

Um Algoritmo de Planos de Corte para o Problema Livre de Garra

Estudo Poliédrico e Implementação Branch-and-Cut

Felipe Crispim Fragoso



CENTRO DE INFORMÁTICA
UNIVERSIDADE FEDERAL DA PARAÍBA

João Pessoa, 2017

Felipe Crispim Fragoso

Um Algoritmo de Planos de Corte para o Problema Livre de Garra

Monografia apresentada ao curso Ciência da Computação
do Centro de Informática, da Universidade Federal da Paraíba,
para a obtenção do grau de Bacharel em Ciência da Computação

Orientador: Dr. Gilberto Farias

João Pessoa, 2017

Ficha Catalográfica elaborada por
Rogério Ferreira Marques CRB15/690

F811a Fragoso, Felipe Crispim.
Um algoritmo de planos de corte para o problema livre de garra: estudo poliédrico e implementação Branch-and-Cut / Felipe Crispim Fragoso. – João Pessoa, 2017.
42p. : il.

Monografia (Bacharelado em Ciência da Computação) – Universidade Federal da Paraíba - UFPB.
Orientador: Prof. Dr. Gilberto Farias de Sousa Filho.

1. Algoritmo. 2. Problema livre de garra. 3. Plano de cortes. 4. Combinatória poliédrica. I. Título.

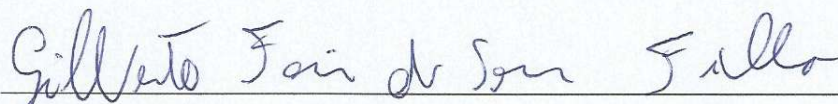
UFPB/BSCI

CDU: 004.021 (043.2)



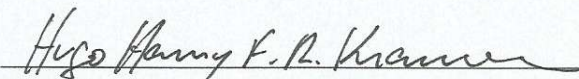
CENTRO DE INFORMÁTICA
UNIVERSIDADE FEDERAL DA PARAÍBA

Trabalho de Conclusão de Curso de Ciência da Computação intitulado *Um Algoritmo de Planos de Corte para o Problema Livre de Garra* de autoria de Felipe Crispim Fragoso, aprovada pela banca examinadora constituída pelos seguintes professores:



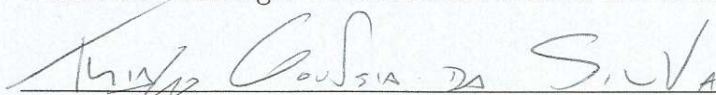
Prof. Dr. Gilberto Farias de Sousa Filho

Centro de Informática - Universidade Federal da Paraíba (UFPB)



Prof. Dr. Hugo Harry Frederico Ribeiro Kramer

Centro de Tecnologia - Universidade Federal da Paraíba (UFPB)



Prof. Msc. Thiago Gouveia da Silva

Instituto Federal de Educação, Ciência e Tecnologia da Paraíba (IFPB)



Coordenador(a) do Departamento Departamento de Ciência da Computação

Daniela Coelho Batista Guedes Pereira

CI/UFPB

João Pessoa, 13 de junho de 2017

**** Ao Infinito..... E além! ****
(Buzz Lightyear)

DEDICATÓRIA

Dedico o trabalho unicamente ao meu pai Urbano Feitoza Fragoso por sua extrema valorização ao estudo e constante contribuição moral na minha formação.

AGRADECIMENTOS

Quero agradecer, em primeiro lugar, a Deus, que se mostrou criador, que foi criativo. Seu fôlego de vida em mim me foi sustento e me deu coragem para questionar realidades e buscar sempre um novo mundo de possibilidades.

A minha namorada, companheira e melhor amiga Hozana por ter dividido comigo tantos dramas e desafios. Ela que pacientemente me ajudou, me entendeu e me suportou ao longo desses anos.

A toda a família por ter sido meu refúgio e morada nas horas difíceis e, além disso, por todo incentivo fornecido.

Agradeço também a todos os professores que me acompanharam durante a graduação, em especial aos professores Anand Subramanian e Gilberto Farias, pelos seus conselhos e por me apresentarem a programação linear.

Aos bons amigos e colegas de curso que encontrei durante o caminho: Carlos Magno, Felipe Tiago, Guguinha, Mayrton Dias (vulgo Mardson ou Maylton), Vitor Soares e tantos outros, pelos diversos momentos de conversas e brincadeiras ao longo da vida acadêmica.

RESUMO

Seja $G = (V, E)$ um grafo, no qual V é o conjunto de vértices e E o conjunto de arestas. Um grafo garra é definido como sendo um grafo bipartido completo $K_{1,3}$. O Problema Livre de Garra (PLG) tem como objetivo encontrar um subconjunto mínimo de vértices $S \subset V$ de modo que nenhum subgrafo induzido por vértice em $G[V \setminus S]$ seja um grafo garra. O presente trabalho realiza um estudo poliedral para o PLG , que é um problema NP-completo, apresentando dois modelos de programação linear inteira (\mathcal{F}_g e \mathcal{F}_{S_k}), sendo o último implementado por meio de um procedimento baseado em planos de corte. Experimentos computacionais foram realizados em instâncias com diversas densidades e contendo até 100 vértices. Os resultados obtidos sugerem que \mathcal{F}_{S_k} teve desempenho superior quando comparado a \mathcal{F}_g .

Palavras-chave: <Problema Livre de Garra>, <Plano de cortes>, <Combinatória poliédrica>.

ABSTRACT

Let $G = (V, E)$ be a simple graph, where V is the set of vertices and E is the set of edges. A claw is defined as a complete bipartite graph $K_{1,3}$. The Claw-free problem (CFP) aims at finding a minimum subset of vertices $S \subset V$ such that none of the vertex-induced subgraphs of $G[V \setminus S]$ are claws. The present work performs a polyhedral study on the CFP, which is a NP-complete problem, presenting 2 integer programming models (\mathcal{F}_g and \mathcal{F}_{S_k}), where \mathcal{F}_{S_k} is implemented with a cutting plane-based procedure. Computational experiments were performed in instances with different densities with up to 100 vertices. The results obtained suggest that \mathcal{F}_{S_k} had a superior performance when compared to \mathcal{F}_g .

Key-words: <Claw-free problem>, <Cutting planes>, <Polyhedral combinatorics>.

LISTA DE FIGURAS

| | | |
|----|--|----|
| 1 | Exemplo de conjunto não convexo | 20 |
| 2 | Exemplo de envoltória convexa | 21 |
| 3 | Exemplo de Matriz | 21 |
| 4 | Exemplo de Grafo | 24 |
| 5 | Exemplo de Lista de Adjacências | 25 |
| 6 | Exemplo de Matriz de Adjacências | 25 |
| 7 | Grafos Complementares entre si | 26 |
| 8 | Exemplos de Subgrafos | 26 |
| 9 | Exemplo de Grafo Bipartido Completo $K_{5,3}$ | 26 |
| 10 | Exemplos de Cliques | 27 |
| 11 | Conjunto Independente e Clique | 28 |
| 12 | Lista de faces de uma pirâmide | 29 |
| 13 | (a) Subgrafo sem garra A . (b) Vetor incidente χ^A | 31 |
| 14 | Garra $\{a, b, c, d\}$ | 32 |
| 15 | Topologia para o Grafo Estrela. | 34 |
| 16 | Subgrafos $L_{a,b}$, $L_{a,d}$ e I_k do Grafo Estrela S_k | 35 |
| 17 | (a) Grafo de entrada $G = (V, E)$. (b) Subgrafo de G induzido pelos vértices vizinhos de v ($N(v) = adj(v)$). (c) Grafos Estrela compostos por 2 conjuntos independentes de $G[adj(v)]$ e o vértice v | 36 |

LISTA DE TABELAS

| | | |
|---|---|----|
| 1 | Resultados computacionais para instâncias com $\ V\ = 50$ | 38 |
| 2 | Resultados computacionais para instâncias com $\ V\ = 100$ | 39 |

LISTA DE ABREVIATURAS

PL – Programação Linear

PLI – Programação Linear Inteira

PLG – Problema Livre de Garras

B&C – Branch-and-Cut

B&B – Branch-and-Bound

NPC – NP-Completo ou NP-Completa

Sumário

| | | |
|----------|---|-----------|
| 1 | INTRODUÇÃO | 17 |
| 1.1 | Delimitação e Definição do Problema de Pesquisa | 17 |
| 1.2 | Objetivos | 17 |
| 1.2.1 | Objetivo Geral | 17 |
| 1.2.2 | Objetivos Específicos | 17 |
| 1.3 | Justificativa | 18 |
| 1.4 | Estrutura da monografia | 18 |
| 2 | FUNDAMENTAÇÃO TEÓRICA | 19 |
| 2.1 | Álgebra Linear | 19 |
| 2.1.1 | Notação Lógica e Matemática | 19 |
| 2.1.2 | Pontos e vetores | 19 |
| 2.1.3 | Envoltória convexa | 20 |
| 2.1.4 | Matrizes | 21 |
| 2.2 | Classes de Problemas P/NP | 22 |
| 2.2.1 | Notação Assintótica e Definição de um problema | 22 |
| 2.2.2 | Linguagens Regulares | 23 |
| 2.2.3 | Classe P | 23 |
| 2.2.4 | Classe NP | 23 |
| 2.2.5 | Classe NP-Completa | 23 |
| 2.3 | Teoria dos Grafos | 24 |
| 2.3.1 | Grafo | 24 |
| 2.3.2 | Subgrafo | 26 |
| 2.3.3 | Grafo Bipartido | 26 |
| 2.3.4 | Clique e Conjunto Independente de vértices | 27 |
| 2.4 | Otimização Combinatória | 28 |
| 2.4.1 | Combinatória Poliédrica | 28 |
| 2.4.2 | Programação Linear e Programação Linear Inteira | 29 |

| | | |
|----------|--|-----------|
| 3 | METODOLOGIA | 31 |
| 3.1 | O Politopo Sem Garra | 31 |
| 3.2 | Conjunto de Inequações Triviais | 31 |
| 3.3 | Conjunto de Inequações Estrela S_k | 33 |
| 3.4 | Planos de Corte para o (\mathcal{F}_{S_k}) | 36 |
| 4 | RESULTADOS COMPUTACIONAIS | 38 |
| 5 | CONCLUSÕES E TRABALHOS FUTUROS | 40 |

1 INTRODUÇÃO

1.1 Delimitação e Definição do Problema de Pesquisa

Seja $G(V, E)$ um grafo, no qual V é o conjunto de vértices e E o conjunto de arestas. Um grafo garra é definido como sendo um grafo bipartido completo $K_{1,3}$. O Problema Livre de Garra (PLG) tem como objetivo encontrar um subconjunto mínimo de vértices $S \subset V$ de modo que nenhum subgrafo induzido por vértice em $G[V \setminus S]$ seja um grafo garra.

