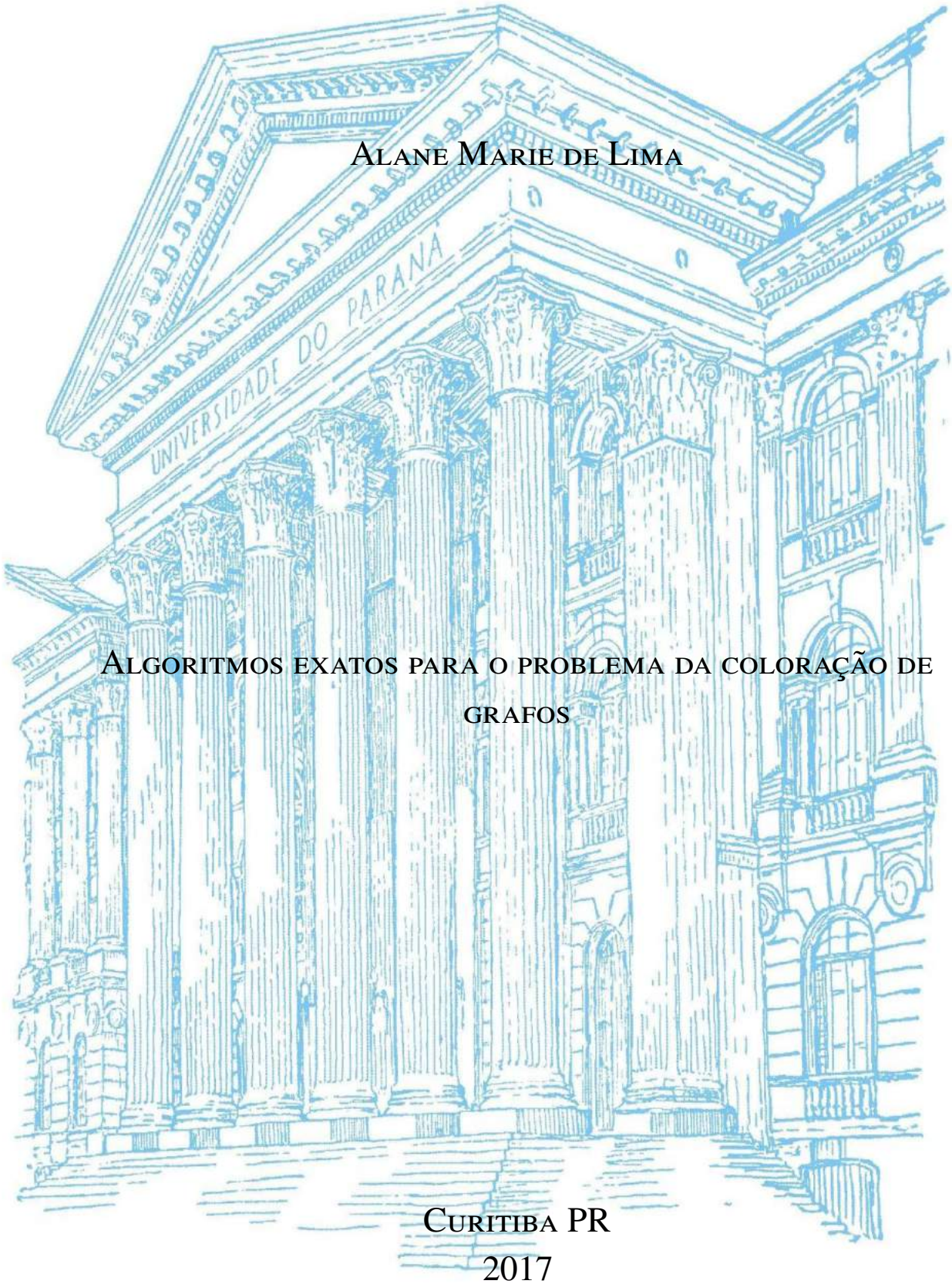
Universidade Federal do Paraná



Alane Marie de Lima

Algoritmos exatos para o problema da coloração de grafos

Curitiba PR
2017

Alane Marie de Lima

Algoritmos exatos para o problema da coloração de grafos

Dissertação apresentada como requisito parcial à obtenção do grau de Mestre em Informática no Programa de Pós-Graduação em Informática, setor de Ciências Exatas, da Universidade Federal do Paraná.

Área de concentração: *Ciência da Computação*.

Orientador: Renato Carmo.

Curitiba PR
2017

L732a Lima, Alane Marie de
Algoritmos exatos para o problema da coloração de grafos / Alane Marie de Lima. – Curitiba, 2017.
83 f. : il. color. ; 30 cm.

Dissertação - Universidade Federal do Paraná, Setor de Ciências Exatas, Programa de Pós-Graduação em Informática, 2017.

Orientador: Renato Carmo.

1. Teoria dos grafos. 2. Coloração de grafos. 3. Algoritmos. I. Universidade Federal do Paraná. II. Carmo, Renato. III. Título.

CDD: 005.1

MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DO PARANÁ
PRÓ-REITORIA DE PESQUISA E PÓS-GRADUAÇÃO
Setor CIÊNCIAS EXATAS
Programa de Pós-Graduação INFORMÁTICA

TERMO DE APROVAÇÃO

Os membros da Banca Examinadora designada pelo Colegiado do Programa de Pós-Graduação em INFORMÁTICA da Universidade Federal do Paraná foram convocados para realizar a arguição da dissertação de Mestrado de **ALANE MARIE DE LIMA** intitulada: **Algoritmos exatos para o problema de coloração de grafos.**, após terem inquirido a aluna e realizado a avaliação do trabalho, são de parecer pela sua Aprovação no rito de defesa.

A outorga do título de mestre está sujeita à homologação pelo colegiado, ao atendimento de todas as indicações e correções solicitadas pela banca e ao pleno atendimento das demandas regimentais do Programa de Pós-Graduação.

Curitiba, 21 de Agosto de 2017.

RENATO JOSÉ DA SILVA CARMO
Presidente da Banca Examinadora (UFPR)

MURILO VICENTE GONÇALVES DA SILVA
Avaliador Externo (UTFPR)

ANDRÉ LUIZ PIRES GUEDES
Avaliador Interno (UFPR)

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

"Ainda que eu tenha o dom da profecia, o conhecimento de todos os mistérios e de toda a ciência, ainda que eu tenha toda a fé, a ponto de remover montanhas, se eu não tenho o amor, eu nada sou." (I Coríntios 13:2)

Agradecimentos

Agradeço primeiramente a Deus por me iluminar e abençoar em todos os momentos de minha vida.

Agradeço a meus pais, Raquel e Luiz, e a meu irmão, Pedro Paulo, por serem os primeiros a acreditar em meu potencial e me incentivar em meu crescimento pessoal e profissional. Agradeço também a meus pais por me ensinarem desde pequena os valores alicerce de meu caráter.

Agradeço aos demais familiares pela preocupação e motivação dada a mim durante esta jornada.

Agradeço a meu namorado Ricardo pelo companheirismo e por todo o suporte dado a mim.

Agradeço aos professores da UNICENTRO por terem me incentivado desde o período da graduação a seguir a carreira acadêmica. Agradeço ao professor Fábio Hernandes por sempre ter me motivado enquanto orientador na graduação a cursar o mestrado.

