5.2.1 Qualidade do fator de Goemans-Williamson

Resultados deste tipo tentam refinar a análise feita por Goemans e Williamson, ou tentam obter um fator melhor por meio de alterações no Programa Vetorial 2. Em geral, as alterações consistem em adicionar restrições que são naturalmente satisfeitas por qualquer corte, com a intenção de melhorar o fator de aproximação sem perder soluções inteiras viáveis (cortes válidos).

Dada uma instância I para o Problema do Corte Máximo, definimos o valor esperado $\mathbb{E}[W(I)]$ do corte devolvido pelo Algoritmo 3; o valor $\mathcal{A}(I)$ do corte devolvido pela versão desaleatorizada deste algoritmo; o valor $\text{OPT}(I)$ do corte máximo; e o valor $Z_{SD}^*(I)$ ótimo do Programa Semidefinido 3 associado. Dessa forma, temos que $\mathbb{E}[W(I)] \leq \mathcal{A}(I) \leq \text{OPT}(I) \leq Z_{SD}^*(I)$. Defina, ainda, os seguintes fatores:

$$\alpha_1 := \inf_I \frac{\mathbb{E}[W(I)]}{Z_{SD}^*(I)}, \qquad \alpha_2 := \inf_I \frac{\mathbb{E}[W(I)]}{\text{OPT}(I)}, \qquad \alpha_3 := \inf_I \frac{\mathcal{A}(I)}{\text{OPT}(I)}, \qquad \gamma := \inf_I \frac{\text{OPT}(I)}{Z_{SD}^*(I)}.$$

Podemos enxergar α_2 como o fator de aproximação do Algoritmo 3, α_3 como o fator de aproxima-

ção da versão desaleatorizada deste algoritmo e γ como o *gap* de integralidade entre o Programa Quadrático 1 e o Programa Semidefinido 3. Dessa forma, temos imediatamente as desigualdades $\alpha_1 \leq \alpha_2 \leq \alpha_3$, e $\alpha_1 \leq \gamma$.

Formalmente, Goemans e Williamson [41] mostraram que $\alpha_{GW} \leq \alpha_1$. Em 1999, Karloff [59] mostrou que $\alpha_2 \leq \alpha_{GW}$, donde se segue a igualdade $\alpha_{GW} = \alpha_1 = \alpha_2$. Além disso, no mesmo artigo, Karloff mostra que a adição de qualquer família de restrições que são lineares em relação aos produtos internos do Programa Vetorial 2 e válidas para todo corte não altera α_2.

Em 2002, Feige e Schechtman [32] mostram que $\gamma \leq \alpha_{GW}$ e que $\alpha_3 \leq \alpha_{GW}$, de modo que vale a igualdade entre todos esses fatores, $\alpha_{GW} = \alpha_1 = \alpha_2 = \alpha_3 = \gamma$. Além disso, esses autores consideraram a adição de um tipo específico de restrições ao Programa Vetorial 2, que são válidas para todo corte, obtendo uma outra relaxação, com *gap* de integralidade não maior que 0.891.

5.2.2 Inaproximabilidade

Em 1998, Arora, Lund, Motwani, Sudan e Szegedy [5] mostram que o Problema do Corte Máximo não admite PTAS. Em outras palavras, existe uma constante $\varepsilon > 0$ tal que é NP-difícil aproximar o Problema do Corte Máximo com um fator $1 - \varepsilon$. O resultado é existencial, ou seja, o valor de ε não é explicitado.

Dado um problema de maximização, vamos chamar uma constante c no intervalo aberto $(0, 1)$ de *fator de inaproximabilidade* se, para cada $\varepsilon > 0$, é NP-difícil aproximar o problema com um fator de $c + \varepsilon$. Bellare, Goldreich e Sudan [9] estabeleceram um fator de inaproximabilidade de 71/72 para o Problema do Corte Máximo. Trevisan, Sorkin, Sudan e Williamson [93] mostraram um fator de inaproximabilidade de 60/61 e subsequentemente, aliado ao trabalho de Håstad [51], foi possível alcançar um fator de inaproximabilidade de 16/17.

Em 2002, Khot [62] propôs a *Unique Games Conjecture*. Em 2007, Khot, Kindler, Mossel e O'Donnell [63] mostraram que se essa conjectura for verdadeira, então o fator de aproximação α_{GW} para o Problema do Corte Máximo é o melhor possível. Os resultados envolvendo fatores de inaproximabilidade a partir dessa conjectura foram posteriormente estendidos por Raghavendra [83] para uma família ainda maior de problemas.

5.2.3 Classes especiais de grafos

Estes resultados estudam o Problema do Corte Máximo restrito a famílias específicas de grafos. A motivação é que, intuitivamente, pode ser mais fácil resolver ou aproximar o problema quando a estrutura do grafo é mais controlada, como por exemplo, o problema restrito a grafos bipartidos se torna trivial.

Casos exatos em tempo polinomial: O problema pode ser resolvido de forma exata e em tempo polinomial para diversas classes, entre elas: grafos planares [27, 47, 69, 89]; grafos fracamente bipartidos [44]; grafos que não podem ser contraídos a K_5 [8]; grafos sem longos ciclos ímpares [45]; cografos não-ponderados [10]; grafos linha não-ponderados [46]; e grafos com *treewidth* limitada [12].

Para cada inteiro $l > 0$, fixados os grafos $H_1, \ldots, H_l$, se existir algum H_i que é uma floresta

cujas componentes possuem no máximo um vértice de grau 3, então o problema pode ser resolvido de forma exata em tempo polinomial em grafos que não possuem H_j como subgrafo, para todo j. Caso contrário, o problema é NP-completo [53].

Fixe um grafo H. Se H é um subgrafo induzido do caminho em 4 vértices, P_4, então o problema não-ponderado pode ser resolvido de forma exata em tempo polinomial para grafos que não possuem H como subgrafo induzido. Caso contrário, o problema é NP-completo [53]. Se H é um grafo que pode ser desenhado no plano com no máximo 1 cruzamento de arestas, então o problema pode ser resolvido de forma exata em tempo polinomial para grafos que não possuem H como menor [53].

Na literatura, apareceram pelo menos duas tentativas de provar que o problema não-ponderado pode ser resolvido em tempo polinomial para grafos de intervalos unitários (*unit/proper interval graphs*, ou *indifference graphs*) [13, 15], que foram refutadas posteriormente [11, 68]. Os resultados corrigidos obtiveram algoritmos polinomiais para o problema não-ponderado em grafos que são, simultaneamente, *split* e de intervalos unitários (*split-indifference graphs*), e em grafos que induzem poucos P_4s [11].