Diversos trabalhos foram propostos para grafos livre de garra. A descrição do politopo *set packing* para grafos sem garra se assemelha ao politopo de emparelhamento [2] que é mais simples. Já [14] propõe um algoritmo polinomial (programação dinâmica) para o *set packing* em grafos sem garra. Em [15] são apresentados estudos sobre a relação entre caminhos hamiltonianos e grafos sem garra, propondo um algoritmo polinomial para o problema de caminhos independentes ponderados.

A remoção de garras de um grafo está relacionado ao problema de transformar grafos de intervalo em grafos unitários. Esse problema foi proposto por [7] há mais de trinta anos, não tendo recebido atenção da literatura desde então. Também é conhecido que grafos unitários são equivalentes a grafos de indiferença [16]. Muitos trabalhos da literatura apresentam a importância dos grafos de indiferença para a área de otimização. Em [13] há descrições de algoritmos gulosos ótimos para os problemas de coloração, caminho mínimo entre dois vértices, emparelhamento máximo e caminho hamiltoniano sobre grafos de indiferença.

Dada a importância do tema, este trabalho propõe modelos de Programação Linear Inteira (PLI) para o PLG.

1.2 Objetivos

Através do que foi apresentado anteriormente, o objetivo geral e os objetivos específicos do estudo são descritos a seguir.

1.2.1 Objetivo Geral

Encontrar modelos PLIs eficientes para o PLG.

1.2.2 Objetivos Específicos

- a) Estudo poliédrico do problema.

- b) Construção de modelos PLIs para resolvê-lo.
- c) Análise da eficiência dos modelos propostos.

1.3 Justificativa

Para [12], a família de problemas de remoção de vértices em grafos, consiste em: dada uma propriedade de grafo π , qual o número mínimo de vértices a serem removidos de um dado grafo tal que o subgrafo resultante satisfaça π ? O autor demonstrou que se π é uma propriedade trivial (existem infinitos grafos que obedecem ou desobedecem π) e hereditária (Qualquer subgrafo induzido de um grafo que obedece π também obedece a π), então o problema de remoção de vértices para π é NP-Completo. Como um grafo sem garra é uma propriedade não trivial e hereditária para subgrafos induzidos, logo, o PLG, que é um problema de remoção de vértices, é NP-Completo. Dessa forma, faz-se necessário encontrar soluções eficientes para o problema.

A escolha no uso de PLI foi feita porque métodos para resolução de PLIs são amplamente difundidos e apresentam diversas implementações em software.

1.4 Estrutura da monografia

O restante do trabalho foi dividido em 4 capítulos. O capítulo 2 apresenta os conceitos necessários para elaboração deste trabalho. O capítulo 3 detalha os passos para demonstração e implementação dos modelos PLI propostos. O capítulo 4 apresenta os resultados computacionais obtidos. O capítulo 5 explicita as conclusões e os trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

2.1 Álgebra Linear

Neste trabalho, denota-se por \mathbb{R} o conjunto dos números reais.

2.1.1 Notação Lógica e Matemática

Um conjunto é um agrupamento de elementos. A cardinalidade de um conjunto A é denotada como $\|A\|$. Para descrever a soma dos elementos $\{a^1, a^2, \dots, a^n\}$ do conjunto A , usa-se $\sum_{i=1}^n a^i$. De forma equivalente, utiliza-se $\prod_{i=1}^n a^i$ para expressar a multiplicação dos mesmos elementos.

Denota-se como $x \in A$ um elemento x pertencente ao conjunto A . Caso contrário, utiliza-se a notação $x \notin A$.

Se A e B são 2 conjuntos, A é subconjunto de B se, para todo $x \in A$, tem-se que $x \in B$, denota-se que A é subconjunto de B como $A \subset B$. Uma função que representa uma relação de correspondência entre A e B é escrita como $f : A \rightarrow B$.

Sejam A e B dois conjuntos tais que $B \subset A$. O conjunto de elementos que estão em A e não estão em B é denotado por $A \setminus B$.

O conjunto C de elementos que, simultaneamente, pertencem aos conjuntos A e B é escrito por $C = A \cap B$. Quando deseja-se o conjunto C dos elementos que estão em A ou B , denota-se $C = A \cup B$.

Para citar todos os elementos x de um conjunto A usa-se $\forall x \in A$. Para denotar a existência de um elemento x qualquer utiliza-se $\exists x$. Quando deseja-se dizer que um conjunto A é formado por qualquer elemento x que é verdadeiro para uma determinada propriedade, denota-se $A = \{x \mid \text{propriedade verdadeira para } x\}$.

Quando um conjunto A é igual ao conjunto B , escreve-se $A = B$. Caso contrário, escreve-se $A \neq B$.

2.1.2 Pontos e vetores

Um ponto pode ser definido como um conjunto de números, usualmente escritos em linha ou coluna. Pontos podem ser visualizados em espaços dimensionais, onde os números representam as coordenadas de cada eixo do espaço. E também, podem ser entendidos como vetores, que são linhas que tem origem em um ponto predefinido e terminam no ponto descrito. Quando um vetor ou ponto possui n coordenadas, pode-se dizer que ele está em \mathbb{R}^n .

A soma de dois vetores do \mathbb{R}^n , escreve-se $x + y$, é efetuada ao somar-se os valores das coordenadas correspondentes em x e y . Note que a operação soma, representada pela equação (1) abaixo, gera outro vetor z também em \mathbb{R}^n .

$$z_i = x_i + y_i \quad \forall i \in 1 \dots n \quad (1)$$

Uma coleção de vetores $x^1, x^2, x^3, \dots, x^k \in \mathbb{R}^n$ é dita *afim-independente* se

$$\sum_{i=1}^k \lambda_i x^i = 0 \text{ e } \sum_{i=1}^k \lambda_i = 0$$

tem como solução única $\lambda_i = 0, \forall i = 1, 2, \dots, k$.

O *produto escalar*, neste trabalho denotado como $x^T y$, é uma função binária definida entre dois vetores que fornece um número real como resultado. O produto escalar pode ser calculado da seguinte forma:

$$x^T y = \sum_{i=1}^n x_i y_i = \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix}^T (y_1 \dots y_n)$$

2.1.3 Envoltória convexa

Um conjunto $S \subset \mathbb{R}^n$ é convexo se para qualquer $x, y \in S$ e qualquer λ é um valor no intervalo $[0, 1]$ tem-se $\lambda x + (1 - \lambda)y \in S$. Ou seja, dados dois pontos $x, y \in S$ temos que qualquer ponto que reside entre x e y também se encontra dentro de S . A figura abaixo ilustra um exemplo de conjunto não convexo.

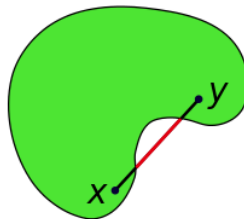


Figura 1: Exemplo de conjunto não convexo

A *envoltória convexa* de um conjunto $S \subset \mathbb{R}^n$ é o menor conjunto convexo formado pelos pontos de S que contém todos os pontos em S . A figura 2 mostra uma envoltória de um conjunto de pontos (delineada em preto). Ainda na figura 2, temos que a envoltória convexa (delineada em azul) é a menor envoltória que engloba todos os pontos.

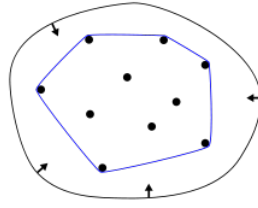


Figura 2: Exemplo de envoltória convexa

2.1.4 Matrizes

Uma matriz $A \in \mathbb{R}^{n \times m}$, representada na figura 3, é uma tabela de n linhas e m colunas de símbolos representadas em forma de quadro. É possível representar vetores como linhas ou colunas de uma matriz. Os elementos de uma matriz são indexados por linha e coluna por números inteiros.

$$A = \begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ a_{3,1} & a_{3,2} & \cdots & a_{3,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m,1} & a_{m,2} & \cdots & a_{m,n} \end{bmatrix}$$

Figura 3: Exemplo de Matriz

Sejam $A \in \mathbb{R}^{n_1 \times m_1}$ e $B \in \mathbb{R}^{n_2 \times m_2}$ duas matrizes. O produto matricial $AB \in \mathbb{R}^{n_1 \times m_2}$, quando $m_1 = n_2$, é realizado da seguinte forma:

$$AB_{ij} = \sum_{k=1}^{m_1} A_{ik} B_{kj} \quad \forall i = \{1, 2, \dots, n_1\} \quad \forall j = \{1, 2, \dots, m_2\}$$

Também é possível representar inequações em forma de produto matricial $Ax \leq b$. Onde, cada linha matriz A corresponde aos coeficientes das variáveis de cada inequação que, por sua vez, são representadas pelo vetor coluna x e b é um vetor coluna que representa os valores obedecidos por cada inequação. No sistema abaixo, podemos ver uma transformação de um conjunto de inequações em uma inequação com produto matricial.