Agradeço ao professor Renato Carmo pela dedicação exemplar e pela paciência e confiança depositadas em mim durante o período de orientação.

Agradeço aos professores André Luiz Pires Guedes e Murilo Vicente Gonçalves da Silva pelas importantes contribuições dadas na qualificação e defesa final. Agradeço aos professores Renato Carmo, André Vignatti e André Luiz Pires Guedes por terem me acolhido no Grupo de Pesquisa em Algoritmos e pelas conversas e conselhos valiosos.

Agradeço a meus amigos e colegas do Grupo de Pesquisa em Algoritmos pelos momentos de descontração, companheirismo e apoio.

Agradeço aos amigos de Guarapuava e aos que fiz em Curitiba pelos bons momentos compartilhados.

Agradeço a Casa do Estudante Universitário do Paraná por ter me recebido em terras curitibanas.

Agradeço a CAPES pela concessão de bolsa durante o período de mestrado.

Resumo

O problema de coloração de grafos consiste em particionar os vértices de um grafo na menor quantidade possível de conjuntos independentes. Este trabalho tem como objetivo agrupar e contextualizar alguns dos principais algoritmos para o problema de coloração de grafos, reunindo num mesmo texto informações que se encontram espalhadas pela literatura técnica sob a forma de artigos científicos. Discutimos os principais métodos apresentados na literatura para o problema de coloração de grafos, a saber, soluções baseadas em Programação Linear Inteira, *Branch-and-Bound* e Programação Dinâmica.

Palavras-chave: Teoria dos Grafos, Coloração de Grafos, Algoritmos Exatos.

Abstract

The graph coloring problem is the problem of partitioning the vertices of a graph into the smallest possible set of independent sets. The goal of this work is to group and contextualize some of the main algorithms for the graph coloring problem, bringing into a single text information which is scattered in the technical literature in scientific papers. We discuss the main methods in the literature for the graph coloring problem, namely, solutions based on Integer Linear Programming, Branch-and-Bound and Dynamic Programming.

Keywords: Graph Theory, Graph Coloring, Exact Algorithms.

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Problemas de otimização da classe NP-Difícil não possuem algoritmos conhecidos que os solucionem em tempo polinomial. Apesar de soluções exatas para tais problemas serem muito mais caras computacionalmente, é de grande interesse da computação encontrar algoritmos desta natureza tão eficientes quanto possível. O problema da coloração de grafos é um dentre estes problemas.

Colorir um grafo não-direcionado significa atribuir cores aos seus vértices respeitando a restrição de que vértices vizinhos não tenham a mesma cor, de modo que a quantidade mínima de cores seja utilizada na solução ótima. A coloração ótima resultante é uma partição dos vértices na menor quantidade de conjuntos independentes possível.

O problema da coloração de grafos é um problema de otimização combinatória. Uma das maneiras de solucioná-lo é por meio de busca exaustiva, ou seja, enumerando todas as soluções possíveis para então determinar a melhor dentre elas. O trabalho de Lawler (1976), por exemplo, encontra o número cromático de cada subconjunto de vértices possível do grafo de entrada para então determinar o número cromático total.

Dentre os algoritmos conhecidos na literatura que solucionam o problema da coloração de maneira exata, tem-se aqueles baseados em Árvores de Zykov (McDiarmid, 1979), Programação Dinâmica (Byskov, 2002), DSATUR (Furini et al., 2017), *Branch-and-Price* (Malaguti et al., 2011; Gualandi e Malucelli, 2012), *Branch-and-Cut* (Méndez-Díaz e Zabala, 2008) e Inclusão-Exclusão (Björklund et al., 2009). Um levantamento bibliográfico sobre diferentes maneiras de solucionar o problema de coloração de grafos pode ser encontrado no trabalho de Chiarandini e Gualandi (2013)[1] e Lewis (2016).

Neste trabalho, iremos apresentar os algoritmos baseados em Programação Dinâmica associada a pré-coloração de subgrafos do grafo de entrada; algoritmos baseados em *Branch-and-Bound* e *backtracking*, e algoritmos que modelam instâncias do problema de coloração de grafos como programas lineares inteiros, solucionando-os por meio de *Branch-and-Price*.

Apesar das abordagens exatas que solucionam instâncias do problema de coloração de grafos diferirem entre si, algumas delas apresentam similaridades e correlações. Porém, visto que algumas destas informações encontram-se dispersas na literatura, nem sempre esta correlação torna-se facilmente visível a quem busca por ela. Desta forma, procuramos facilitar o acesso ao leitor em sua busca por estas informações. Além disso, buscamos deixar claros conceitos, definições e informações necessárias para a introdução e compreensão dos algoritmos estudados, de maneira que o leitor tenha uma visão ampla sobre o problema.

A organização deste trabalho se encontra da seguinte forma: o Capítulo 2 apresenta limitantes ao valor de uma solução ótima de uma instância do problema de coloração de grafos e

[1] Disponível em http://www.imada.sdu.dk/~marco/gcp/

casos em que o problema é solucionável em tempo polinomial. Os Capítulos 3, 4 e 5 apresentam os algoritmos encontrados na literatura baseados em Programação Dinâmica, *Branch-and-Bound* e Programação Linear Inteira, respectivamente. Por fim, o Capítulo 6 descreve as conclusões.

1.1 Definições e notação

Um grafo G é um par $(V(G), E(G))$, onde $V(G)$ é um conjunto finito e $E(G) \subseteq \binom{V(G)}{2}$. Cada elemento de $V(G)$ é chamado de vértice e cada elemento de $E(G)$ é chamado de aresta.

Um grafo é *completo* se $E(G) = \binom{V(G)}{2}$. Um grafo completo é denotado K_n, para $n = |V(G)|$.

O *complemento* de um grafo G é o grafo $\overline{G} = (V(G), K \setminus E(G))$ onde $K = \binom{V(G)}{2}$.

Um vértice v é vizinho de um vértice u no grafo G se $\{u, v\}$ é uma aresta e de G. Neste caso, dizemos que os vértices u e v são as pontas de e, e dizemos que a aresta e é incidente em u e v. A vizinhança do vértice v é denotada $N_G(v) = \{u \in V(G) \text{ tal que } \{u, v\} \in E(G)\}$. A anti-vizinhança do vértice v em G é o conjunto de vértices que não são vizinhos de v, denotado $\overline{N}_G(v) = \{u \in V(G) \text{ tal que } \{u, v\} \notin E(G)\}$. O grau do vértice v em G é a quantidade de arestas incidentes a v em G, denotado $d_G(v) = |N_G(v)|$. O menor grau de um vértice v em G, denominado *grau mínimo* de G, é denotado $\delta(G)$. O maior grau de um vértice v em G, denominado *grau máximo* de G, é denotado $\Delta(G)$.

A *densidade* de um grafo G é a razão do número de arestas de G em relação ao número máximo possível de arestas de G. Isto é, se o número máximo possível de arestas de um grafo é $\frac{|V(G)|(|V(G)|-1)}{2}$, então a densidade é igual a $\frac{2|E(G)|}{|V(G)|(|V(G)|-1)}$.

Um grafo H é subgrafo de um grafo G se $V(H) \subseteq V(G)$ e $E(H) \subseteq E(G)$. Dado um conjunto $S \subseteq V(G)$, o subgrafo induzido por S é o subgrafo com maior conjunto de arestas possível de G cujo conjunto de vértices é S, denotado por $G[S] = \left(S, E(G) \cap \binom{|V(G)|}{2}\right)$. Um subconjunto $S \subseteq V(G)$ é uma *clique* de G se o grafo $G[S]$ é completo. Se k é um inteiro, uma clique de tamanho k é uma k-clique. O tamanho de uma clique máxima de G é denotado por $\omega(G)$.