Outras aproximações, PTAS e similares: Aplicações do ferramental desenvolvido por Goemans-Williamson alcançaram, para grafos onde os vértices têm grau no máximo 3, fatores de 0.921 [31] e 0.9326 [48] e, para grafos 3-regulares, um fator de 0.924 [31]. Abordagens combinatórias resultaram em uma 4/5-aproximação para grafos com vértices de grau no máximo 3 e uma 22/27-aproximação para grafos 3-regulares [48]. Fixado um grafo H qualquer, o problema admite um PTAS para grafos que não possuem H como menor [25]. Para grafos densos, o problema não-ponderado admite um esquema de aproximação aleatorizado [95] e um PTAS [4], enquanto a versão ponderada admite um PTAS, sob uma definição especial de densidade [96]. Para grafos aleatórios k-regulares, existe um algoritmo local próximo do ótimo [28]. Para o problema não-ponderado, existe uma $(\alpha_{GW} + 10^{-34})$-aproximação para grafos de intervalo, e uma $(\alpha_{GW} + 10^{-16})$-aproximação para grafos *split* [3].

Casos difíceis: O Problema do Corte Máximo é NP-completo em grafos que não podem ser contraídos a K_6 [8], grafos de permutação [33] e grafos de intervalo com contagem de intervalo quatro [34]. O problema não-ponderado é NP-completo para grafos cúbicos [98], grafos totais (não confunda com grafos completos) [46] e grafos de intervalo [2]. A versão não-ponderada também é NP-completa em grafos cordais (*chordal graphs*), grafos de caminhos não-direcionados (*undirected path graphs*), grafos *split*, grafos tripartidos e complementos de grafos bipartidos [12].

Outro resultado envolvendo uma conjectura é que, se a *Small-Set Expansion Hypothesis* for verdadeira, então existe uma constante $\varepsilon > 0$ tal que não existe $(1 - \varepsilon)$-aproximação de tempo polinomial para o problema em grafos *split* [3].

Algoritmos Parametrizados: Foram desenvolvidos algoritmos parametrizados pelo *crossing number* do grafo, de modo que, quando esse parâmetro é fixo, resolvem o problema em tempo polinomial para grafos 1-planares [23] e também para grafos mais gerais [20, 66]. Além disso, quando a parametrização é feita no valor do corte, o problema também é tratável em parâmetro fixo para grafos não-ponderados [22, 72, 73, 84] e para multigrafos [70].

5.2.4 Resultados espectrais

Técnicas espectrais também produziram resultados para este problema. Quanto a limitantes, mostrou-se ser possível, em tempo polinomial, computar um limite superior para o valor do corte máximo por meio da minimização de um parâmetro espectral do grafo, obtido a partir da sua matriz laplaciana [24]. Quanto a algoritmos, um critério espectral guiado pelo menor autovalor da matriz de adjacências do grafo fornece uma 0.531-aproximação [92].

6 Considerações finais

Este relatório reuniu e organizou conceitos, técnicas e resultados centrais sobre problemas de cortes em grafos, com ênfase no Problema do Corte Mínimo e no Problema do Corte Máximo. O objetivo principal foi investigar o estado da arte e consolidar o ferramental teórico que sustenta as diferentes abordagens para esses problemas, privilegiando a compreensão das ideias de prova e das fronteiras atualmente conhecidas.

Para o Corte Mínimo, o texto apresentou a equivalência clássica com o Fluxo Máximo e os algoritmos polinomiais que exploram essa dualidade, além de abordagens alternativas envolvendo contração de arestas e empacotamento de árvores. Para o Corte Máximo, apresentou-se a complexidade do problema e revisaram-se as principais técnicas de aproximação para o caso geral, que utilizam método guloso, busca local, aleatorização e relaxações lineares, e culminam na relaxação semidefinida com arredondamento por hiperplano aleatório, expressa no Algoritmo de Goemans-Williamson. O quadro do Corte Máximo foi complementado com referências a resultados para o problema restrito a classes particulares de grafos, além de resultados sobre limites de inaproximabilidade, que ajudam a entender por que a literatura ainda não superou o fator de aproximação $\alpha_{GW} \approx 0.878$ para o caso geral.

Do ponto de vista formativo, o processo de estudo trouxe ganhos concretos. Entre eles, pode-se citar uma maior familiaridade com as técnicas de projeto e análise de algoritmos (exatos e de aproximação); a compreensão de formulações matemáticas, como programação linear inteira e programação semidefinida, além dos conceitos de relaxação, arredondamento e *gaps*; a capacidade de interpretar conceitos teóricos como complexidade e redução de problemas, fatores de aproximação e barreiras de inaproximabilidade; e, sobretudo, prática de leitura crítica, desenvolvendo a habilidade de selecionar e explicar o que cada método garante e quais são os resultados alcançados, considerando o escopo do trabalho. Esses elementos introduzem de maneira consistente o autor à pesquisa científica e complementam sua formação em Ciência da Computação, oferecendo uma base sólida para leituras e investigações posteriores na área.

Referências

[1] Abboud, A., Krauthgamer, R., Li, J., Panigrahi, D., Saranurak, T., Trabelsi, O.: Breaking the cubic barrier for all-pairs max-flow: Gomory-hu tree in nearly quadratic time. In: 2022 IEEE 63rd Annual Symposium on Foundations of Computer Science (FOCS). pp. 884–895 (2022). DOI 10.1109/FOCS54457.2022.00088

[2] Adhikary, R., Bose, K., Mukherjee, S., Roy, B.: Complexity of maximum cut on interval graphs. Discrete & Computational Geometry **70**(2), 307–322 (Jan 2023). DOI 10.1007/s00454-022-00472-y

[3] Ahn, J., DeHaan, I., Kim, E.J., Lee, E.: Approximating maximum cut on interval graphs and split graphs

beyond goemans-williamson. In: Ene, A., Chattopadhyay, E. (eds.) Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2025). Leibniz International Proceedings in Informatics (LIPIcs), vol. 353, pp. 20:1–20:22. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany (2025). DOI 10.4230/LIPIcs.APPROX/RANDOM.2025.20

[4] Arora, S., Karger, D., Karpinski, M.: Polynomial time approximation schemes for dense instances of np-hard problems. Journal of Computer and System Sciences **58**(1), 193–210 (1999). DOI 10.1006/jcss.1998.1605

[5] Arora, S., Lund, C., Motwani, R., Sudan, M., Szegedy, M.: Proof verification and the hardness of approximation problems. J. ACM **45**(3), 501–555 (May 1998). DOI 10.1145/278298.278306

[6] Bae, E., Shi, J., Tai, X.C.: Graph cuts for curvature based image denoising. IEEE Transactions on Image Processing **20**(5), 1199–1210 (2011). DOI 10.1109/TIP.2010.2090533