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \cdots + a_{1n}x_n &\leq b_1 \\ a_{21}x_1 + a_{22}x_2 + a_{23}x_3 + \cdots + a_{2n}x_n &\leq b_2 \\ a_{31}x_1 + a_{32}x_2 + a_{33}x_3 + \cdots + a_{3n}x_n &\leq b_3 \end{aligned} \Rightarrow \begin{bmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \cdots & a_{2n} \\ a_{31} & a_{32} & a_{33} & \cdots & a_{3n} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix} \leq \begin{bmatrix} b_1 & b_2 & b_3 \end{bmatrix}$$

2.2 Classes de Problemas P/NP

2.2.1 Notação Assintótica e Definição de um problema

As *notações assintóticas* são formas de descrever o tempo de execução de um algoritmo computacional por meio de funções do tamanho das suas entradas [4]. Dessa forma, torna-se possível estimar o tempo de execução de um algoritmo para diferentes tamanhos de entradas.

Em geral, utiliza-se a notação $f(n) = O(g(n))$ para indicar que o algoritmo $f(n)$ é um membro do conjunto de tempos de execução $O(g(n))$. Dizer que $f(n) = O(g(n))$ significa que existe uma constante c , tal que, $f(n) \leq cg(n) \forall n \in \mathbb{R}$. Usando a notação O , podemos descrever o tempo de execução de um algoritmo observando a sua estrutura. Através de observação, a comunidade científica pode perceber que existem um conjunto de funções $g(n)$, mostradas abaixo, que são comuns a diversos algoritmos.

- a) Polinômios : $g(n) = O(p(n^d))$, onde $p(n^d) = \sum_{i=0}^d a_i n^i$ e $a_d \neq 0$.
- b) Exponencial: $g(n) = O(d^n)$, onde n é variável.
- c) Logaritmo: $g(n) = O(\log_b(n))$, onde $\log_b(n)$ é uma função logaritmo (Vide [4] para mais detalhes).
- d) Fatorial: $g(n) = O(n!)$, onde $n! = \prod_{x=1}^n x$.
- e) Constante: $g(n) = O(k)$, onde $k \in \mathbb{R}$.

A fim de entender as classificações de complexidade de um problema, é necessário formalizar o significado do que seja um problema. Um *problema abstrato* é uma relação binária que leva um conjunto de instâncias I para o problema à um conjunto de soluções S do problema.

Para um programa de computador resolver um problema abstrato Q , as instâncias de Q devem ser representadas por uma codificação binária. Será denominado *problema concreto* a codificação binária do problema abstrato Q .

Um problema abstrato Q é classificado como *problema de decisão* quando a solução para qualquer instância de Q é “sim” ou “não”.

Vale ressaltar que a categorização de problemas nas classes P (vide seção 2.2.3), NP (seção 2.2.4) ou NPC (seção 2.2.5) se aplica apenas a versão de decisão de um *problema concreto*.

2.2.2 Linguagens Regulares

Um *alfabeto* Σ é um conjunto finito de símbolos. Uma *linguagem regular* L sobre Σ é qualquer cadeia formada pelos símbolos de Σ .

Problemas concretos de decisão podem ser representados por linguagens regulares. Pois, seu conjunto de instâncias e soluções podem ser representados por uma linguagem sobre o alfabeto $\{0,1\}$.

Seja A um algoritmo. Uma cadeia x qualquer é dita “aceita por A ” se $A(x) = 1$. Caso $A(x) = 0$, a cadeia é dita “rejeitada por A ”. Além disso, a linguagem L é aceita por A se $\forall x \in L$ implica em $A(x) = 1$.

2.2.3 Classe P

Segundo [4], a classe de problemas P é o conjunto das linguagens L aceitas por um algoritmo A de função assintótica polinomial. Ou seja,

$$P = \{L \mid L \text{ é aceita por } A \text{ em tempo polinomial}\}$$

2.2.4 Classe NP

Há também *algoritmos de verificação* A , que são algoritmos que possuem a sua cadeia de entrada comum x e um *certificado* binário y . Esse tipo de algoritmo verifica se existe algum certificado y para a cadeia de entrada x , tal que, $A(x, y) = 1$. Ou seja, A é um algoritmo capaz de verificar se uma solução de um problema concreto é aceita.

A classe de complexidade NP é a classe de linguagens que possuem um algoritmo de verificação de tempo polinomial. É possível concluir que se uma linguagem L está em P , então L também está em NP , porque se existe um algoritmo que é capaz de aceitar L em tempo polinomial, também existem um algoritmo de verificação para L em tempo polinomial. Para isto, é necessário apenas descartar o certificado e usar a resposta do algoritmo de aceitação como resultado do algoritmo de verificação.

Foi apresentado em [3] o problema de P versus NP , que permanece em aberto. Esse problema consiste em provar se $P = NP$ ou $P \neq NP$.

2.2.5 Classe NP-Completa

Uma linguagem L_1 é redutível para L_2 , escrita $L_1 \leq_P L_2$, se existe alguma função calculável $f(x)$ que recebe a instância de L_1 como parâmetro e gera uma instância de L_2

em tempo polinomial. Denomina-se $f(x)$ como *função de redução*, e um algoritmo que calcula $f(x)$ em tempo polinomial como *algoritmo de redução*.

Uma linguagem L é *NP-Completa* (NPC) se:

1. $L \in NP$
2. $\forall L' \in NP, L' \leq_P L$ em tempo polinomial

A classe de problemas *NPC* é composta por problemas que tem complexidade de tempo assintótica exponencial ou fatorial. Isso significa que são problemas computacionais difíceis de resolver. O mecanismo de redução permite que seja possível identificar se um problema X qualquer é *NP-Completo* ao reduzi-lo para outro problema que ja foi provado estar em *NPC*.

2.3 Teoria dos Grafos

2.3.1 Grafo

Um *grafo* $G(V, E)$ (Vide figura 4 para um exemplo) é uma estrutura de dados. Onde, V representa o conjunto de vértices do grafo e E o conjunto de arestas. Ou seja, V e E são atributos do grafo [4]. Grafos podem ser representados computacionalmente através de listas de adjacências ou matriz de adjacências.

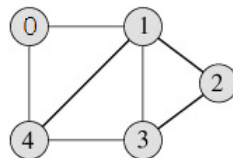


Figura 4: Exemplo de Grafo

A representação por *lista de adjacências* de $G(V, E)$ é formada por um arranjo de $\|V\|$ listas, uma para cada vértice, onde a lista de um vértice $u \in V$ qualquer, contém um nó $w \in V$, se e somente se, existe uma aresta $(u, w) \in E$, isto é, existe uma aresta que possui u e w como pontos extremos. A figura abaixo ilustra uma lista adjacências do grafo apresentado na figura 4

Representação por lista de adjacências permite descobrir se uma aresta (u, w) está presente em $G(V, E)$ em $O(n)$, porque, a forma de verificar se (u, v) está em $G(V, E)$, seria obtendo a lista de adjacências de u ($adj[u]$), para então, verificar se $w \in adj[u]$ com busca linear.

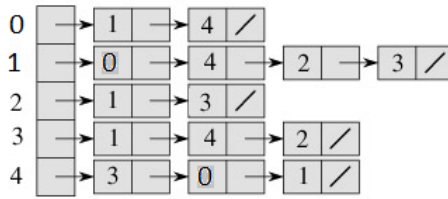


Figura 5: Exemplo de Lista de Adjacências

Essa desvantagem pode ser contornada com representação por *matriz de adjacências*. Nesse caso, cada vértice é numerado com valores $\{1, 2, 3, \dots, \|V\|\}$. Então, podemos representar $G(V, E)$ como uma matriz $A^{\|V\| \times \|V\|}$, tal que, para cada célula $A_{i,j}$:

$$A_{i,j} = \begin{cases} 0, & \text{se } (i, j) \notin E \\ 1, & \text{se } (i, j) \in E \end{cases}$$

Nesse caso, podemos descobrir se uma aresta (u, w) está presente em $G(V, E)$ em $O(1)$, porque é necessário apenas consultar a linha u e coluna v da matriz de adjacências. A figura 6 mostra a representação do grafo da figura 4 como matriz de adjacências.

| | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| 0 | 0 | 1 | 0 | 0 | 1 |
| 1 | 1 | 0 | 1 | 1 | 1 |
| 2 | 0 | 1 | 0 | 1 | 0 |
| 3 | 0 | 1 | 1 | 0 | 1 |
| 4 | 1 | 1 | 0 | 1 | 0 |

Figura 6: Exemplo de Matriz de Adjacências

Também é possível obter um *grafo complementar* $\overline{G(V, E)}$ de um grafo $G(V, E)$, representado na figura 7, da seguinte forma:

Algoritmo 1: ALGORITMO PARA ENCONTRAR GRAFO COMPLEMENTAR

Entrada: $G(V, E)$

Saída: $\overline{G(V, E)}$

1 **início**

2 Cria-se um grafo $\overline{G(V, E')}$ com mesma quantidade de vértices que $G(V, E)$
 e $E' = \emptyset$

3 **para** cada par de vértices $(u, w) \in V$ **faça**

4 **se** aresta $(u, w) \notin E$ **então**

5 adiciona (u, w) em E'

6 **fim**

7 **fim**

8 **retorna** $\overline{G(V, E')}$

9 **fim**

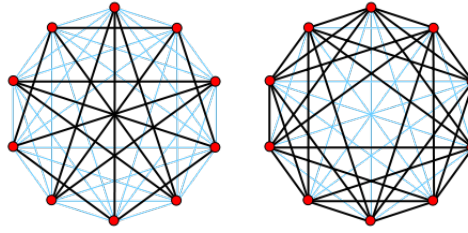
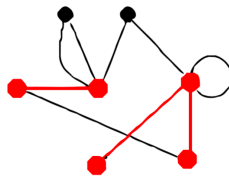


Figura 7: Grafos Complementares entre si

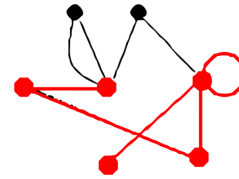
As linhas pretas representam arestas reais. Linhas azuis representam arestas complementares.