Dado $X \subseteq V(G)$ definimos $G - X$ como o grafo dado por $V(G - X) = V(G) \setminus X$ e $E(G - X) = E(G) \setminus \{a \in E(G) \text{ tal que } a \cap X \neq \emptyset\}$. Se $v \in V(G)$, denotamos $G - v = G - \{v\}$.

Um *passeio* P em um grafo G é uma sequência $(v_0, ..., v_n)$ de vértices de G na qual v_{i-1} e v_i são vizinhos, para todo $1 \leq i \leq n$. Os vértices v_0 e v_n são chamados respectivamente *início* e *fim* de P. Todo vértice diferente do início e do fim é denominado *vértice interno* do passeio. Um passeio é *fechado* se seu início e fim coincidem. Um *caminho* é um passeio cujos vértices são todos distintos. Uma *trilha* é um passeio cujas arestas são todas distintas. Um *ciclo* (ou *circuito*) é uma trilha fechada cujos vértices internos são todos distintos. O subgrafo induzido por um ciclo é denominado *ciclo induzido*. Um ciclo induzido de n vértices é denotado C_n. Um grafo é *acíclico* se não tem *ciclos*. Um grafo é *conexo* se existir um caminho entre qualquer par de vértices.

Um *componente* do grafo G é um subgrafo conexo maximal de G. O conjunto dos componentes de G é denotado $\mathcal{C}(G)$. Um conjunto K de vértices de um grafo G é um corte de vértices se $G - K$ tem mais componentes do que G, isto é, se $|\mathcal{C}(G - K)| > |\mathcal{C}(G)|$.

Uma *árvore* é um grafo acíclico conexo. Um vértice de grau 1 em uma árvore é chamado de *folha*. Uma *floresta* é um grafo em que cada componente é árvore, ou seja, é um grafo acíclico.

Um subconjunto $S \subseteq V(G)$ é independente se $E(G[S]) = \emptyset$. Denotamos por $\mathbf{I}(G)$ o conjunto dos conjuntos independentes de G. O tamanho do maior conjunto independente de G é

denotado por $\alpha(G)$. Dado $s \in \mathbb{N}$, o conjunto de conjuntos independentes de tamanho s em G é denotado por $I_s(G)$.

Dado $k \leq |V(G)|$, uma *k-coloração* de G é uma partição de $V(G)$ em k conjuntos independentes, onde cada conjunto I_k é chamado de *cor* da coloração. Um grafo G é k-colorível se admite uma k-coloração. O menor k para qual G é k-colorível é chamado de *número cromático* de G, denotado $\chi(G)$. Uma 2-partição de G em conjuntos independentes também é denominada *bipartição* de G.

Definimos os seguintes problemas computacionais:

Coloração de grafos
Instância: um grafo G.
Resposta: uma coloração ótima de G.

Número cromático
Instância: um grafo G.
Resposta: $\chi(G)$.

3-coloração
Instância: um grafo G.
Resposta: SIM, se for possível colorir G com até 3 cores; NÃO, caso contrário.

Programação Linear (PL)
Instância: uma matriz $A \in \mathbb{Q}^{n \times m}$ e dois vetores $b \in \mathbb{Q}^m$ e $c \in \mathbb{Q}^n$, onde $n, m \in \mathbb{N}^*$.
Resposta: um vetor $x \in \mathbb{Q}^n_+$ tal que $Ax = b$ e $c^T x$ é mínimo.

Programação Linear Inteira (PLI)
Instância: uma matriz $A \in \mathbb{Q}^{n \times m}$ e dois vetores $b \in \mathbb{Q}^m$ e $c \in \mathbb{Q}^n$, onde $n, m \in \mathbb{N}^*$.
Resposta: um vetor $x \in \mathbb{Z}^n_+$ tal que $Ax = b$ e $c^T x$ é mínimo.

Cobertura de conjuntos
Instância: um conjunto U e um conjunto D formado por subconjuntos $D_1, D_2, ..., D_n$ de U.
Resposta: o menor conjunto $I \subseteq \{1, ..., n\}$, tal que $\bigcup_{i \in I} D_i = U$.

Conjunto Independente de Peso Máximo (CIPM)
Instância: um grafo G e uma função $w : V(G) \to \mathbb{Q}$ tal que $w(v)$ é denominado *peso* do vértice $v \in V(G)$.
Resposta: um conjunto independente de peso máximo, isto é, um conjunto independente $I \subseteq V(G)$ tal que $\sum_{v \in I} w(v)$ é máximo.

Corte mínimo de vértices
Instância: um grafo G.
Resposta: o menor conjunto K tal que K é um corte de vértices.

Capítulo 2

O Problema da Coloração de Grafos

Acredita-se que o estudo do problema de coloração de grafos originou-se no século XIX enquanto o matemático Francis Guthrie coloria o mapa da Inglaterra e observou que quatro cores eram suficientes para colorí-lo de maneira que regiões vizinhas não recebessem a mesma cor. Em 1976, enfim foi provado que todo mapa é colorível com no máximo 4 cores, o que foi enunciado como o Teorema das Quatro Cores (Appel e Haken, 1977; Appel et al., 1977). Contudo, vale salientar que O Teorema das Quatro Cores é aplicável apenas a classe de grafos *planares*, pois a coloração de mapas representa um caso específico da coloração de grafos.

O problema da coloração de grafos é da classe NP-Difícil (Karp, 1972). Aplicações como escalonamento de horários ou tarefas e alocação de frequências podem ser modelados como instâncias do problema de coloração de grafos.

Limitantes para o número cromático são importantes para funções de pré-processamento em algoritmos de coloração. A seguir são apresentados alguns destes limitantes.

2.1 Limitantes para o número cromático

O Algoritmo 1, baseado na descrição de Araújo Neto e Gomes (2014), obtém uma coloração gulosa para um grafo G. Os vértices são coloridos sequencialmente a partir de uma ordenação prévia destes. Atribui-se a cada vértice da ordenação a cor de menor índice possível não utilizada por um de seus vizinhos.

A complexidade do Algoritmo 1 é $O(n^2)$. A partir dele, é possível obter um limitante superior para o número cromático de G. Isto é, para cada vértice v_i tal que $i \in \{1, ..., n\}$, sabe-se que v_i tem no máximo $\Delta(G)$ vizinhos. Dessa forma, na execução do Algoritmo 1, ou é possível atribuir a v_i uma das cores já utilizadas até então, ou todos os seus vizinhos estão coloridos, e portanto, serão necessárias $\Delta(G) + 1$ cores para colorir o grafo até este ponto. Logo, $\chi(G) \leq \Delta(G) + 1$.

Limitantes inferiores, por sua vez, são obtidos em função do tamanho da maior clique de G e do maior conjunto independente.