[7] Ball, M.O., Colbourn, C.J., Provan, J.S.: Chapter 11 network reliability. In: Network Models, Handbooks in Operations Research and Management Science, vol. 7, pp. 673–762. Elsevier (1995). DOI 10.1016/S0927-0507(05)80128-8

[8] Barahona, F.: The max-cut problem on graphs not contractible to k5. Operations Research Letters **2**(3), 107–111 (1983). DOI 10.1016/0167-6377(83)90016-0

[9] Bellare, M., Goldreich, O., Sudan, M.: Free bits, pcps, and nonapproximability—towards tight results. SIAM Journal on Computing **27**(3), 804–915 (1998). DOI 10.1137/S0097539796302531

[10] Bodlaender, H.L.: The maximum cut and minimum cut into bounded sets problems on cographs (1987)

[11] Bodlaender, H.L., de Figueiredo, C.M.H., Gutierrez, M., Kloks, T., Niedermeier, R.: Simple max-cut for split-indifference graphs and graphs with few p4's. In: Ribeiro, C.C., Martins, S.L. (eds.) Experimental and Efficient Algorithms. pp. 87–99. Springer Berlin Heidelberg, Berlin, Heidelberg (2004)

[12] Bodlaender, H.L., Jansen, K.: On the complexity of the maximum cut problem. Nordic J. of Computing **7**(1), 14–31 (Mar 2000)

[13] Bodlaender, H.L., Kloks, T., Niedermeier, R.: Simple max-cut for unit interval graphs and graphs with few p4s. Electronic Notes in Discrete Mathematics **3**, 19–26 (1999). DOI 10.1016/S1571-0653(05)80014-9, 6th Twente Workshop on Graphs and Combinatorial Optimization

[14] Bondy, J.A., Murty, U.S.R.: Graph theory, Graduate Texts in Mathematics, vol. 244. Springer, New York (2008). DOI 10.1007/978-1-84628-970-5

[15] Boyacı, A., Ekim, T., Shalom, M.: A polynomial-time algorithm for the maximum cardinality cut problem in proper interval graphs. Information Processing Letters **121**, 29–33 (2017). DOI 10.1016/j.ipl.2017.01.007

[16] Boykov, Y., Veksler, O.: Graph Cuts in Vision and Graphics: Theories and Applications, pp. 79–96. Springer US, Boston, MA (2006). DOI 10.1007/0-387-28831-7_5

[17] Boykov, Y., Funka-Lea, G.: Graph cuts and efficient n-d image segmentation. International Journal of Computer Vision **70**(2), 109–131 (Nov 2006). DOI 10.1007/s11263-006-7934-5

[18] Chan, S.O., Lee, J.R., Raghavendra, P., Steurer, D.: Approximate constraint satisfaction requires large lp relaxations. J. ACM **63**(4) (Oct 2016). DOI 10.1145/2811255

[19] Chen, X., Pan, L.: A survey of graph cuts/graph search based medical image segmentation. IEEE Reviews in Biomedical Engineering **11**, 112–124 (2018). DOI 10.1109/RBME.2018.2798701

[20] Chimani, M., Dahn, C., Juhnke-Kubitzke, M., Kriege, N., Mutzel, P., Nover, A.: Maximum cut parameterized by crossing number. Journal of Graph Algorithms and Applications **24**(3), 155–170 (Mar 2020). DOI 10.7155/jgaa.00523

[21] Cormen, T., Leiserson, C., Rivest, R., Stein, C.: Introduction to Algorithms. The MIT Press, 4th edn. (2022)

[22] Crowston, R., Jones, M., Mnich, M.: Max-cut parameterized above the edwards-erdős bound. Algorithmica **72**(3), 734–757 (Jan 2014). DOI 10.1007/s00453-014-9870-z

[23] Dahn, C., Kriege, N.M., Mutzel, P., Schilling, J.: Fixed-parameter algorithms for the weighted max-cut problem on embedded 1-planar graphs. Theoretical Computer Science **852**, 172–184 (2021). DOI 10.1016/j.tcs.2020.11.030

[24] Delorme, C., Poljak, S.: Laplacian eigenvalues and the maximum cut problem. Mathematical Programming **62**(1–3), 557–574 (Feb 1993). DOI 10.1007/bf01585184

[25] Demaine, E., Hajiaghayi, M., Kawarabayashi, K.: Algorithmic graph minor theory: Decomposition, appro-

ximation, and coloring. In: 46th Annual IEEE Symposium on Foundations of Computer Science (FOCS'05). pp. 637–646 (2005). DOI 10.1109/SFCS.2005.14

[26] Dinitz, Y.: Dinitz' Algorithm: The Original Version and Even's Version, pp. 218–240. Springer Berlin Heidelberg, Berlin, Heidelberg (2006). DOI 10.1007/11685654_10

[27] Dorfman, Y., Orlova, G.: Finding the maximal cut in a graph. Engineering Cybernetics **10**(3) (1972)

[28] El Alaoui, A., Montanari, A., Sellke, M.: Local algorithms for maximum cut and minimum bisection on locally treelike regular graphs of large degree. Random Structures & Algorithms **63**(3), 689–715 (2023). DOI 10.1002/rsa.21149

[29] Erickson, J.: Algorithms. Independently published (Jun 2019), https://jeffe.cs.illinois.edu/teaching/algorithms/

[30] Even, S.: Graph Algorithms. Cambridge University Press, USA, 2nd edn. (2011)

[31] Feige, U., Karpinski, M., Langberg, M.: Improved approximation of max-cut on graphs of bounded degree. Journal of Algorithms **43**(2), 201–219 (2002). DOI 10.1016/S0196-6774(02)00005-6

[32] Feige, U., Schechtman, G.: On the optimality of the random hyperplane rounding technique for max cut. Random Structures & Algorithms **20**(3), 403–440 (2002). DOI 10.1002/rsa.10036

[33] de Figueiredo, C.M.H., de Melo, A.A., Oliveira, F.S., Silva, A.: Maxcut on permutation graphs is np-complete. Journal of Graph Theory **104**(1), 5–16 (2023). DOI 10.1002/jgt.22948

[34] de Figueiredo, C.M.H., de Melo, A.A., Oliveira, F.S., Silva, A.: Maximum cut on interval graphs of interval count four is np-complete. Discrete & Computational Geometry **71**(3), 893–917 (Apr 2024). DOI 10.1007/s00454-023-00508-x

[35] Fulkerson, D.R., Ford, L.R.: Maximal flow through a network. Canadian journal of Mathematics **8**, 399–404 (1956). DOI 10.4153/CJM-1956-045-5