2.3.2 Subgrafo

Um *subgrafo* $G'(V', E')$ de um grafo $G(V, E)$ é um grafo (Vide a porção em vermelho da figura 8a), em que, $V' \subset V$ e $E' \subset E$. Um subgrafo $G'(V', E')$ de $G(V, E)$ é chamado de *subgrafo induzido* por vértice (Porção em vermelho da Figura 8b), denotado por $G[V']$, se toda aresta $(u, w) \in E$, onde $u \in V'$ e $w \in V'$, também é uma aresta em E' . Note que na figura 8b todas as arestas que ligam qualquer par de vértices estão presentes no subgrafo em vermelho.



(a) Um subgrafo em vermelho



(b) Um Subgrafo Induzido por Vértice em vermelho

Figura 8: Exemplos de Subgrafos

2.3.3 Grafo Bipartido

Um grafo $G(V, E)$ é classificado como *grafo bipartido* quando é possível dividi-lo em dois conjuntos de vértices X e Y , tal que, para qualquer aresta $(u, w) \in E$ temos que $u \in X$ e $w \in Y$. Ou seja, nenhum vértice do conjunto X é adjacente a um vértice do conjunto Y e vice-versa. Além disso, grafos bipartidos podem ser *completos*, isto é, para cada $u \in X$ existe uma aresta para $w \in Y$. Denota-se um *grafo bipartido completo* como $K_{\|X\|, \|Y\|}$.

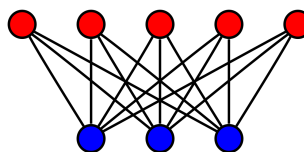


Figura 9: Exemplo de Grafo Bipartido Completo $K_{5,3}$

2.3.4 Clique e Conjunto Independente de vértices

Um grafo $G(V, E)$ é *completo* quando todo vértice $u \in V$ é adjacente a todos os vértices do conjunto $V \setminus \{u\}$. Uma *clique* C é um subgrafo $G'(V', E')$ de $G(V, E)$ que é completo.

Cliques maximal é o conjunto de subgrafos $G'(V', E')$ de $G(V, E)$ que não podem ser estendidos através da inclusão de um novo vértice $u \in G(V, E)$.

Cliques máximo é o conjunto de subgrafos $G'(V', E')$ de $G(V, E)$ que têm o maior tamanho $\|V'\|$ possível.

Na figura 10 pode-se visualizar, em vermelho, o clique máximo e maximal de um grafo. A figura 10a apresenta um exemplo de clique não maximal, porque a clique ainda pode ser estendida por um vértice para formar o clique máximo e maximal descrito na figura 10b. É apresentado na figura 10c, um exemplo de clique maximal e não máximo.

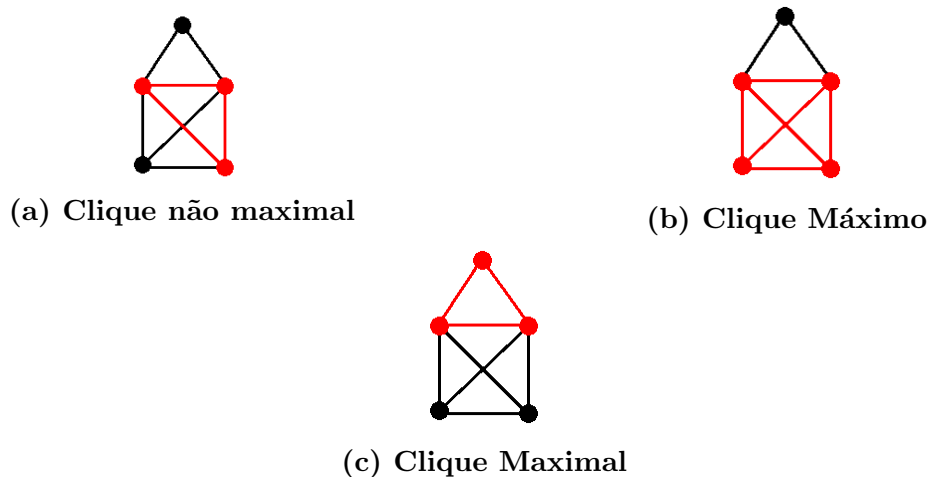


Figura 10: Exemplos de Cliques

O *conjunto independente* de vértices I de um grafo $G(V, E)$ é um conjunto em que nenhum par de vértices $(u, w) \in I$ são adjacentes. Ou seja, $(u, w) \notin E$. Assim como *cliques*, os conjuntos de vértices independentes podem ser *maximal* ou *máximo*. O conjunto máximo é o maior conjunto I em $G(V, E)$, e por sua vez, o conjunto maximal um conjunto independente de vértices que não pode ser estendido.

É possível deduzir que os conjuntos independente máximo e maximal de um grafo $G(V, E)$ podem ser obtidos ao encontrar-se, respectivamente, as cliques maximas e maximais no grafo complementar $\overline{G(V, E)}$. A figura 11 mostra como encontrar um conjunto independente de um grafo $G(V, E)$ (Figura 11a) ao encontrar-se um clique no grafo $\overline{(V, E)}$ (Figura 11b).



Figura 11: Conjunto Independente e Clique

2.4 Otimização Combinatória

2.4.1 Combinatória Poliédrica

Um *poliedro* P pode ser definido como um subconjunto $P \subset \mathbb{R}^n$ em que

$$P = \{x \in \mathbb{R}^n \mid Ax \leq b\}$$

para uma matriz $A \in \mathbb{R}^{n \times m}$ e um vetor $b \in \mathbb{R}^m$. A *Combinatória poliédrica* estuda propriedades de poliedros que são formados por problemas combinatórios. Segundo [6], esses estudos podem levar ao desenvolvimento de algoritmos ou revelar propriedades do problema estudado.

Se P é um poliedro que obedece a sentença

$$\exists \alpha \in \mathbb{R}, \forall x \in P : \|x\| \leq \alpha$$

Então, P é dito ser um *politopo*.

Um atributo importante de um poliedro é a sua *dimensão*, denotada por $\dim(P)$, que tem valor d se P contém $d+1$ vetores afim-independentes. Diz-se que P tem *dimensão cheia* se $P \subset \mathbb{R}^n$ é um poliedro onde $\dim(P) = n$.

Seja F um subconjunto de um poliedro P . F é uma *face* de P se

$$F = P \cap \{x \mid c^T x = d\}$$

onde $c^T x \leq d$ é uma *inequação válida* para P . Ou seja, $c^T x \leq d$ satisfaz todos os pontos x em P . Uma face F de um poliedro P é chamada de *faceta* se $\dim(F) = \dim(P) - 1$. Um poliedro pode ser completamente descrito através do seu conjunto de inequações que são *definidoras de facetas*. Se P tem dimensão cheia, então P pode ser representado, de forma única, através de uma inequação associada a cada faceta de P .

A figura 12 lista todas as faces de um politopo em forma de pirâmide. A primeira

linha $(abcde)$ lista os pontos que formam a face de dimensão 3, ou seja, a própria pirâmide é face de si mesma. A segunda linha mostra as facetas (faces de dimensão 2), que são os planos formados por todos os conjuntos três pontos. A terceira linha representa as cristas da pirâmide (faces de dimensão 1) formados pelas arestas que ligam dois pontos. A quarta linha apresenta os cumes (faces de dimensão 0) da pirâmide, que são seus vértices. E a ultima linha apresenta a face vazia (dimensão -1).

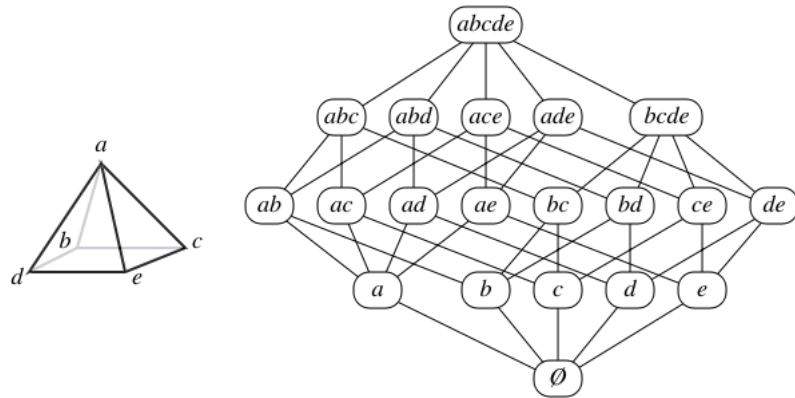


Figura 12: Lista de faces de uma pirâmide

Em poliedros de dimensão cheia, sabe-se que, se F é uma faceta, então o conjunto de todas as representações possíveis de F é obtido ao tomarmos múltiplos escalares de F . Essa observação é importante para demonstrar se uma inequação qualquer, válida para um poliedro de dimensão cheia P , é uma inequação definidora de faceta de P . Para isto, basta demonstrar que a inequação conhecida é múltiplo de uma suposta inequação definidora de faceta.