Teorema 1. *Seja $\omega(G)$ o tamanho da maior clique de G. Então,*

$$\chi(G) \geq \omega(G)$$

Demonstração. Como todos os vértices da maior clique de G são vizinhos entre si, então cada um deve ser colorido com uma cor distinta. □

Algoritmo 1: COLORAÇÃO GULOSA(G)

Entrada: Um grafo G
Saída: Uma coloração ζ de G.
Início
$n \leftarrow |V(G)|$
I_i é um conjunto que representa uma cor de G, para $1 \leq i \leq n$.
$\zeta \leftarrow \emptyset$
Obtenha uma ordenação $(v_1, v_2, ..., v_n)$ para $V(G)$.

Para $i \leftarrow 1$ *até* n
  $I_i \leftarrow \emptyset$
Acrescente v_1 a I_1.
Para $i \leftarrow 2$ *até* n
  $k \leftarrow 1$
  Enquanto v_i *não tiver cor atribuída a si e* $k \leq n$
    Se $N_G(v_i) \cap I_k = \emptyset$ **então**
      Acrescente v_i a I_k.
      Acrescente I_k a ζ.
    Senão
      $k \leftarrow k + 1$
Devolva ζ

Teorema 2. *Seja $\alpha(G)$ o tamanho do maior conjunto independente de G. Então,*

$$\chi(G) \geq \frac{|V(G)|}{\alpha(G)}$$

Demonstração. Seja uma coloração ótima de G onde cada cor é um conjunto independente I_i para $1 \leq i \leq \chi(G)$. Como $|I_i| \leq \alpha(G)$, então

$$|V(G)| = \sum_{i=1}^{\chi(G)} |I_i| \leq \sum_{i=1}^{\chi(G)} \alpha(G) = \chi(G).\alpha(G)$$

□

2.2 Coloração em tempo polinomial

Conforme dito no início do capítulo, o problema da coloração de vértices é solucionável em tempo polinomial para determinadas classes de grafos, destacando-se a classe de grafos perfeitos, que são grafos onde $\chi(H) = \omega(H)$, para todo subgrafo induzido H de G. Berge (1961) conjecturou que para todo grafo perfeito G, tanto G quanto seu complemento não possuem ciclos induzidos de tamanho ímpar maior do que 3. Esta conjectura só foi provada 45 anos desde que foi enunciada (Teorema 3).

Teorema 3. *Chudnovsky et al. (2006) Um grafo G é perfeito se e somente se G e seu complemento não possuem ciclos induzidos de tamanho ímpar maior que 3.*

A classe dos grafos perfeitos é ampla. Nela incluem-se os grafos bipartidos e cordais, por exemplo. Os problemas de coloração de grafos, clique máxima e conjunto independente

máximo podem ser resolvidos em tempo polinomial em grafos perfeitos (Grötschel et al., 1984). Além disso, há algoritmo polinomial que reconhece se um grafo é perfeito ou não (Cornuejols et al., 2003).

2.2.1 Grafos Cordais

Um grafo é cordal se e somente se não admite nenhum ciclo induzido de tamanho maior que 3. Assim, todo subgrafo induzido de um grafo cordal também é cordal. Dessa forma, pelo Teorema 3, obviamente grafos cordais são perfeitos.

Grafos cordais possuem a propriedade de conterem uma *ordem de eliminação perfeita* em sua estrutura; isto é, uma permutação de vértices $(v_1, ..., v_n)$ tal que para todo $i \in \{1, 2, ..., n\}$, $N_G(v_i)$ é uma clique no grafo $G - \{v_1, ..., v_{i-1}\}$ e todo vértice v_i neste grafo é denominado *simplicial*.

Teorema 4. *Todo grafo cordal G ou é completo ou possui pelo menos dois vértices simpliciais.*

Demonstração. (Bondy e Murty (2008)) Se G é completo, então todo vértice de G é simplicial, pois todo subgrafo induzido de G também é completo.

Por outro lado, se G não é completo, seja X um corte mínimo de vértices de G de maneira que os vértices de $G - X$ estão particionados em dois conjuntos X_1 e X_2. Assim, $G[X]$ deve ser um subgrafo completo de G. Supondo por contradição que isto não ocorresse, seja um par de vértices $x, y \in X$ não adjacentes entre si. Como X é um corte mínimo, então x e y estão conectados a vértices de X_1 e X_2. Dessa forma, seja $a \in X_1$ tal que $\{a, x\} \in E(G)$ e $b \in X_2$ tal que $\{b, y\} \in E(G)$. Se os menores caminhos de x a y são $P_1 = (x, a, y)$ e $P_2 = (x, b, y)$, ao juntarem-se P_1 e P_2 haverá um ciclo induzido de tamanho 4 em G. Porém, isto contraria o fato de que G é cordal. Portanto, X é uma clique e a e b são dois vértices simpliciais contidos em G. □

Teorema 5. *Um grafo tem uma ordem de eliminação perfeita se e somente se é cordal.*

Demonstração. Seja G um grafo e $(v_1, ..., v_n)$ uma ordem de eliminação perfeita. Seja também C um ciclo induzido em G de tamanho maior que 3. O primeiro vértice pertencente a C que aparece na ordem de eliminação perfeita de G, denotado v, tem no mínimo dois vizinhos u e w. Como C é induzido, então não existe aresta entre u e w, e assim, eles não formam um K_3. Por outro lado, como existe uma ordem de eliminação perfeita em G, então na verdade deve haver uma aresta entre u e w, o que contraria o fato de que C é um ciclo induzido de tamanho maior que 3. Logo, se G tem uma ordem de eliminação perfeita, então G é cordal.

Para provar que se G é cordal então G tem uma ordem de eliminação perfeita, sabemos que se G é cordal, então G tem pelo menos dois vértices simpliciais, conforme provado no Teorema 4.

Seja v um destes vértices simpliciais. Dessa forma, $G - v$ é subgrafo induzido de G que também é cordal, e pelo Teorema 4, $G - v$ também possui ao menos dois vértices simpliciais. Assim, retirando-se sucessivamente um dos vértices simpliciais do grafo $G - v$ até que todos os vértices sejam retirados de G, obter-se-á uma ordem de eliminação perfeita. □

O algoritmo de busca em largura lexicográfica (LexBFS) proposto por Rose et al. (1976) encontra uma ordem de eliminação perfeita em grafos cordais. Além disso, a partir da ordenação gerada pela execução do LexBFS em um grafo G, o algoritmo dos autores verifica se esta ordenação é perfeita. Com isso, é possível também reconhecer se um grafo G é cordal. Portanto, como o LexBFS é polinomial, então é possível verificar em tempo polinomial se um grafo é cordal ou não.

A seguir, será demonstrado como o Algoritmo 1 é exato para grafos cordais. A demonstração é a mesma apresentada em Lewis (2016).

Teorema 6. *O algoritmo de coloração gulosa (Algoritmo 1) encontra uma solução ótima para grafos cordais com valor $\chi(G) = \omega(G)$ a partir de uma ordem perfeita de eliminação.*

Demonstração. Aplicando-se o Algoritmo 1 sobre um grafo cordal, temos que para a ordem de eliminação perfeita $(v_1, ..., v_n)$, todo vértice v_i é colorido com a cor que possui o menor índice e ainda não foi utilizada por seus vizinhos. Como cada v_i está contido em uma clique no grafo $G[\{v_1, ..., v_i\}]$ e o tamanho da maior clique maximal de G é $\omega(G)$, então pelo menos uma das cores do conjunto $\{1, ..., \omega(G)\}$ poderá ser atribuída a v_i. Portanto, $\chi(G) \leq \omega(G)$. Contudo, como $\omega(G) \leq \chi(G)$, então $\chi(G) = \omega(G)$. □

Como será visto nos próximos capítulos, alguns algoritmos de coloração utilizam rotinas de pré-processamento em sua estrutura. Estes algoritmos buscam por subgrafos induzidos do grafo de entrada que possam ser coloridos eficientemente, de maneira que bons limitantes inferiores sejam encontrados. Como dito anteriormente, tanto identificar grafos perfeitos bem como colori-los são problemas solucionáveis em tempo polinomial. Desta forma, pode-se pensar em rotinas de pré-processamento que funcionem desta maneira. Contudo, notou-se também que a maioria dos resultados presentes na literatura para o problema de coloração de grafos perfeitos contém apenas análise de pior caso, tendo-se o trabalho de Chudnovsky et al. (2017) como o mais recente.