[36] Gabow, H.N.: A matroid approach to finding edge connectivity and packing arborescences. In: Proceedings of the Twenty-Third Annual ACM Symposium on Theory of Computing. p. 112–122. STOC '91, Association for Computing Machinery, New York, NY, USA (1991). DOI 10.1145/103418.103436

[37] Gabow, H.: A matroid approach to finding edge connectivity and packing arborescences. Journal of Computer and System Sciences **50**(2), 259–273 (1995). DOI 10.1006/jcss.1995.1022

[38] Garey, M.R., Johnson, D.S.: Computers and Intractability; A Guide to the Theory of NP-Completeness. W. H. Freeman & Co., USA (1990)

[39] Garey, M., Johnson, D., Stockmeyer, L.: Some simplified np-complete graph problems. Theoretical Computer Science **1**(3), 237–267 (1976). DOI 10.1016/0304-3975(76)90059-1

[40] Gawrychowski, P., Mozes, S., Weimann, O.: Minimum cut in $O(m \log^2 n)$ Time. Theory of Computing Systems **68**(4), 814–834 (Aug 2024). DOI 10.1007/s00224-024-10179-7

[41] Goemans, M.X., Williamson, D.P.: Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. J. ACM **42**(6), 1115–1145 (Nov 1995). DOI 10.1145/227683.227684

[42] Goldberg, A.V., Tarjan, R.E.: A new approach to the maximum-flow problem. J. ACM **35**(4), 921–940 (Oct 1988). DOI 10.1145/48014.61051

[43] Gomory, R.E., Hu, T.C.: Multi-terminal network flows. Journal of the Society for Industrial and Applied Mathematics **9**(4), 551–570 (1961). DOI 10.1137/0109047

[44] Grötschel, M., Pulleyblank, W.: Weakly bipartite graphs and the max-cut problem. Operations Research Letters **1**(1), 23–27 (1981). DOI 10.1016/0167-6377(81)90020-1

[45] Grötschel, M., Nemhauser, G.L.: A polynomial algorithm for the max-cut problem on graphs without long odd cycles. Mathematical Programming **29**(1), 28–40 (May 1984). DOI 10.1007/bf02591727

[46] Guruswami, V.: Maximum cut on line and total graphs. Discrete Applied Mathematics **92**(2), 217–221 (1999). DOI 10.1016/S0166-218X(99)00056-6

[47] Hadlock, F.: Finding a maximum cut of a planar graph in polynomial time. SIAM Journal on Computing **4**(3), 221–225 (1975). DOI 10.1137/0204019

[48] Halperin, E., Livnat, D., Zwick, U.: Max cut in cubic graphs. Journal of Algorithms **53**(2), 169–185 (2004). DOI 10.1016/j.jalgor.2004.06.001

[49] Hao, J., Orlin, J.: A faster algorithm for finding the minimum cut in a directed graph. Journal of Algorithms

17(3), 424–446 (1994). DOI 10.1006/jagm.1994.1043

[50] Harris, D.G., Srinivasan, A.: Improved bounds and algorithms for graph cuts and network reliability. Random Structures & Algorithms **52**(1), 74–135 (2018). DOI 10.1002/rsa.20724

[51] Håstad, J.: Some optimal inapproximability results. J. ACM **48**(4), 798–859 (Jul 2001). DOI 10.1145/502090.502098

[52] Kahng, A.B., Lienig, J., Markov, I.L., Hu, J.: VLSI Physical Design: From Graph Partitioning to Timing Closure. Springer Netherlands (2011). DOI 10.1007/978-90-481-9591-6

[53] Kamiński, M.: max-cut and containment relations in graphs. Theoretical Computer Science **438**, 89–95 (2012). DOI 10.1016/j.tcs.2012.02.036

[54] Karger, D.R.: Global min-cuts in rnc, and other ramifications of a simple min-cut algorithm. In: Proceedings of the Fourth Annual ACM-SIAM Symposium on Discrete Algorithms. p. 21–30. SODA '93, Society for Industrial and Applied Mathematics, USA (1993), https://dl.acm.org/doi/10.5555/313559.313605

[55] Karger, D.R.: A randomized fully polynomial time approximation scheme for the all terminal network reliability problem. In: Proceedings of the Twenty-Seventh Annual ACM Symposium on Theory of Computing. p. 11–17. STOC '95, Association for Computing Machinery, New York, NY, USA (1995). DOI 10.1145/225058.225069

[56] Karger, D.R.: Minimum cuts in near-linear time. J. ACM **47**(1), 46–76 (Jan 2000). DOI 10.1145/331605.331608

[57] Karger, D.R., Stein, C.: An $\tilde{O}(n^2)$ algorithm for minimum cuts. In: Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing. p. 757–765. STOC '93, Association for Computing Machinery, New York, NY, USA (1993). DOI 10.1145/167088.167281

[58] Karger, D.R., Stein, C.: A new approach to the minimum cut problem. J. ACM **43**(4), 601–640 (Jul 1996). DOI 10.1145/234533.234534

[59] Karloff, H.: How good is the goemans–williamson max cut algorithm? SIAM Journal on Computing **29**(1), 336–350 (1999). DOI 10.1137/S0097539797321481

[60] Karp, R.M.: Reducibility among Combinatorial Problems, pp. 85–103. Springer US, Boston, MA (1972). DOI 10.1007/978-1-4684-2001-2_9

[61] Keusch, R.: A solution to the 1-2-3 conjecture. Journal of Combinatorial Theory, Series B **166**, 183–202 (2024). DOI 10.1016/j.jctb.2024.01.002

[62] Khot, S.: On the power of unique 2-prover 1-round games. In: Proceedings of the Thiry-Fourth Annual ACM Symposium on Theory of Computing. p. 767–775. STOC '02, Association for Computing Machinery, New York, NY, USA (2002). DOI 10.1145/509907.510017

[63] Khot, S., Kindler, G., Mossel, E., O'Donnell, R.: Optimal inapproximability results for max-cut and other 2-variable csps? SIAM Journal on Computing **37**(1), 319–357 (2007). DOI 10.1137/S0097539705447372

[64] King, V., Rao, S., Tarjan, R.: A faster deterministic maximum flow algorithm. Journal of Algorithms **17**(3), 447–474 (1994). DOI 10.1006/jagm.1994.1044

[65] Kleinberg, J., Tardos, E.: Algorithm Design. Addison-Wesley Longman Publishing Co., Inc., USA (2005)

[66] Kobayashi, Y., Kobayashi, Y., Miyazaki, S., Tamaki, S.: An improved fixed-parameter algorithm for max-cut parameterized by crossing number. In: Colbourn, C.J., Grossi, R., Pisanti, N. (eds.) Combinatorial Algorithms. pp. 327–338. Springer International Publishing, Cham (2019)