2.4.2 Programação Linear e Programação Linear Inteira

Programação linear (PL) é um paradigma que busca minimizar ou maximizar uma função linear $f : \mathbb{R}^n \rightarrow \mathbb{R}$ sobre um conjunto finito L de \mathbb{R}^n [10].

Através do algoritmo *Simplex*, é possível resolver um PL descrito por meio de inequações lineares. Para isto, basta descrever a envoltória convexa de L como um conjunto de inequações lineares que são definidoras de facetas desse politopo e resolver o programa linear:

$$\text{minimizar/maximizar } f(x) \quad \text{sujeito a } Ax \leq b$$

Onde, Ax representa a matriz de inequações para o politopo de L e $f(x)$ representa uma função linear a ser maximizada ou minimizada.

Existem problemas que exigem que uma ou mais variáveis em x sejam inteiras. Sendo assim, surge a *programação linear inteira* (PLI). A primeira solução para problemas

PLI foi proposta por [9], essa solução consiste em *planos de corte* que são adicionados ao modelo linear de forma que cortem soluções fracionárias sem eliminar soluções inteiras válidas para o PLI.

Em 1960, foi apresentado o algoritmo *branch-and-bound* (B&B), que é capaz de resolver um PLI ao construir, através de novas restrições, novas formulações de PL que serão resolvidas pelo método Simplex com uma solução inteira.

Posteriormente, outro método denominado *Branch-and-Cut* (B&C) foi introduzido para resolver problemas de PLI. Esse método resolve uma formulação PLI adicionando cortes, que não excluem soluções inteiras válidas, ao modelo PLI. Além disso, ao mesmo tempo, o B&C executa um algoritmo B&B.

Para mais detalhes sobre os métodos de Planos de corte, B&B e B&C consulte [11] e [10].

3 METODOLOGIA

3.1 O Politopo Sem Garra

Seja $G_n(V, E)$, com $\|V\| = n$, um grafo. Para evitar trivialidades, assumamos que $n \geq 2$. Seja P_n a envoltória convexa dos vetores incidentes que representam subgrafos sem garra de G_n , isto é,

$$P_n = \text{conv}\{\chi^A \in \mathbb{R}^n \mid A \text{ é subgrafo sem garra de } G_n\}.$$

P_n é o politopo sem garra (de ordem n). A Figura 13 representa um subgrafo sem garra A e seu vetor correspondente.

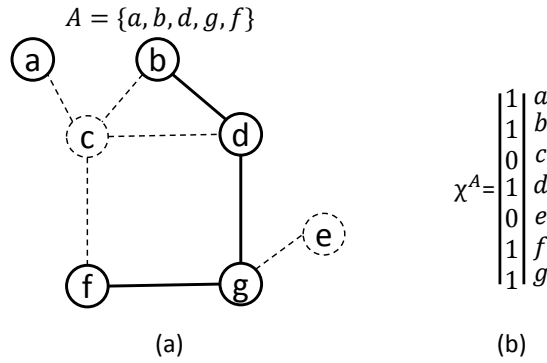


Figura 13: (a) Subgrafo sem garra A . (b) Vetor incidente χ^A

O PLG pode ser formulado como um PL da forma

$$\begin{aligned} \min \quad & \sum (1 - x_i) \\ \text{sujeito a } & x \in P_n \end{aligned}$$

uma vez que cada solução básica do PL é um vetor incidente sem garra. Entretanto, com o objetivo de aplicar o método B&C para resolver esse problema, faz-se necessário uma descrição do P_n por meio de um sistema de inequações lineares.

3.2 Conjunto de Inequações Triviais

O PLG foi, então, modelado como um PLI, pois dado que P_n está contido em um hipercubo unitário, as *inequações triviais*

$$0 \leq x_v \leq 1 \quad \forall v \in V$$

são claramente válidas e toda garra $\{a, b, c, d\}$ de G_n , ilustrado na Figura 14, é proibida pela *inequação garra*

$$(x_a + x_b + x_d) + x_c \leq 3 \tag{2}$$

que é satisfeita por todo vetor em P_n , e portanto, válida para P_n .

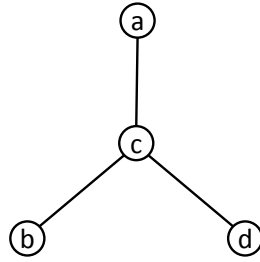


Figura 14: Garra $\{a, b, c, d\}$.

Considere agora o politopo

$$F_n = \{x \in \mathbb{R}^{\|V\|} \mid 0 \leq x_v \leq 1, \quad \forall v \in V, \\ (x_a + x_b + x_d) + x_c \leq 3 \quad \text{para toda garra } \{a, b, c, d\} \text{ de } G_n\}.$$

Das observações anteriores decorre que $P_n \subseteq F_n$, e é possível ver que os pontos inteiros de F_n são exatamente os vetores de incidência dos subgrafos sem garras de G_n . Então, $P_n = \text{conv}\{x \in F_n \mid x \text{ é inteiro}\}$ e isso implica que

$$(\mathcal{F}_g) \min \sum (1 - x_i) \\ \text{sujeito a } 0 \leq x_v \leq 1, \quad \forall v \in V, \\ (x_a + x_b + x_d) + x_c \leq 3, \quad \text{para toda garra } \{a, b, c, d\} \text{ de } G_n, \\ x \text{ é inteiro,}$$

é uma formulação PLI do PLG. Observe que P_n contém o vetor nulo e todos os vetores unitários, somando $n + 1$ pontos afim independentes; então, P_n tem dimensão cheia, isto é,

$$\dim P_n = \|V\| = n.$$

Isso implica que para toda faceta de P_n existe uma única inequação que a define.

Teorema 3.1. *Para todo politopo sem garras P_n , $n \geq 2$, podemos afirmar que:*

- a) *Toda restrição de não negatividade $x_v \geq 0$ define uma faceta de P_n ;*
- b) *Toda inequação de limite superior $x_v \leq 1$ define uma faceta de P_n ;*
- c) *Toda inequação garra $(x_a + x_b + x_d) + x_c \leq 3$, com vértice central único, define uma faceta de P_n .*

Demonstração. (a) Seja $v \in V$. Então, $x_v = 0$ é satisfeita pelo vetor nulo e todos os vetores unitários $\chi^{\{u\}}$, $u \in V$, $u \neq v$. Estes $\|V\|$ vetores são pertencentes a P_n e são afim independentes.

(b) Seja $v \in V$. Logo, $x_v = 1$ é satisfeito pelo vetor unitário $\chi^{\{v\}}$ e os vetores $\chi^{\{v+u\}}$, $\forall u \in V$, $u \neq v$. Estes $\|V\|$ vetores são pertencentes a P_n e são afim independentes.

(c) Seja $\{a, b, c, d\}$ uma garra com vértice central c único em G_n , isto é, não há outro vértice central diferente de c para $\{a, b, d\}$. Seja também $a^T x \leq a_0$ a inequação (2), isto é, $a^T x = (x_a + x_b + x_d) + x_c \leq 3 = a_0$, e seja $b^T x \leq b_0$ uma faceta definida para P_n tal que $F_a = \{x \in P_n \mid a^T x = a_0\} \subseteq F_b = \{x \in P_n \mid b^T x = b_0\}$. Claramente, $F_a \neq P_n$; assim, se for possível provar que $b = \alpha a$ para algum $\alpha \in \mathbb{R}$, uma vez que $F_a \neq \emptyset$, concluiria-se que, (2) define uma faceta de P_n .

Observando a Figura 14 têm-se que $\chi^{\{a,b,c\}}$, $\chi^{\{a,c,d\}}$, $\chi^{\{b,c,d\}}$, $\chi^{\{a,b,d\}} \in F_a \subseteq F_b$, e portanto, $0 = b_0 - a_0 = b^T \chi^{\{a,b,c\}} - a^T \chi^{\{a,b,c\}} = b_a - a_a$, $0 = b_0 - a_0 = b^T \chi^{\{a,b,c\}} - a^T \chi^{\{a,b,c\}} = b_c - a_c$ e $0 = b_0 - a_0 = b^T \chi^{\{a,b,c\}} - a^T \chi^{\{a,b,c\}} = b_b - a_b$; então $b_a = b_b = b_c = a_a = a_b = a_c = \alpha$.

Por fim, para qualquer $v \in (V - \{a, b, c, d\})$, temos $\chi^{\{a,b,d,v\}} \in F_a \subseteq F_b$, e portanto, $0 = b_0 - a_0 = b^T \chi^{\{a,b,d,v\}} - a^T \chi^{\{a,b,d,v\}} = b_v - a_v$. Logo, para qualquer $v \in (V - \{a, b, c, d\})$ vale que $b_v = a_v$.

Essas observações implicam que a inequação (2), associada ao grafo garra de vértice central único, define uma faceta de P_n .

□

3.3 Conjunto de Inequações Estrela S_k

Definição 3.1. *(Grafo Estrela) O Grafo Estrela S_k é um grafo bipartido completo $K_{1,k}$, para $k \geq 3$. S_k possui um vértice central c e um conjunto independente I_k .*

A Figura 15 ilustra uma topologia para o Grafo Estrela.