Capítulo 3

Algoritmos baseados em Programação Dinâmica

De acordo com Kreher e Stinson (1999), algoritmos baseados em programação dinâmica computam e armazenam em uma tabela de consulta a solução ótima de estruturas menores de uma instância para chegar a solução ótima do problema total. Dentre aqueles existentes na literatura, o apresentado por Byskov (2002) possui o melhor desempenho de pior caso para encontrar o número cromático de um grafo.

A ideia do algoritmo de Byskov (2002) tem como partida o trabalho de Lawler (1976), que foi o primeiro a formular o problema de coloração de vértices aplicando programação dinâmica, desenvolvendo algoritmos para testar se um grafo é 3-colorível em tempo $O(1.4422^n)$ e para computar o número cromático em tempo $O(2.4422^n)$. Estes resultados foram melhorados por Beigel e Eppstein (2005), que alcançaram tempo $O(1.3289^n)$ em seu algoritmo de 3-coloração, e por Eppstein (2001), que alcançou tempo $O(2.4150^n)$ em seu algoritmo que determina o número cromático e a respectiva coloração ótima de um grafo. Por fim, tem-se o trabalho de Byskov (2002), que alcançou tempo $O(2.4023^n)$ no problema do número cromático.

Todos os trabalhos supracitados utilizam espaço exponencial. Dessa forma, Bodlaender e Kratsch (2006) interessaram-se em desenvolver um algoritmo que utiliza espaço polinomial para a coloração de grafos, alcançando tempo $O(5.283^n)$.

3.1 O Algoritmo de Lawler

O primeiro algoritmo exato baseado em programação dinâmica para resolver o problema de coloração foi desenvolvido por Lawler (1976). Além de computar o número cromático de um grafo por meio deste método, o autor também apresenta resultados para um algoritmo que testa se um grafo é 3-colorível.

3.1.1 Algoritmo para o Número Cromático

Baseando-se em Wang (1974), Lawler (1976) observa que para alguma k-coloração de um grafo G tal que $\chi(G) = k$, então pelo menos uma das classes de cores pertencentes a esta k-coloração é um conjunto independente maximal.

Teorema 7. *(Wang (1974)) Dado um grafo G, para todo vértice $v \in V(G)$, seja $\{I_1, ..., I_l\}$ os conjuntos independentes maximais que contém v. Então existe uma k-coloração ótima de G tal que uma de suas cores é I_i para algum $1 \leq i \leq l$.*

Demonstração. Seja $\{P_1, ..., P_k\}$ uma coloração ótima de G. Sem perda de generalidade, seja P_1 um conjuto não maximal contido em I_i, para algum $i \in \{1, ..., l\}$. Sejam $\{P'_2, ..., P'_k\}$ os conjuntos obtidos da remoção dos vértices de I_i dos conjuntos $\{P_2, ..., P_k\}$, respectivamente. Dessa forma, $\{I_i, P'_2, ..., P'_k\}$ é coloração ótima de G. □

Teorema 8. *(Lawler (1976)) Seja S um dos 2^n subconjuntos de vértices possíveis de G, tal que $n = |V(G)|$. Então para cada subgrafo $G[S]$, tal que $S \neq \emptyset$, existe uma coloração onde ao menos um conjunto independente maximal I em $G[S]$ é tal que*

$$\chi(G[S]) = 1 + \chi(G[S \setminus I]) \tag{3.1}$$

Demonstração. Seja $S \subseteq V(G)$ e $\chi(G) = k$. Como consequência do Teorema 7, para todo grafo G, há uma k-coloração de G onde ao menos uma das cores é um conjunto independente maximal. Dessa forma, se o número cromático de $G[S]$ é igual a k, então $\chi(G[S \setminus I])$ é $k - 1$, pois I é conjunto independente maximal que também é uma cor de $G[S]$. □

Teorema 9. *Para todo $S \subseteq V(G)$, o número cromático de $G[S]$ é dado pela recorrência*

$$\chi(G[S]) = \begin{cases} 0, & \text{se } S = \emptyset \\ 1 + \min\{\chi(G[S \setminus I])\}, \text{ para todo conjunto independente } I \subseteq G[S] & \text{se } S \neq \emptyset \end{cases} \tag{3.2}$$

Demonstração. Será provado que $\chi(G[S]) \leq 1 + \chi(G[S \setminus I])$ para todo conjunto independente $I \subseteq G[S]$, visto que a recursão pode ser expressa dessa forma.

Seja $G[S \setminus I]$ colorível com k cores, isto é, $\chi(G[S \setminus I]) = k$. Então, há duas possibilidades para I:

- se para algum $i \in I$ não for possível atribuir-lhe a mesma cor de algum vértice de $S \setminus I$, isto é, se k cores são utilizadas pelos vizinhos já coloridos de i, então i deve receber uma nova cor, e portanto, $\chi(G[S]) = 1 + k = 1 + \chi(G[S \setminus I])$;
- se para todo $i \in I$, é possível atribuir-lhe uma das k cores de $S \setminus I$, então $\chi(G[S])$ não se altera, isto é, $\chi(G[S]) = \chi(G[S \setminus I])$.

Assim, conclui-se que $\chi(G[S]) \leq 1 + \chi(G[S \setminus I])$ para todo conjunto independente $I \subseteq G[S]$. □

Byskov (2002) descreveu um pseudocódigo para a recorrência acima (Algoritmo 2).

Seja S um subconjunto de vértices de $V(G)$ de modo que para $V(G) = \{v_0, ..., v_{n-1}\}$, S está representado como um vetor binário de n bits. Cada um dos n bits desta representação indica se o vértice v_b, para $b \in \{0, ..., n - 1\}$, está contido no referido subconjunto ou não. Podemos dizer ainda que o conjunto S é

$$\sum_{v_i \in S} 2^i$$

Por exemplo, seja o conjunto de vértices $\{v_0, v_1, v_2\}$. Assim, a representação binária de todo conjunto $S \subseteq V(G)$ é um vetor de tamanho 3, e a associação de cada S ao seu respectivo valor decimal é dada por:

$0 \rightarrow \emptyset$ $\quad$ $4 \rightarrow \{v_2\}$
$1 \rightarrow \{v_0\}$ $\quad$ $5 \rightarrow \{v_0, v_2\}$
$2 \rightarrow \{v_1\}$ $\quad$ $6 \rightarrow \{v_1, v_2\}$
$3 \rightarrow \{v_0, v_1\}$ $\quad$ $7 \rightarrow \{v_0, v_1, v_2\}$

Algoritmo 2: $\chi(G)$ - LAWLER

Entrada: Um grafo G
Saída: O número cromático de G.
Início
  $X \leftarrow$ array indexado de 0 a $2^n - 1$
  $S \leftarrow 0$ //conjunto vazio
  $X[S] \leftarrow 0$
  Para $S \leftarrow 1$ *até* $2^n - 1$
    $X[S] \leftarrow \infty$
    Para *todo conjunto independente maximal* $I \subseteq G[S]$
      $X[S] \leftarrow \min\{X[S], X[S \setminus I] + 1\}$
Devolva $X[2^n - 1]$ *//V(G)*

O algoritmo percorre todos os 2^n subgrafos induzidos G, de maneira que os subconjuntos de S sejam processados antes de S. Para cada S, computam-se os seus conjuntos independentes maximais e $X[S]$ conforme o Teorema 9.