[67] Kolmogorov, V., Zabih, R.: Multi-camera scene reconstruction via graph cuts. In: Heyden, A., Sparr, G., Nielsen, M., Johansen, P. (eds.) Computer Vision — ECCV 2002. pp. 82–96. Springer Berlin Heidelberg, Berlin, Heidelberg (2002)

[68] Kratochvíl, J., Masařík, T., Novotná, J.: $\mathcal{U}$-bubble model for mixed unit interval graphs and its applications: The maxcut problem revisited. Algorithmica **83**(12), 3649–3680 (Sep 2021). DOI 10.1007/s00453-021-00837-4

[69] Liers, F., Pardella, G.: Partitioning planar graphs: a fast combinatorial approach for max-cut. Computational Optimization and Applications **51**(1), 323–344 (Jan 2012). DOI 10.1007/s10589-010-9335-5

[70] Lill, J., Petrova, K., Weber, S.: Linear-time maxcut in multigraphs parameterized above the poljak-turzík bound. Algorithmica **87**(7), 983–1007 (Apr 2025). DOI 10.1007/s00453-025-01306-y

[71] Lovász, L., Plummer, M.: Matching Theory. Annals of Discrete Mathematics, North Holland (1986)

[72] Madathil, J., Saurabh, S., Zehavi, M.: Max-cut above spanning tree is fixed-parameter tractable. In: Fomin, F.V., Podolskii, V.V. (eds.) Computer Science – Theory and Applications. pp. 244–256. Springer International Publishing, Cham (2018)

[73] Mahajan, M., Raman, V.: Parameterizing above guaranteed values: Maxsat and maxcut. Journal of Algorithms **31**(2), 335–354 (1999). DOI 10.1006/jagm.1998.0996

[74] Mahajan, S., Ramesh, H.: Derandomizing semidefinite programming based approximation algorithms. In: Proceedings of IEEE 36th Annual Foundations of Computer Science. pp. 162–169 (1995). DOI 10.1109/SFCS.1995.492473

[75] Malcolm, J., Rathi, Y., Tannenbaum, A.: Multi-object tracking through clutter using graph cuts. In: 2007 IEEE 11th International Conference on Computer Vision. pp. 1–5 (2007). DOI 10.1109/ICCV.2007.4409178

[76] Mitzenmacher, M., Upfal, E.: Probability and Computing: Randomized Algorithms and Probabilistic Analysis. Cambridge University Press (2005)

[77] Mooser, J., You, S., Neumann, U.: Real-time object tracking for augmented reality combining graph cuts and optical flow. In: 2007 6th IEEE and ACM International Symposium on Mixed and Augmented Reality. pp. 145–152 (2007). DOI 10.1109/ISMAR.2007.4538839

[78] Nagamochi, H., Ibaraki, T.: Computing edge-connectivity in multigraphs and capacitated graphs. SIAM Journal on Discrete Mathematics **5**(1), 54–66 (1992). DOI 10.1137/0405004

[79] Nagamochi, H., Ibaraki, T.: A linear-time algorithm for finding a sparse k-connected spanning subgraph of a k-connected graph. Algorithmica **7**(1), 583–596 (Jun 1992). DOI 10.1007/BF01758778

[80] Nash-Williams, C.S.A.: Edge-disjoint spanning trees of finite graphs. Journal of the London Mathematical Society **s1-36**(1), 445–450 (1961). DOI 10.1112/jlms/s1-36.1.445

[81] Orlin, J.B.: Max flows in o(nm) time, or better. In: STOC '13: Proceedings of the forty-fifth annual ACM symposium on Theory of Computing. p. 765–774. STOC '13, Association for Computing Machinery, New York, NY, USA (2013). DOI 10.1145/2488608.2488705

[82] Papadakis, N., Bugeau, A.: Tracking with occlusions via graph cuts. IEEE Transactions on Pattern Analysis and Machine Intelligence **33**(1), 144–157 (2011). DOI 10.1109/TPAMI.2010.56

[83] Raghavendra, P.: Optimal algorithms and inapproximability results for every csp? In: Proceedings of the Fortieth Annual ACM Symposium on Theory of Computing. p. 245–254. STOC '08, Association for Computing Machinery, New York, NY, USA (2008). DOI 10.1145/1374376.1374414

[84] Raman, V., Saurabh, S.: Improved fixed parameter tractable algorithms for two "edge" problems: Maxcut and maxdag. Information Processing Letters **104**(2), 65–72 (2007). DOI 10.1016/j.ipl.2007.05.014

[85] Sahni, S., Gonzalez, T.: P-complete approximation problems. J. ACM **23**(3), 555–565 (Jul 1976). DOI 10.1145/321958.321975

[86] Schäffer, A.A.: Simple local search problems that are hard to solve. SIAM Journal on Computing **20**(1), 56–87 (1991). DOI 10.1137/0220004

[87] Schrijver, A.: Theory of linear and integer programming. John Wiley & Sons, Inc., USA (1986)

[88] Sherwani, N.A.: Algorithms for VLSI Physical Design Automation. Springer US (1993). DOI 10.1007/978-1-4757-2219-2

[89] Shih, W.K., Wu, S., Kuo, Y.: Unifying maximum cut and minimum cut of a planar graph. IEEE Transactions on Computers **39**(5), 694–697 (1990). DOI 10.1109/12.53581

[90] Stoer, M., Wagner, F.: A simple min-cut algorithm. J. ACM **44**(4), 585–591 (Jul 1997). DOI 10.1145/263867.263872

[91] Tran, S., Davis, L.: 3d surface reconstruction using graph cuts with surface constraints. In: Leonardis, A., Bischof, H., Pinz, A. (eds.) Computer Vision – ECCV 2006. pp. 219–231. Springer Berlin Heidelberg, Berlin, Heidelberg (2006)

[92] Trevisan, L.: Max cut and the smallest eigenvalue. In: Proceedings of the Forty-First Annual ACM Symposium on Theory of Computing. p. 263–272. STOC '09, Association for Computing Machinery, New York, NY, USA (2009). DOI 10.1145/1536414.1536452

[93] Trevisan, L., Sorkin, G.B., Sudan, M., Williamson, D.P.: Gadgets, approximation, and linear programming. SIAM Journal on Computing **29**(6), 2074–2097 (2000). DOI 10.1137/S0097539797328847

[94] Vazirani, V.V.: Approximation Algorithms. Springer Publishing Company, Incorporated (2010)