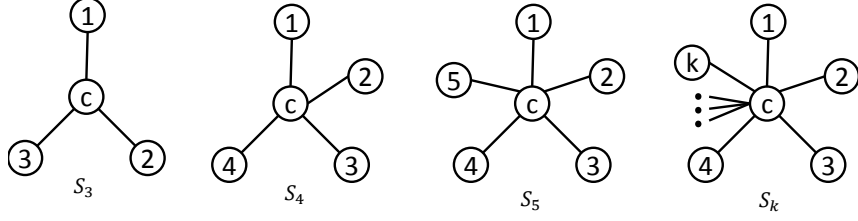


Figura 15: Topologia para o Grafo Estrela.

Teorema 3.2. *Seja $S_k \subset G_n$ o Grafo Estrela com $k \geq 3$, a correspondente inequação*

$$\sum_{v \in I_k} x_v + (k-2)x_c \leq k, \quad (3)$$

é válida para P_n e define uma faceta.

Demonstração. Seja A uma solução sem garra. Para que (3) seja válida, então, para todo grafo estrela $S_k \subset G_n$, se o vértice central c de S_k pertencer à A e não há outro vertice central c_2 para qualquer subconjunto maior ou igual que 2 de I_k , então no máximo dois vértices de I_k pertencem à A ($\sum_{v \in I_k} x_v \leq 2$), caso contrário, qualquer quantidade de vértices de I_k pode estar em A ($\sum_{v \in I_k} x_v \leq k$).

Fazendo $x_c = 1$ em (3), ou seja, vértice central $c \in A$, temos que $\sum_{v \in I_k} x_v + (k-2) \leq k$, logo, $\sum_{v \in I_k} x_v \leq 2$. Fazendo $x_c = 0$ em (3), ou seja, vértice central $c \notin A$, temos que $\sum_{v \in I_k} x_v \leq 2$. Portanto, (3) é válida para P_n .

Prova que (3) define uma faceta de P_n para estrelas com vértice central único.

Seja $a^T x \leq a_0$ a inequação (3), isto é, $a^T x = \sum_{v \in I_k} x_v + (k-2)x_c \leq k = a_0$, e seja $b^T x \leq b_0$ uma faceta definida para P_n tal que $F_a = \{x \in P_n \mid a^T x = a_0\} \subseteq F_b = \{x \in P_n \mid b^T x = b_0\}$. Claramente, $F_a \neq P_n$; assim, se for provado que $b = \alpha a$, para $\alpha \in \mathbb{R}$ uma vez que $F_a \neq \emptyset$, poderia-se concluir que (3) define uma faceta de P_n . Para isto, será estabelecido o seguinte:

Lema 3.3. *Existe $\alpha \in \mathbb{R}$ tal que $b_v = \alpha$ para todo $v \in I_k$ e $b_c = \alpha(k-2)$ sendo $c \in S_k$ o vértice central.*

Demonstração. Para provar o Lema 3.3 será utilizada a definição do conjunto $L_{i,j}$ como

$$L_{i,j} = \{\{i, j, c\} \mid i \text{ e } j \in I_k \text{ e } c \text{ vértice central de } S_k\}.$$

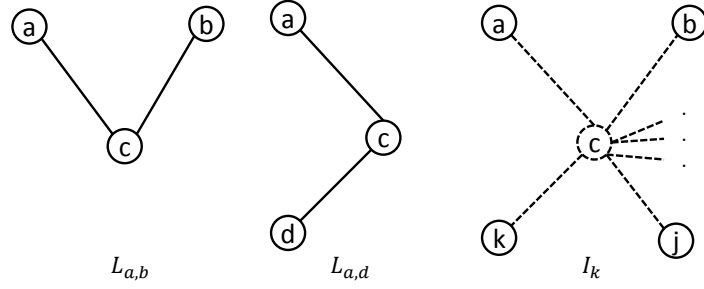


Figura 16: Subgrafos $L_{a,b}$, $L_{a,d}$ e I_k do Grafo Estrela S_k .

Observando os subgrafos $L_{a,b}$ e $L_{a,d}$ de S_k , ilustrados na Figura 16, é possível perceber que $\chi^{L_{a,b}}$ e $\chi^{L_{a,d}} \in F_a \subseteq F_b$, e portanto $0 = b_0 - b_0 = b^T \chi^{L_{a,b}} - b^T \chi^{L_{a,d}} = b_b - b_d$, logo $b_b = b_d = \alpha$. Fazendo esse mesmo processo $\forall v \in I_k$ temos que $b_v = \alpha$.

Observando o subgrafo I_k de S_k , ilustrado na Figura 16, é possível perceber que $\chi^{I_k} \in F_a \subseteq F_b$, e portanto, $0 = b_0 - b_0 = b^T \chi^{I_k} - b^T \chi^{L_{a,b}} = \sum_{v \in I_k - \{a,b\}} b_v - b_c$, como $\|I_k\| = k$ e $\forall v \in I_k$ vale $b_v = \alpha$, e portanto, $b_c = \alpha(k - 2)$.

□

Lema 3.4. Para todo $v \notin V[S_k]$ vale que $b_v = 0$.

Demonstração. É possível perceber que $\chi^{I_k \cup \{l\}} \in F_a \subseteq F_b$, e portanto $0 = b_0 - b_0 = b^T \chi^{I_k \cup \{l\}} - b^T \chi^{I_k} = b_l$, logo, $b_l = 0$. Fazendo este mesmo processo $\forall v \notin V[S_k]$ temos que $b_v = 0$.

□

Os dois lemas implicam que a inequação (3) define uma faceta de P_n para estrelas com vértice central único.

□

Como a inequação (2) é um caso particular da inequação (3) quando $k = 3$, pode-se reescrever (\mathcal{F}_g) da seguinte forma:

$$\begin{aligned}
 (\mathcal{F}_{S_k}) \min & \sum (1 - x_i) \\
 \text{sujeito a } & 0 \leq x_v \leq 1, & \forall v \in V, \\
 \sum_{v \in I_k} x_v + (k - 2)x_c & \leq k, & \forall S_k \subset G_n, k \in 3, 4, \dots, \|V\| - 1 \\
 & x \text{ é inteiro.}
 \end{aligned}$$

3.4 Planos de Corte para o (\mathcal{F}_{S_k})

Seja $G(V, E)$ um grafo. Cada subgrafo Estrela $S_k \subset G$ está associado a um vértice central $v \in V$ e um conjunto independente $I_k \subset G[adj(v)]$, onde $G[adj(v)]$ é o subgrafo de G induzido por $adj(v)$. Pode-se observar, por exemplo, que o grafo estrela S_4 , ilustrado na Figura 17c, é composto pelo vértice $v \in V$, ilustrado na Figura 17a, e o conjunto independente $I_k = \{0, 1, 4, 6\}$ contido no subgrafo induzido $G[adj(v)]$, ilustrado na Figura 17b.

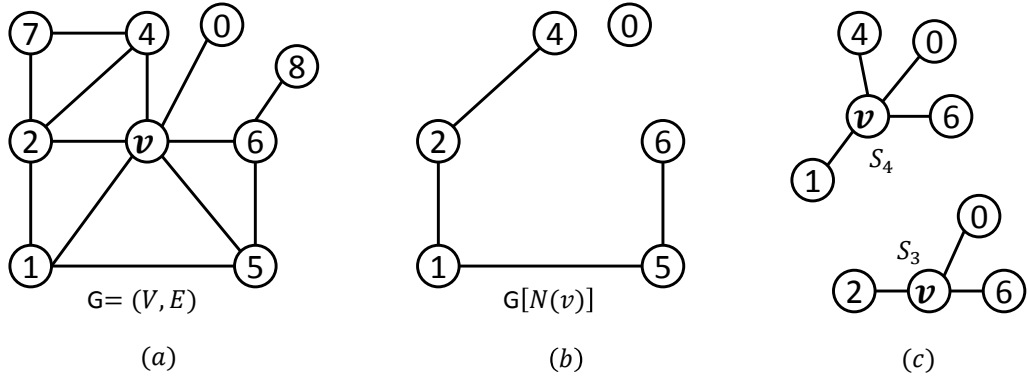


Figura 17: (a) Grafo de entrada $G = (V, E)$. (b) Subgrafo de G induzido pelos vértices vizinhos de v ($N(v) = adj(v)$). (c) Grafos Estrela compostos por 2 conjuntos independentes de $G[adj(v)]$ e o vértice v .

Segundo [8], para um dado grafo G , seja $i(G)$ o número de conjuntos independentes de G , então temos que:

$$i(G) \leq 2^{iso(G)} \prod_{vu \in E[G]} (2^{\delta(v)} + 2^{\delta(u)} - 1)^{\frac{1}{\delta(v)\delta(u)}},$$

onde $\delta(v)$ é o número de vizinhos de v e $iso(G)$ é o número de vértices isolados.

Sendo assim, existe um número exponencial de inequações (3), tornando computacionalmente inviável a inserção de todas elas *a priori*, sobretudo em instâncias de médio e grande porte. Portanto, será usado um procedimento baseado em planos de corte para adicioná-las sempre que violadas. A construção do conjunto das inequações se dá através da obtenção da lista de conjuntos independentes maximais (I) contidos em cada subgrafo induzido $G[adj(v)]$ para todo $v \in V$. Para isso, foi utilizado o algoritmo *hybrid(.)*, proposto por [5], sobre o grafo complementar $\overline{G[adj(v)]}$ para identificação das suas cliques maximais, que por sua vez, representam a lista de conjuntos independentes maximais I_v de $G[adj(v)]$. Logo, a construção de um grafo Estrela S_k se dá pela composição $S_k = G[\{v\} \cup I_k]$, onde $I_v = hybrid(\overline{G[adj(v)]})$ $v \in V$ e $I_k \subset I_v \subset I$. O algoritmo de separação 2 demonstra a forma como os subgrafos estrelas de $G(V, E)$ foram obtidos.