Teorema 10. *O algoritmo de Lawler (1976) para o problema do número cromático executa em tempo* $O(nm2.4423^n)$ *e espaço* $\Theta(2^n)$*, para* $n = |V(G)|$ *e* $m = |E(G)|$.

Demonstração. O número máximo de conjuntos independentes maximais em um grafo é igual a $3^{n/3}$ (Moon e Moser, 1965). Lawler (1976) cita o trabalho de Tsukiyama et al. (1977) como referência de algoritmo que executa em tempo $O(nm3^{n/3})$ para geração dos conjuntos independentes maximais.

Computar o conjunto de conjuntos independentes maximais contidos em $G[S]$, denotado $I(G[S])$, para todo o grafo G é igual a

$$\sum_{S \subseteq V(G)} |I(G[S])| \leq \sum_{i=0}^{n} \binom{n}{i} nm3^{i/3} = nm \sum_{i=0}^{n} \binom{n}{i} 3^{i/3} = nm(1 + 3^{1/3})^n < nm2.4423^n$$

Logo, o Algoritmo 2 executa em espaço $\Theta(2^n)$ e tempo $O(nm2.4423^n)$. □

3.1.2 Algoritmo para a 3-coloração

O algoritmo de Lawler (1976) que testa se um grafo é 3-colorível é simples, pois para cada conjunto independente maximal $I \subseteq V(G)$, é verificado se $G - I$ é um grafo bipartido (Beigel e Eppstein, 2005). Este passo pode ser feito percorrendo-se $G - I$ por meio do algoritmo de busca em largura. Como este é linear em relação ao tamanho da entrada (Cormen et al., 2009), e um grafo tem no máximo $(3^{1/3})^n < 1.4423^n$ conjuntos independentes maximais, então o algoritmo de 3-coloração de Lawler (1976) tem tempo $O(nm1.4423^n)$.

3.2 Algoritmos de Beigel e Eppstein

O algoritmo de Lawler (1976) que encontra o número cromático de um grafo permaneceu como o de melhor desempenho de pior caso por 25 anos até que Eppstein (2001) realizasse a primeira adaptação deste algoritmo. O autor propôs que os subgrafos 3-coloríveis do grafo de entrada fossem pré-processados antes de aplicar-se o algoritmo que encontra o número cromático. O algoritmo utilizado no passo de pré-processamento é o que contém os melhores resultados de pior caso na literatura para o problema da 3-coloração. Este foi proposto pelos autores Beigel e Eppstein (2005), e será brevemente apresentado a seguir.

3.2.1 O Algoritmo de Beigel e Eppstein para a 3-coloração

O algoritmo de Beigel e Eppstein (2005) reduz uma instância do problema de 3-coloração a uma instância do Problema de Satisfação de Restrições (*Constraint Satisfaction Problem* - CSP), mais especificamente o problema $(3, 2)$-CSP.

Dados $a, b \in \mathbb{N}$, definimos o problema (a, b)-CSP como abaixo:

Problema (a,b)-CSP
Instância: uma tripla (X, D, R), onde X, D e R são conjuntos finitos disjuntos. X é denominado conjunto de variáveis, D é o conjunto de valores que serão atribuídos às variáveis e R é o conjunto de restrições. Uma restrição é um par (t, f), onde t é uma b-upla de variáveis e f é uma relação de b valores de D.
Resposta: uma valoração às variáveis de tal forma que todas as restrições sejam satisfeitas.

No problema (a, b)-CSP, o valor que será atribuído a cada variável está limitado a no máximo a elementos de D, e cada restrição descreve a combinação de valores que cada b-upla não pode ter simultaneamente.
Antes de exemplificar uma instância (a, b)-CSP, definimos o conceito de *multigrafo*.

Definição 1. *Um* multigrafo G *é uma tripla* $(V(G), E(G), J)$*, tal que* $V(G)$ *é o conjunto de vértices,* $E(G)$ *é o conjunto de arestas e* J *é uma função que associa cada elemento* $e \in E(G)$ *a um elemento de* $\binom{V(G)}{2}$*. As arestas* $e, e' \in E(G)$ *são denominadas* paralelas *se* $J(e) = J(e')$.

Como dito anteriormente, uma instância do problema de coloração de grafos pode ser reduzida a uma instância de $(3, 2)$-CSP, que por sua vez, é uma instância de $(a, 2)$-CSP. Isto é, instâncias que podem ser descritas como multigrafos onde os vértices estão associados as variáveis de X, e cada vértice é composto pelo conjunto de no máximo a valores que podem ser atribuídos a variável a qual está associado. Cada aresta conecta pares de valores incompatíveis entre si.

A seguir, temos um exemplo de redução do problema de 3-coloração ao $(3, 2)$-CSP para melhor compreensão. No exemplo, cada vértice do grafo de entrada é uma variável do problema restrita a no máximo 3 valores do domínio $D = \{0, 1, 2\}$, tal que cada valor é uma cor. A Figura 1 ilustra este exemplo, onde a bolinha preta representa a cor 0, a bolinha branca representa a cor 1, e a bolinha estampada representa a cor 2. Cada aresta do multigrafo representa uma restrição entre cores incompatíveis com um par de vértices. Dessa forma, para a instância inicial de uma 3-coloração, criam-se três restrições para cada aresta do grafo original:

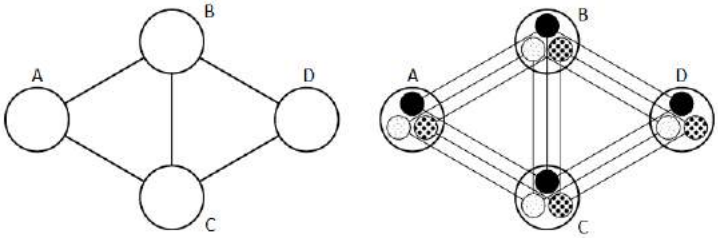


Figura 1: Exemplo do problema de 3-coloração (adaptado de Beigel e Eppstein (2005))

- variáveis: $\{A, B, C, D\}$
- domínio: $\{0, 1, 2\}$
- restrições:
  $\{(A = 1, B = 1), (A = 2, B = 2), (A = 3, B = 3),$
  $(A = 1, C = 1), (A = 2, C = 2), (A = 3, C = 3),$
  $(B = 1, C = 1), (B = 2, C = 2), (B = 3, C = 3),$
  $(B = 1, D = 1), (B = 2, D = 2), (B = 3, D = 3),$
  $(C = 1, D = 1), (C = 2, D = 2), (C = 3, D = 3)\}$

Os autores apresentam um algoritmo de complexidade $O(1.3645^n)$ baseado em *backtracking* para resolução do $(3, 2)$-CSP, bem como simplificações de tempo polinomial que podem reduzir o tamanho de uma instância desse problema. Uma destas simplificações está descrita no Lema 1. A demonstração é a mesma apresentada pelos autores.