[95] Fernandez de la Vega, W.: Max-cut has a randomized approximation scheme in dense graphs. Random Structures & Algorithms **8**(3), 187–198 (1996). DOI 10.1002/(SICI)1098-2418(199605)8:3<187::AID-RSA3>3.0.CO;2-U

[96] Fernandez de la Vega, W., Karpinski, M.: Polynomial time approximation of dense weighted instances of max-cut. Random Structures & Algorithms **16**(4), 314–332 (2000). DOI 10.1002/1098-2418(200007)16:4<314::AID-RSA2>3.0.CO;2-E

[97] Williamson, D.P., Shmoys, D.B.: The Design of Approximation Algorithms. Cambridge University Press, USA, 1st edn. (2011)

[98] Yannakakis, M.: Node-and edge-deletion np-complete problems. In: Proceedings of the Tenth Annual ACM Symposium on Theory of Computing. p. 253–264. STOC '78, Association for Computing Machinery, New York, NY, USA (1978). DOI 10.1145/800133.804355

[99] Zwick, U.: The smallest networks on which the ford-fulkerson maximum flow procedure may fail to terminate. Theoretical Computer Science **148**(1), 165–170 (1995). DOI 10.1016/0304-3975(95)00022-O

A Detalhes omitidos no texto principal sobre o problema do Corte Mínimo

Um exemplo de simulação do algoritmo de Ford-Fulkerson é ilustrado na Figura 4.

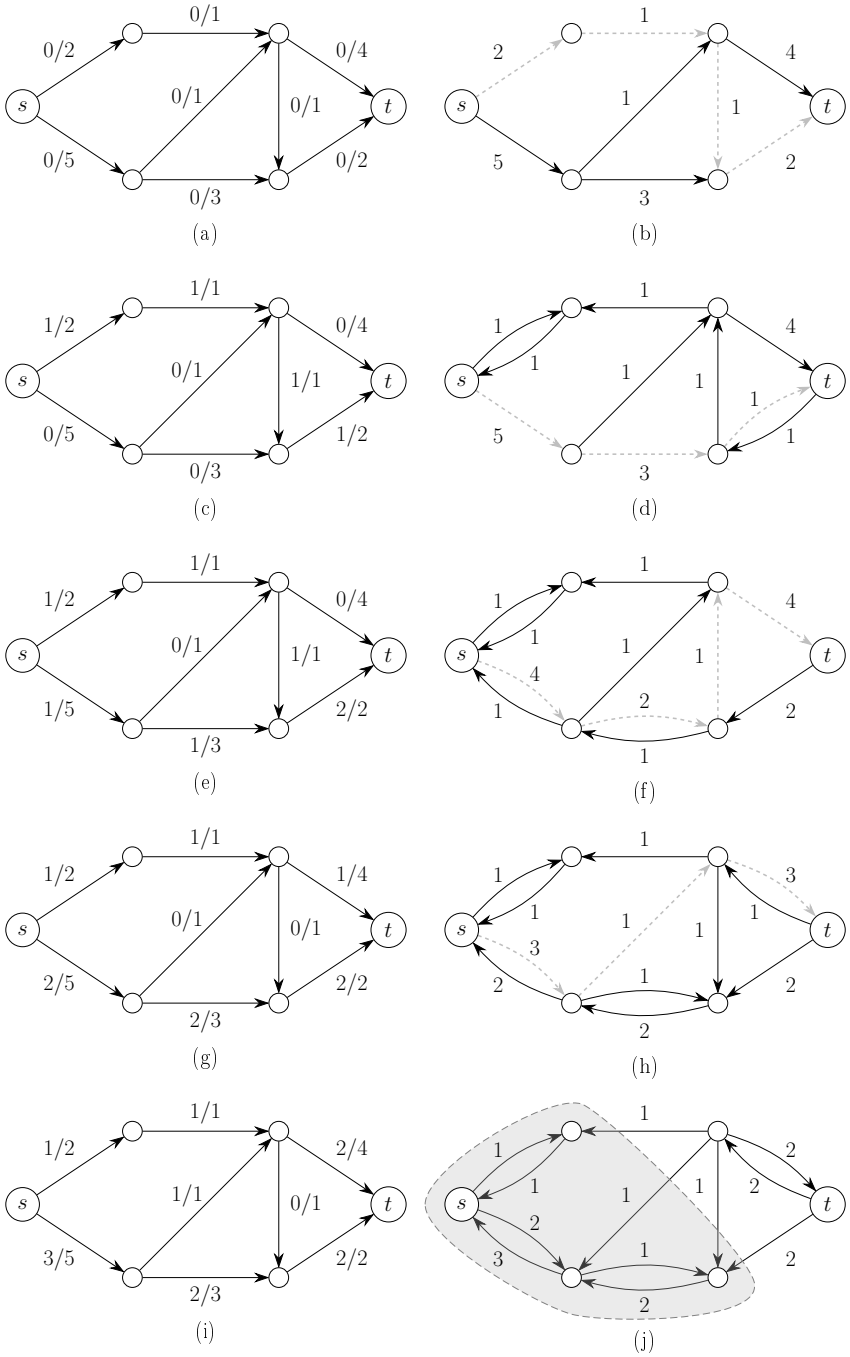

Figura 4: Exemplo de simulação do Algoritmo 1. (a) Digrafo inicial D, em sua versão reduzida. Os números em cada arco representam, respectivamente, o fluxo e a capacidade do arco. (b) Digrafo residual D_f e caminho aumentador P em cinza tracejado. (c) Atualização de D ao longo de P. (d)-(i) Iterações seguintes. (j) Versão final de D_f, onde a região cinza representa o conjunto S.

A Figura 5 contém um exemplo que ilustra a execução do algoritmo de Dinitz.