A otimização do PLI (\mathcal{F}_{S_k}) é heurísticamente inicializada com as inequações (3)

Algoritmo 2: ALGORITMO DE SEPARAÇÃO

Entrada: $G(V, E)$
Saída: S

- 1 **início**
- 2 Cria-se o conjunto S que representará os subgrafos estrela compostos por um vértice central e um conjunto independente
- 3 **para** cada vértice $v \in V$ **faça**
- 4 $G'(V', E') = \overline{G[adj(v)]}$
- 5 $I_v = hybrid(G')$
- 6 **para** cada $I_k \in I_v$ **faça**
- 7 **se** $k \geq 2$ **então**
- 8 | Adiciona I_k e v em S
- 9 **fim**
- 10 **fim**
- 11 **fim**
- 12 **retorna** S
- 13 **fim**

associadas aos $10 \times \|V\|$ maiores conjuntos independentes encontrados em S (Consulte algoritmo 2 para detalhes sobre S). Em seguida, para cada $v \in V[G_n]$ procura-se por inspeção até 4 grafos S_3 que tenham v como vértice central para adicionar suas respectivas restrições associadas. Essa versão restrita do (\mathcal{F}_{S_k}) é, então, resolvida e o vetor x^* correspondente à solução da relaxação de (\mathcal{F}_{S_k}) é coletado. Logo após, é executada uma heurística para separação que verifica se alguma inequação $S_k \subset S$ baseada em (3) foi violada de acordo com a inequação (4).

$$\sum_{v \in I_k} x_v^* + (k - 2)x_c^* - k \geq \epsilon \quad (4)$$

onde $\epsilon = 0,0001$, $k = \sum_{x_v^* > \epsilon: \forall v \in I_k} 1$, $I_k \in S_k$ e $x_c \in S_k$. Caso a inequação tenha sido violada, ela é adicionada a uma fila de prioridades que será utilizada para selecionar as $10 \times \|V\|$ inequações mais violadas.

Os métodos estudados neste trabalho foram desenvolvidos em linguagem C++, utilizando o compilador g++ versão 4.6.3 (opção de compilação -O3), com uso da tecnologia Concert do resolvidor matemático CPLEX versão 12.04. Os experimentos computacionais foram executados em um computador com processador *Intel Core I7* com 4 *cores* de 3,40 GHz, 16 GB de RAM e executando o sistema operacional Linux Ubuntu 14.04. Os experimentos foram realizados com 1 *thread* e tempo limite de duas horas. Como entrada, foram utilizados grafos gerados por [1]. Escolheu-se as instâncias com $\|V\| = \{25, 50, 100\}$ e $d > 0,3$, onde d representa a densidade do grafo, formando 31 instâncias de teste.

4 RESULTADOS COMPUTACIONAIS

A Tabela 1 apresenta a comparação dos resultados computacionais dos dois métodos propostos neste trabalho, cada linha representa o resultado de uma instância específica. Nessa tabela tem-se o primeiro conjunto de colunas representando o grupo de instâncias testados com $n = \{25, 50\}$, sendo *id* a coluna de identificação da instância e *Opt* o valor ótimo da instância, enquanto as colunas restantes são divididas em dois grupos: \mathcal{F}_g e \mathcal{F}_{S_k} , representado, respectivamente, os métodos baseados nos modelos \mathcal{F}_g e \mathcal{F}_{S_k} . O conjunto de colunas \mathcal{F}_g é dividido em dois grupos: *root*, que possui o resultado da relaxação linear do nó raiz e *B&C* com os resultados do método completo. O conjunto *root* possui as colunas **LB** (*lower bound*) com o valor da solução na relaxação linear na raiz e a coluna **T(s)** com o tempo computacional em segundos. O conjunto *B&C* possui a coluna **T(s)** com o tempo computacional do método e **#Nós** que indica o número de nós resolvidos para encontrar a solução ótima. Já o conjunto de colunas \mathcal{F}_{S_k} que também é dividido em dois grupos: *root* e *B&C* possui as mesmas colunas do grupo anterior, com a adição, no conjunto *B&C*, das colunas **#Cortes** que indica o número de cortes adicionados pelo algoritmo de planos de corte proposto e $T_{sep}(s)$ que indica o tempo, em segundos, da heurística de separação das inequações violadas durante o método de planos de corte embutido no *Branch-and-Cut* proposto.

Tabela 1: Resultados computacionais para instâncias com $\|V\| = 50$.

| <i>Instâncias</i> | | \mathcal{F}_g | | | | \mathcal{F}_{S_k} | | | | | |
|-------------------|------------|-----------------|--------------|----------------|---------------|---------------------|--------------|--------------|----------------|----------------|---------------|
| | | <i>root</i> | | <i>B&C</i> | | <i>root</i> | | | <i>B&C</i> | | |
| <i>id</i> | <i>Opt</i> | LB | T (s) | T (s) | #Nós | LB | T (s) | T (s) | #Nós | #Cortes | T_{sep} (s) |
| 101.6.25 | 11 | 7,86 | 0,15 | 0,32 | 59 | 9,87 | 0,07 | 0,13 | 3 | 0 | 0 |
| 101.7.25 | 11 | 7,98 | 0,18 | 0,38 | 58 | 11 | 0,08 | 0,16 | 1 | 0 | 0 |
| 101.8.25 | 13 | 7,77 | 0,13 | 0,33 | 252 | 9,65 | 0,1 | 0,22 | 47 | 21 | 0 |
| 101.9.25 | 12 | 7,86 | 0,19 | 0,44 | 150 | 9,53 | 0,17 | 0,37 | 40 | 24 | 0 |
| 101.10.25 | 13 | 8 | 0,12 | 0,3 | 237 | 9,48 | 0,09 | 0,2 | 48 | 15 | 0 |
| 102.6.25 | 12 | 7,89 | 0,2 | 0,46 | 166 | 9,62 | 0,08 | 0,17 | 17 | 13 | 0 |
| 102.7.25 | 13 | 7,82 | 0,23 | 0,52 | 252 | 9,61 | 0,11 | 0,24 | 49 | 28 | 0 |
| 102.8.25 | 12 | 7,91 | 0,19 | 0,42 | 106 | 9,83 | 0,09 | 0,18 | 11 | 4 | 0 |
| 102.9.25 | 14 | 7,94 | 0,25 | 0,66 | 639 | 9,53 | 0,13 | 0,31 | 188 | 68 | 0 |
| 102.10.25 | 13 | 8,01 | 0,33 | 0,76 | 298 | 9,49 | 0,13 | 0,28 | 28 | 3 | 0 |
| 105.6.50 | 32 | 15,12 | 6,02 | 163,09 | 76850 | 20,97 | 1,41 | 31,56 | 6279 | 4881 | 0,43 |
| 105.7.50 | 32 | 15,23 | 6,73 | 187,37 | 92869 | 21,09 | 1,2 | 28,12 | 5376 | 5347 | 0,45 |
| <u>105.8.50</u> | 35 | <u>15,38</u> | <u>13,37</u> | <u>422,4</u> | <u>140614</u> | <u>21,07</u> | <u>2,21</u> | <u>82,05</u> | <u>12784</u> | <u>7563</u> | <u>1,3</u> |
| 105.9.50 | 35 | 15,58 | 16,84 | 382,9 | 128933 | 21,12 | 2,51 | 68,64 | 9875 | 7066 | 0,13 |
| 105.10.50 | 35 | 15,16 | 13,18 | 435,5 | 156555 | 20,77 | 2,63 | 75,66 | 12608 | 5843 | 1,44 |
| 106.5.50 | 31 | 15,23 | 4,63 | 102 | 48413 | 21,09 | 1,32 | 15,5 | 2888 | 3555 | 0,2 |
| 106.6.50 | 32 | 15,15 | 6,53 | 156,67 | 60757 | 21,07 | 1,2 | 21,12 | 4064 | 4035 | 0,27 |
| 106.7.50 | 34 | 15,18 | 10,32 | 259,7 | 101508 | 21,05 | 2,16 | 33,34 | 4979 | 4811 | 0,51 |
| 106.8.50 | 35 | 15,13 | 10,38 | 403,35 | 145112 | 20,99 | 2,62 | 15,46 | 27547 | 8746 | 2,59 |
| 106.9.50 | 35 | 15,21 | 10,64 | 441,2 | 158671 | 20,97 | 2,59 | 85,82 | 13048 | 7319 | 1,47 |
| 106.10.50 | 34 | 15,49 | 11,11 | 294,39 | 84188 | 20,76 | 3,11 | 65,81 | 9458 | 6039 | 1,33 |

Os dois métodos propostos encontraram a solução ótima de todas as instâncias

contidas na Tabela 1. Entretanto, o \mathcal{F}_{S_k} resolveu todas as instâncias de forma mais eficiente. Pode-se observar, por exemplo, que para a instância ($id = 105_8_50$) o método \mathcal{F}_g resolveu a instância em 422 segundos precisando processar 140.614 nós para encontrar a solução ótima. Já o método \mathcal{F}_{S_k} resolveu a instância em apenas 82 segundos, processando apenas 12.784 nós. Essa melhora é explicada devido ao impacto das novas inequações propostas no modelo \mathcal{F}_{S_k} . Note que a relaxação linear melhorou de 15,38 para 21,07, gerando um ganho de 36% no *lower bound* da instância. Essa característica pode ser observada no resultado de todas as instâncias.