Lema 1. *(Beigel e Eppstein (2005)) Dada uma instância (X, D, R) do $(a, 2)$-CSP, seja $v \in X$ uma variável limitada a somente dois valores do domínio. Existe uma instância (X', D', R') obtida a partir de (X, D, R), com uma variável a menos, onde (X', D', R') não contém v e que qualquer solução ótima para (X', D', R') é válida para (X, D, R).*

Demonstração. Seja a instância (X, D, R) do $(a, 2)$-CSP e sejam $x, y \in X$. Seja v uma variável limitada a somente dois valores, denominados h e i. Sem perda de generalidade, sejam as restrições $\{(y = w), (v = h)\}$ e $\{(x = z), (v = i)\}$, tais que w e z são dois valores quaisquer pertencentes ao domínio. Se y for valorado com w e x com z, então não haverá valor possível a ser atribuído a v. Logo, adicionando-se a restrição $\{(y = w), (x = z)\}$, garante-se que isso não ocorrerá. Consequentemente, uma solução para (X', D', R') com a nova restrição e contendo uma variável a menos poderá ser estendida para (X, D, R) atribuindo-se h ou i a v. □

A Figura 2 ilustra o Lema 1. Neste exemplo, $\{0, 1, 2\}$ é o conjunto que representa o domínio, tal que a bolinha preta representa a cor 0, a bolinha branca representa a cor 1, e a bolinha estampada representa a cor 2.

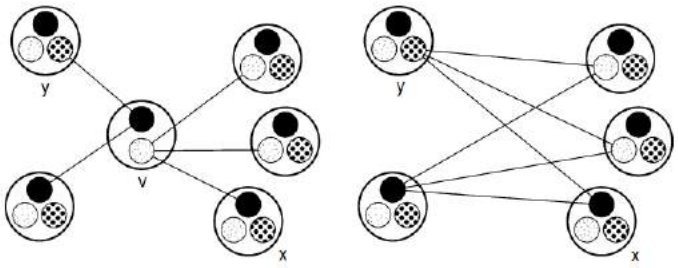


Figura 2: Exemplo da simplificação

A principal ideia do algoritmo de Beigel e Eppstein (2005) para 3-coloração consiste em, dado um grafo G, encontrar um subconjunto $T \subset V(G)$ tal que T seja uma árvore, pois supondo-se que o grafo de entrada é 3-colorível, garante-se que cada vizinho dos vértices de T estará limitado a 1 ou 2 valores do domínio. Além disso, T é tal que seu tamanho é pequeno em relação a quantidade total de vértices do grafo e possui um conjunto grande de vizinhos, denotado N. Colore-se todos os vértices de T escolhendo-se uma dentre $3^{|T|}$ colorações próprias. Desta forma, para cada $v \in N$ há no máximo duas cores possíveis restantes, pois seus vizinhos em T já estão coloridos. Pelo Lema 1, os vértices de N podem sem removidos da instância de entrada. Assim, o subgrafo induzido pelos vértices não coloridos que restaram formam uma instância do $(3, 2)$-CSP de tamanho consideravelmente reduzido para ser solucionada pelo algoritmo sugerido pelos autores.

A complexidade de tempo do algoritmo de Beigel e Eppstein (2005) é $O(1.3289^n)$ e, conforme dito no início desta Seção, é o que atualmente apresenta os melhores resultados de pior caso para o problema da 3-coloração.

3.2.2 O Algoritmo de Eppstein para o Número Cromático

Assim como o algoritmo de Lawler (1976), o algoritmo de Eppstein (2001) também percorre todos os possíveis subconjuntos de vértices $S \subseteq V(G)$ e armazena seu número cromático num vetor X. Contudo, ao invés de percorrer todos os conjuntos independentes maximais contidos em S e computar o valor $\chi(G[S]) = 1 + \min\{\chi(G[S \setminus I])\}$, são percorridos todos os conjuntos independentes maximais contidos em $G - S$ e é computado o número cromático de $G[S \cup I]$, isto é, $\chi(G[S \cup I]) = \min\{\chi(G[S \cup I]), \chi(G[S]) + 1\}$, para todo conjunto independente maximal I. Conforme observado pelo autor, esta inversão por si só não influi no tempo de execução do algoritmo de programação dinâmica. Entretanto, isto evita que todos os conjuntos independentes maximais sejam percorridos, pois seu tamanho pode ser limitado superiormente, conforme explicação abaixo (baseada em Byskov (2004) e Marin (2005)).

Lema 2. *(Madsen et al. (2002)) Seja o conjunto $S' \subseteq V(G)$ tal que $G[S']$ é um subgrafo k-colorível maximal de G. Então, existe um subgrafo $G[S]$ que é k_1-colorível maximal, para $S \subseteq S'$ e para todo $0 \leq k_1 < k$, tal que $G[S' \setminus S]$ é um subgrafo $(k - k_1)$-colorível maximal de $G - S$.*

Demonstração. Seja k_1 um valor qualquer tal que $0 \leq k_1 < k$. Para cada k-coloração ótima de $G[S']$, sejam os vértices de $G[S']$ separados em cores $I_1, I_2, ..., I_k$ tal que $|I_1| \geq |I_2| \geq ... \geq |I_{k_1}| \geq ... \geq |I_k|$. Seja a coloração escolhida aquela onde as classes $I_1, I_2, ..., I_{k_1}$ formam o maior conjunto de vértices possível. Assim, estas classes constituem o conjunto $S \subset S'$, tal que $G[S]$ é k_1-colorível, e por construção, $G[S]$ é maximal em $G[S']$. Se houver algum vértice $v \in V(G) \setminus S'$ que puder ser adicionado a $G[S]$ ou $G[S' \setminus S]$ de modo que $G[S' \cup \{v\}]$ permaneça k-colorível, então é contraditório que $G[S']$ seja maximal. Portanto, $G[S]$ é k_1-colorível maximal em G e $G[S' \setminus S]$ é $(k - k_1)$-colorível maximal em $G - S$. □

Corolário 1. *Seja o conjunto $S' \subseteq V(G)$ tal que $G[S']$ é um subgrafo k-colorível maximal de G. Então, existe S tal que o subgrafo $G[S]$ é $(k - 1)$-colorível maximal, para $S \subseteq S'$, tal que $G[S' \setminus S]$ é um conjunto independente maximal de $G - S$.*

Demonstração. Para cada k-coloração ótima de $G[S']$, sejam os vértices de $G[S']$ separados em cores $I_1, I_2, ..., I_k$ tal que $|I_1| \geq |I_2| \geq ... \geq |I_k|$. Seja a coloração escolhida aquela onde a classe I_k obtém o menor tamanho possível. Assim, ao remover I_k de S' obtemos o conjunto $S = S' \setminus I_k$, tal que $G[S]$ é $(k - 1)$-colorível, e por construção, $G[S]$ é maximal em $G[S']$. Logo, pelo Lema 2 e fazendo $k_1 = k - 1$, tem-se que $G[S' \setminus S]$ é um conjunto independente maximal de $G - S$. □

A partir dos resultados acima, podemos entender por que é possível limitar o tamanho dos conjuntos independentes maximais a serem examinados em $G - S$.