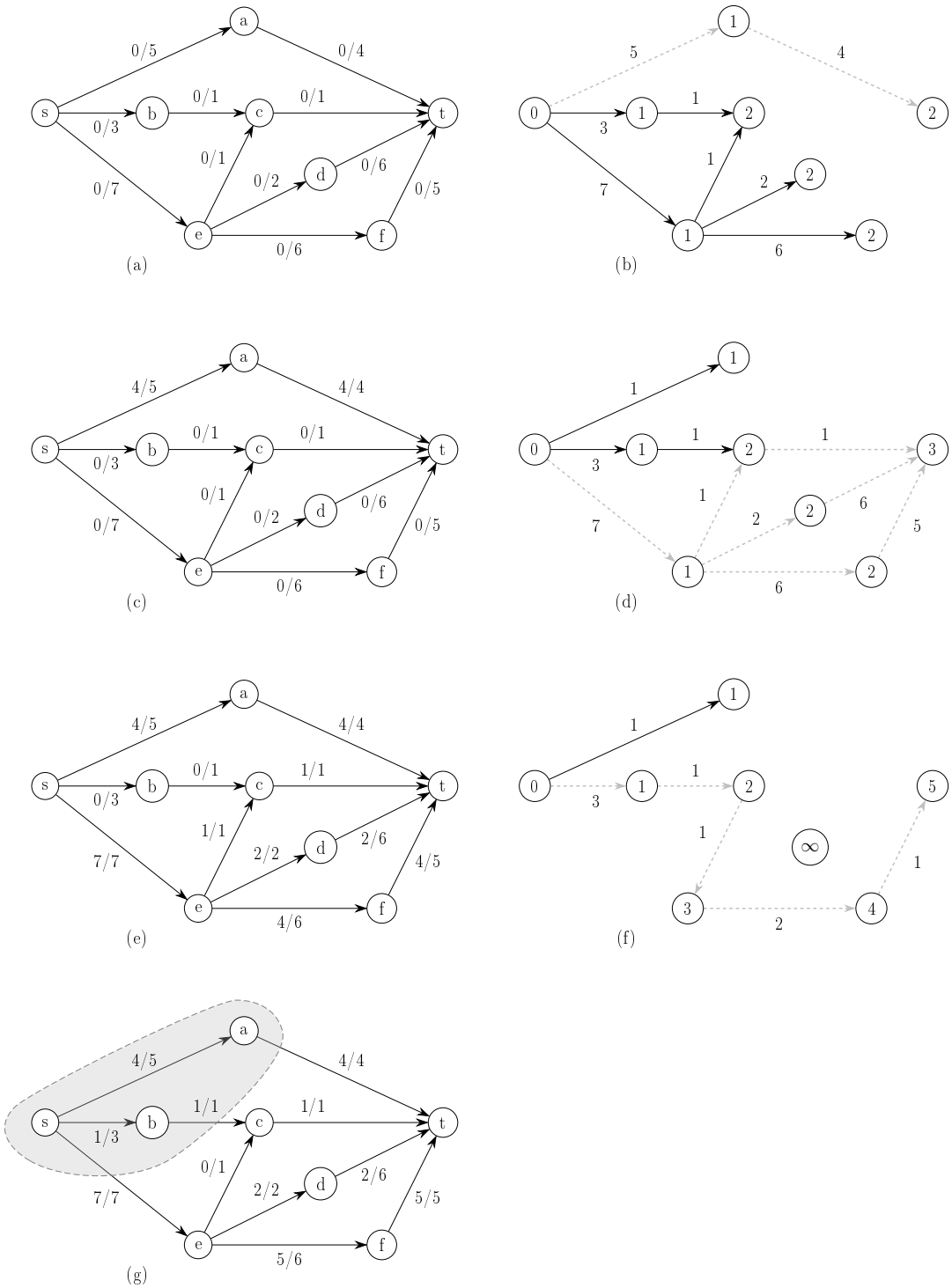

Figura 5: Exemplo de execução do Algoritmo de Dinitz. (a) Digrafo inicial, onde os números em cada arco representam, respectivamente, o fluxo e a capacidade do arco. (b) Estrutura gerada pela BFS estendida. Os números nos vértices indicam suas distâncias a partir de s. Nesta fase, o único caminho aumentador é (s, a, t), destacado em cinza tracejado, que transfere 4 unidades de fluxo (u.f.). (c) Digrafo resultante após a primeira fase. (d) Nova estrutura gerada pela BFS estendida. Três caminhos aumentadores são encontrados: primeiro, (s, e, c, t) transfere 1 u.f.; depois, (s, e, d, t) transfere 2 u.f.; por fim, (s, e, f, t) transfere 4 u.f. (e) Digrafo após a segunda fase. (f) Estrutura gerada pela BFS estendida. Resta apenas o caminho (s, b, c, e, f, t), que transfere 1 u.f. (g) Digrafo final. A BFS estendida não alcança mais o vértice t. O conjunto dos vértices alcançáveis a partir de s, destacado pela região cinza, é $S = \{s, a, b\}$ e define o corte mínimo de valor 12.

A.0.1 Aplicações do st-Fluxo Máximo

Nesta seção, ilustramos a versatilidade do fluxo máximo na resolução de diversos problemas combinatórios. As aplicações aqui descritas foram extraídas do texto de Erickson [29], que também apresenta outros exemplos interessantes.

Emparelhamento em Grafos Bipartidos: Um grafo $G = (V, E)$ não-direcionado é *bipartido* se V pode ser particionado em subconjuntos disjuntos $X, Y \subset V$ tais que $X \cup Y = V$, e toda aresta em E tem uma extremidade em X e outra em Y, isto é, $E[X, Y] = E$. Um *emparelhamento* em G é um subconjunto de arestas duas a duas disjuntas por vértices.

Dado um grafo $G = (V, E)$ e sua bipartição $\{X, Y\}$, o objetivo é determinar o tamanho do maior emparelhamento em G. Para isso, modelamos esse problema como um problema de fluxo máximo. Orientamos toda aresta de X para Y, criamos novos vértices s e t, adicionamos arcos sx para todo $x \in X$ e arcos yt para todo $y \in Y$. Finalmente, atribuímos capacidade 1 a todos os arcos e computamos o fluxo máximo do grafo resultante, que corresponde ao tamanho do maior emparelhamento em G.

Caminhos Aresta-Disjuntos: Dado um grafo $G = (V, E)$, podendo ser orientado ou não, e vértices $s, t \in V$, o objetivo é determinar quantos caminhos aresta-disjuntos existem de s a t. Podemos resolver esse problema atribuindo capacidade 1 a todas as arestas (ou arcos) e computando o fluxo máximo. Seu valor será exatamente a quantidade de caminhos aresta-disjuntos entre s e t.

Fluxo Máximo com Capacidades nos Vértices: Considere o caso em que, além do digrafo $D = (V, A)$ com $s, t \in V$ e da função $c_a : V^2 \to \mathbb{R}_{\geq 0}$ de capacidades nos arcos, há também uma função $c_v : V \to \mathbb{R}_{\geq 0}$ que atribui capacidades aos vértices. Podemos remodelar o problema construindo uma instância $D' = (V', A')$ com apenas uma função $c' : V' \times V' \to \mathbb{R}_{\geq 0}$ de capacidades nos arcos. Para cada vértice $v \in V$, criamos v^- e v^+ em V', conectados por um arco $v^-v^+ \in A'$ com capacidade $c'(v^-v^+) = c_v(v)$. Além disso, cada arco $uw \in A$ terá um correspondente $u^+w^- \in A'$ com $c'(u^+w^-) = c_a(uw)$. Dessa forma, transformamos a instância do problema original para uma instância do Problema do Fluxo Máximo convencional, preservando o valor da solução.