Na tabela 2, que apresenta os resultados computacionais das instâncias com $n = 100$, nenhum método finalizou sua execução em menos de duas horas de processamento. Entretanto, é possível perceber que o método \mathcal{F}_{S_k} permanece mais eficiente na resolução das instâncias. Os conjuntos de colunas continuam iguais ao da tabela anterior. Como não foram encontradas soluções ótimas, foram removidas as colunas *Opt* de **Instâncias** e as colunas **T(s)** e **#Nós** de cada método. Para comparar seus resultados adiciona-se no conjunto de colunas *B&C* as colunas **LB** (*lower bound*), **UB** (*upper bound*) que são os valores alcançados por cada método específico em duas horas de processamento e **GAP** = $(LB-UB)/UB*100$ que representa a porcentagem da diferença entre **LB** e **UB**.

Tabela 2: Resultados computacionais para instâncias com $\|V\| = 100$.

| <i>Instâncias</i> | \mathcal{F}_g | | | | | \mathcal{F}_{S_k} | | | | | | |
|-------------------|-----------------|--------------|----------------|-----------|-------------|---------------------|--------------|----------------|-----------|-------------|----------------|---------------------------------|
| | root | | <i>B&C</i> | | | root | | <i>B&C</i> | | | | |
| | LB | T (s) | LB | UB | GAP | LB | T (s) | LB | UB | GAP | #Cortes | T_{sep} (s) |
| 109_7_100 | 30,02 | 1187,59 | 43,69 | 81 | 46,1 | 44,45 | 406,74 | 60,1 | 80 | 24,9 | 133876 | 69,9 |
| 109_8_100 | 30,09 | 1753,61 | 43,71 | 83 | 47,3 | 44,19 | 551,14 | 61,31 | 82 | 25,2 | 126453 | 74,26 |
| 109_9_100 | 30,3 | 3134,32 | 41,72 | 83 | 49,7 | 43,92 | 639,26 | 60,96 | 83 | 26,6 | 122985 | 77,96 |
| 109_10_100 | 30,43 | 3872,27 | 40,33 | 81 | 50,2 | 43,41 | 663,1 | 63,24 | 82 | 22,9 | 103525 | 85,16 |
| 110_5_100 | - | EM | - | - | - | 44,42 | 281,38 | 59,02 | 77 | 23,4 | 120124 | 62,21 |
| 110_6_100 | 30,38 | 1536,55 | 46 | 80 | 42,5 | 44,44 | 352,09 | 60,8 | 80 | 24 | 130519 | 70,18 |
| 110_7_100 | 30,54 | 1858,81 | 44 | 82 | 46,3 | 44,38 | 463,19 | 60,1 | 82 | 26,7 | 126578 | 72,61 |
| 110_8_100 | 30,19 | 1793,49 | 44 | 84 | 47,6 | 43,82 | 586,73 | 62,5 | 83 | 24,7 | 122918 | 77,62 |
| 110_9_100 | <u>30,29</u> | 3828,67 | 38,69 | 85 | <u>54,5</u> | 43,94 | 457,21 | 62,8 | 83 | <u>24,3</u> | 112755 | 76,41 |
| 110_10_100 | 30,35 | 5928,82 | 34,33 | 86 | 60,1 | 43,69 | 409,32 | 62,94 | 85 | 26 | 104797 | 82,88 |

Por fim, a Tabela 2 mostra, para $\|V\| = 100$, que o método \mathcal{F}_{S_k} continua sendo mais eficiente na resolução de todas as instâncias. Na instância 110_9_100, por exemplo, o **GAP** caiu de 54,5% para 24,3%, motivado também pelo impacto das novas inequações definidas no modelo \mathcal{F}_{S_k} que elevam o *lower bound* da relaxação linear de 30,29 para 43,94 em um ganho superior a 45%. Os resultados obtidos sugerem que a heurística de separação das inequações violadas é eficiente, pois além de acelerar a resolução dos problemas, gastou menos de 1 minuto e meio nas maiores instâncias.

5 CONCLUSÕES E TRABALHOS FUTUROS

Um estudo poliédrico do PLG foi proposto, identificando dois modelos PLI: o \mathcal{F}_g que possui as facetas definidas pelas inequações garras básicas e o \mathcal{F}_{S_k} que possui novas facetas definidas pelas inequações estrela como uma generalização das inequações garra. Como o número de inequações estrela é exponencial, foi proposto ainda um algoritmo baseado em planos de corte para resolver a relaxação linear deste problema de forma mais eficiente.

Os resultados computacionais demonstram o impacto das novas facetas propostas em \mathcal{F}_{S_k} , onde a resolução de todas as instâncias se mostrou mais eficiente no tempo computacional, além de melhorar os valores de *lower bound* e *upper bound* das soluções não resolvidas em 2 horas de processamento.

Como propostas de trabalhos futuros tem-se: encontrar novas facetas para o PLG, construir instâncias específicas para o problema (grafos de intervalo), realizar estudo para identificação dos parâmetros de complexidade dessas instâncias, e por fim, construir métodos heurísticos para a resolução do PLG em instâncias de grande porte.

Referências

- [1] Lucas Bastos et al. “Efficient algorithms for cluster editing”. Em: *Journal of Combinatorial Optimization* 31.1 (2016), pp. 347–371. ISSN: 1382-6905.
- [2] Ralf Borndörfer. “Aspects of set packing, partitioning, and covering”. Em: *Ph. D. Dissertation, Technical University of Berlin* (1998).
- [3] Stephen A. Cook. “The Complexity of Theorem-proving Procedures”. Em: *Proceedings of the Third Annual ACM Symposium on Theory of Computing*. STOC ’71. Shaker Heights, Ohio, USA: ACM, 1971, pp. 151–158. DOI: 10.1145/800157.805047. URL: <http://doi.acm.org/10.1145/800157.805047>.
- [4] Thomas H. Cormen et al. *Introduction to Algorithms, Third Edition*. 3rd. The MIT Press, 2009. ISBN: 0262033844, 9780262033848.
- [5] David Eppstein e Darren Strash. “Listing All Maximal Cliques in Large Sparse Real-World Graphs”. Em: *Experimental Algorithms: 10th International Symposium, SEA 2011, Kolimpari, Chania, Crete, Greece, May 5-7, 2011. Proceedings*. Ed. por Panos M. Pardalos e Steffen Rebennack. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 364–375. ISBN: 978-3-642-20662-7. DOI: 10.1007/978-3-642-20662-7_31. URL: http://dx.doi.org/10.1007/978-3-642-20662-7_31.
- [6] Carlos Ferreira e Yoshiko Wakabayashi. “Combinatória Polidédrica e Planos-de-Corte Faciais”. Texto produzido pelo departamento de Ciência da Computação IME – USP. URL: <https://www.ime.usp.br/~yw/2010/progint/livro-update2010.pdf>.
- [7] Peter C. Fishburn. “Interval graphs and interval orders”. Em: *Discrete Mathematics* 55.2 (1985), pp. 135–149. ISSN: 0012-365X. DOI: [http://dx.doi.org/10.1016/0012-365X\(85\)90042-1](http://dx.doi.org/10.1016/0012-365X(85)90042-1). URL: <http://www.sciencedirect.com/science/article/pii/0012365X85900421>.
- [8] David Galvin e Yufei Zhao. “The Number of Independent Sets in a Graph with Small Maximum Degree”. Em: *Graphs and Combinatorics* 27.2 (2011), pp. 177–186. ISSN: 1435-5914. DOI: 10.1007/s00373-010-0976-z. URL: <http://dx.doi.org/10.1007/s00373-010-0976-z>.
- [9] Ralph E. Gomory. “Outline of an algorithm for integer solutions to linear programs”. Em: *Bull. Amer. Math. Soc.* 64.5 (set. de 1958), pp. 275–278. URL: <http://projecteuclid.org/euclid.bams/1183522679>.
- [10] Michael Jünger et al., eds. *50 Years of Integer Programming 1958-2008 - From the Early Years to the State-of-the-Art*. Springer, 2010. ISBN: 978-3-540-68274-5. DOI: 10.1007/978-3-540-68279-0. URL: <https://doi.org/10.1007/978-3-540-68279-0>.

- [11] A. H. Land e A. G. Doig. “An automatic method for solving discrete programming problems”. Em: *ECONOMETRICA* 28.3 (1960), pp. 497–520.
- [12] John M. Lewis e Mihalis Yannakakis. “The node-deletion problem for hereditary properties is NP-complete”. Em: *Journal of Computer and System Sciences* 20.2 (1980), pp. 219 –230. ISSN: 0022-0000. DOI: [http://dx.doi.org/10.1016/0022-0000\(80\)90060-4](http://dx.doi.org/10.1016/0022-0000(80)90060-4). URL: <http://www.sciencedirect.com/science/article/pii/0022000080900604>.
- [13] Peter J Looges e Stephan Olariu. “Optimal greedy algorithms for indifference graphs”. Em: *Computers & Mathematics with Applications* 25.7 (1993), pp. 15–25.
- [14] William R Pulleyblank e Bruce Shepherd. “Formulations for the stable set polytope”. Em: *Proceedings Third IPCO Conference*. 1993, pp. 267–279.
- [15] F Bruce Shepherd. “Hamiltonicity in claw-free graphs”. Em: *Journal of Combinatorial Theory, Series B* 53.2 (1991), pp. 173–194.
- [16] René Van Bevern et al. “Measuring Indifference: Unit Interval Vertex Deletion”. Em: *Proceedings of the 36th International Conference on Graph-theoretic Concepts in Computer Science*. WG’10. Zarós, Crete, Greece: Springer-Verlag, 2010, pp. 232–243. ISBN: 3-642-16925-2, 978-3-642-16925-0. URL: <http://dl.acm.org/citation.cfm?id=1939238.1939262>.