Caminhos Vértice-Disjuntos: Dado um grafo $G = (V, E)$ com $s, t \in V$, deseja-se saber quantos caminhos vértice-disjuntos existem de s a t, isto é, caminhos que não compartilham vértices entre si, a menos das extremidades s e t. Note que a disjunção por vértices é mais restritiva que a disjunção por arestas. Todavia, este problema também pode ser resolvido com o ferramental desenvolvido para o Problema do Fluxo Máximo. Mais especificamente, atribuímos capacidade 1 a cada vértice, além de substituir cada aresta por um par de arcos opostos com capacidades unitárias. Desse modo, obtemos uma instância do Problema do Fluxo Máximo com Capacidades nos Vértices. O valor do fluxo máximo resultante corresponde ao número de caminhos vértice-disjuntos de s a t.

A.0.2 O Problema do Fluxo Máximo Global e Outros Avanços

É importante observar que Ford e Fulkerson resolveram, originalmente, o st-Fluxo Máximo, definido no Problema 4.3. Naturalmente, surge o problema do *Fluxo Máximo Global*, cuja entrada é quase a mesma, exceto pelo fato de que os vértices s e t não estão especificados. Assim, o objetivo é encontrar o fluxo viável de maior valor entre qualquer par de vértices.

Em um grafo de n vértices e m arestas, uma abordagem imediata para resolver o Fluxo Máximo Global seria computar o st-fluxo máximo para todas as $\binom{n}{2}$ escolhas possíveis de vértices s e t e devolver o maior valor obtido. Da mesma forma, se estivermos interessados em resolver o Corte Mínimo Global — definido no Problema 4.2 — via st-fluxos, aplicamos o mesmo procedimento, mas devolvemos o menor valor obtido. O algoritmo resultante possui tempo polinomial, pois a quantidade de pares de vértices (s, t) possíveis é $O(n^2)$ e a computação de um st-fluxo é polinomial em n e m.

Entretanto, esse problema pode ser resolvido de forma ainda mais eficiente. Em 1961, Gomory e Hu [43] mostraram que apenas $n - 1$ computações de st-fluxos são suficientes para resolver o Fluxo Máximo Global, o que deu origem às Árvores de Gomory-Hu. Posteriormente, em 1994, Hao e Orlin [49] mostraram que, essencialmente, o tempo de computação de um único st-fluxo é suficiente.

Outros avanços incluem o trabalho de Goldberg e Tarjan [42], que desenvolveram o método de *Push-Relabel* para o st-Fluxo Máximo, alcançando um algoritmo de tempo $O(n^3)$ ou, utilizando a estrutura de *dynamic trees*, $O(nm \log(n^2/m))$. Em 1994, King, Rao e Tarjan [64] apresentaram um algoritmo com tempo $O(nm)$ sempre que valer $m/n \in \Omega(n^\epsilon)$ para alguma constante $\epsilon > 0$. Em 2013, Orlin [81] propôs um algoritmo de tempo $O(nm)$ para quaisquer valores de n e m. Mais recentemente, em 2022, Abboud, Krauthgamer, Li, Panigrahi, Saranurak e Trabelsi [1] desenvolveram um algoritmo com tempo $\tilde{O}(n^2)$ para o Fluxo Máximo Global, onde $\tilde{O}(n^2)$ denota $O(n^2 \log^k(n))$ para alguma constante $k > 0$.

De fato, há uma vasta literatura para problemas de fluxo em grafos. Esses resultados podem ser explorados com mais profundidade em Cormen *et al.* [21, Cap. 24]; Erickson [29, Cap. 10]; e Goldberg e Tarjan [42, Tab. 1].

B Aproximações com fator 1/2 para o problema do Corte Máximo

Na literatura, diversas técnicas têm sido estudadas para o Problema do Corte Máximo. Muitas delas garantem fator de aproximação 1/2. Essa seção apresenta algumas dessas aproximações clássicas.

B.0.1 Método Guloso

Esta técnica de aproximação é conhecida, pelo menos, desde 1976, conforme Sahni e Gonzalez [85]. Dado um grafo $G = (V, E)$ e uma função de pesos $w : E \to \mathbb{R}_{\geq 0}$, fixa-se uma ordem arbitrária nos vértices $V = \{v_1, v_2, \ldots, v_n\}$. O algoritmo opera com dois conjuntos disjuntos S e T, inicialmente vazios, e realiza n iterações. Para cada $i \in \{1, \ldots, n\}$, na i-ésima

iteração, o algoritmo determina em qual dos conjuntos o vértice v_i será colocado. Mais especificamente, considerando o estado de S e T na i-ésima iteração, defina os cortes $C_i^S := E[S, \{v_i\}]$ e $C_i^T := E[T, \{v_i\}]$. Se $w(C_i^S) \geq w(C_i^T)$, coloca-se v_i em T pois, dessa forma, toda aresta em C_i^S cruza o corte. Caso contrário, coloca-se v_i em S. Ao final do processo, devolve-se o corte definido pelas partes S e T. Afirmamos que este algoritmo resulta em uma 1/2-aproximação [3]. De fato, para cada aresta do grafo, defina a extremidade que aparece mais tarde na ordenação tomada pelo algoritmo como *responsável* pela aresta. Além disso, seja $R_i := \{v_j v_i \in E : j < i\}$ o conjunto de arestas pelas quais o vértice v_i é responsável. Note que, na i-ésima iteração, toda aresta pela qual v_i é responsável já tem a outra extremidade pertencente (exclusivamente) a S ou a T, donde vale a igualdade $w(C_i^S) + w(C_i^T) = w(R_i)$. Como é impossível que as duas parcelas sejam simultaneamente menores que $w(R_i)/2$, então a i-ésima iteração do algoritmo incrementa o valor da resposta por $c_i := \max\{w(C_i^S), w(C_i^T)\} \geq w(R_i)/2$. Além disso, como cada aresta tem exatamente um responsável, segue-se que $\sum_{i=1}^{n} w(R_i) = W_{tot}$, onde $W_{tot} := w(E)$ é o peso total do grafo. Ao final do algoritmo, lembrando que o valor do corte é definido como $\|S\| := w(\partial(S))$, tem-se que

$$\|S\| = \sum_{i=1}^{n} c_i \geq \sum_{i=1}^{n} \frac{w(R_i)}{2} = \frac{1}{2} W_{tot} \geq \frac{1}{2} \mathrm{OPT}.$$

[3] Demontração adaptada das notas de Anupam Gupta, redigidas por Haowen Chan. Disponível em: cs.cmu.edu/afs/cs/academic/class/15854-f05/www/scribe/lec02.pdf