Como $G[S']$ é um subgrafo k-colorível maximal, sabemos que existe uma classe de cor I_k em uma de suas colorações ótimas tal que I_k é a de menor tamanho possível. Pelo Corolário 1, sabemos também que o conjunto $S = S' \setminus I_k$ é tal que $G[S]$ é $(k-1)$-colorível e I_k é conjunto independente maximal de $G - S$.

Analogamente a S', temos que $G[S]$ também possui uma ou várias $(k-1)$-colorações ótimas, e dentre estas, existe uma cor I_{k-1} que possui o menor número de vértices possível. Assim, sejam as cores $I_1, I_2, ..., I_{k-1}$ referentes a esta coloração. Temos que

$$|S| = \sum_{i=1}^{\chi(G[S])} |I_i| \geq \sum_{i=1}^{\chi(G[S])} |I_{k-1}| = |I_{k-1}|\chi(G[S])$$

e portanto

$$|I_{k-1}| \leq \frac{|S|}{\chi(G[S])}$$

Como a cor I_{k-1} será sempre maior ou igual que I_k, e I_{k-1} é limitada superiormente por $\frac{|S|}{\chi(G[S])}$, então $|I_k| \leq \frac{|S|}{\chi(G[S])}$.

Portanto, no algoritmo de Eppstein (2001), encontrar apenas cada conjunto independente maximal I de $G - S$ para cada conjunto $S \subseteq V(G)$, tal que $I \leq \frac{|S|}{\chi(G[S])}$, mantém a corretude de $\chi(G[S \cup I]) = \min\{\chi(G[S \cup I]), \chi(G[S]) + 1\}$.

Outra mudança do algoritmo de Eppstein (2001) em relação ao de Lawler (1976) diz respeito ao pré-processamento dos subgrafos de G que são 3-coloríveis. Nesta etapa, é utilizado o algoritmo de Beigel e Eppstein (2005), introduzido na subseção 3.2.1.

Temos uma descrição para o algoritmo de Eppstein (2001) (Algoritmo 3), onde S é um subconjunto de $V(G)$ representado tal como no Algoritmo 2, X é o vetor que armazena o número cromático de S e $c(S)$ é uma função que associa cada subconjunto de vértices de $V(G)$ a sua representação binária S.

Lema 3. *(Croitoru, 1979) O número de conjuntos independentes maximais de tamanho até k em um grafo G é*

$$\lfloor n/k \rfloor^{(\lfloor n/k \rfloor + 1)k - n} (\lfloor n/k \rfloor + 1)^{n - \lfloor n/k \rfloor k}$$

Teorema 11. *O algoritmo de Eppstein (2001) para o problema do número cromático executa em tempo $O(2.4150^n)$ e espaço $O(2^n)$.*

Demonstração. (Byskov (2002))

A parte onde o vetor X é inicializado é executada em tempo $O(2.3289^n)$, pois como o algoritmo de Beigel e Eppstein (2005) é executado em tempo $O(1.3289^n)$, então encontrar os subgrafos $G[S]$ que são 3-coloríveis é

$$\sum_{S \subseteq V(G)} 1.3289^{|S|} \leq \sum_{i=0}^{n} \binom{n}{i} 1.3289^i = (1 + 1.3289)^n = O(2.3289^n)$$

Na segunda parte temos que no pior caso, se o valor do número cromático de $G[S]$ é maior ou igual a 3, então o maior valor de k é atingido em $|S|/3$. Portanto, pelo Teorema 3,

$$\lfloor n/(n/3) \rfloor^{(\lfloor n/(n/3) \rfloor + 1)k - n} (\lfloor n/(n/3) \rfloor + 1)^{n - \lfloor n/(n/3) \rfloor k} = O(3^{4k-n} 4^{n-3k})$$

Algoritmo 3: $\chi(G)$ - EPPSTEIN

Entrada: Um grafo G
Saída: O número cromático de G.
Início
  $n \leftarrow |V(G)|$
  $X \leftarrow$ vetor indexado de 0 a $2^n - 1$
  $S \leftarrow 0$ //conjunto vazio
  $X[S] \leftarrow 0$
  Para $S \leftarrow 1$ *até* $2^n - 1$
    Execute o algoritmo de Beigel e Eppstein (2005) em $G[S]$.
    Se $\chi(G[c(S)]) \leq 3$ **então**
      $X[S] \leftarrow \chi(G[c(S)])$
    Senão
      $X[S] \leftarrow \infty$
  Para $S \leftarrow 1$ *até* $2^n - 1$
    Se $3 \leq X[S] < \infty$ **então**
      Para *todo conjunto independente maximal* $I \subseteq G - S$ *tal que* $|I| \leq |S|/X[S]$
        $X[S \cup I] \leftarrow \min\{X[S \cup I], X[S] + 1\}$
Devolva $X[2^n - 1]$ //$V(G)$

Utilizando-se de *backtracking* para gerar os conjuntos independentes maximais de tamanho até k de G, Eppstein (2001) prova que este passo é executado em tempo no máximo $O(3^{4k-n}4^{n-3k})$. O número máximo de conjuntos independentes maximais de tamanho até k é alcançado quando o grafo é formado por uma união disjunta de $4k - n$ K_3 e $n - 3k$ K_4 (Figura 3).

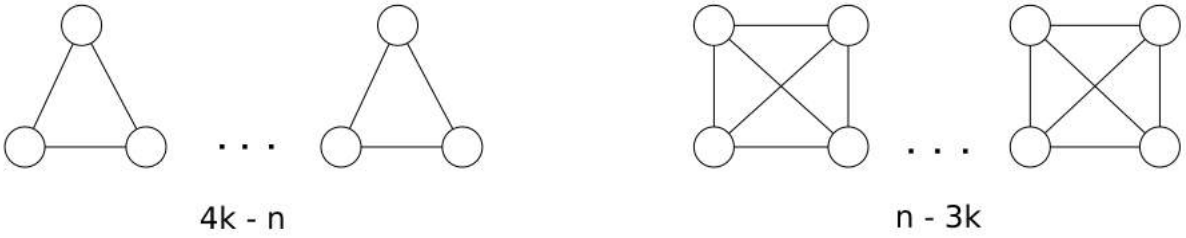


Figura 3: Exemplo de grafo que alcança o número máximo de conjuntos independentes de tamanho até k

Finalmente, temos que

$$\sum_{S \subseteq V(G)} 3^{\frac{4|S|}{3} - (n-|S|)} 4^{(n-|S|) - 3\frac{|S|}{3}} = \sum_{S \subseteq V(G)} 3^{\frac{7|S|}{3} - n} 4^{n-2|S|} \leq$$

$$\leq \sum_{i=0}^{n} \binom{n}{i} 3^{\frac{7i}{3} - n} 4^{n-2i} = \sum_{i=0}^{n} \binom{n}{i} \frac{3^{\frac{7i}{3}}}{3^n} \frac{4^n}{4^{2i}} =$$

$$= \left(\frac{4}{3}\right)^n \sum_{i=0}^{n} \binom{n}{i} \frac{3^{\frac{7i}{3}}}{4^{2i}} = \left(\frac{4}{3}\right)^n \sum_{i=0}^{n} \binom{n}{i} \left(\frac{3^{\frac{7}{3}}}{4^2}\right)^i =$$

$$= \left(\frac{4}{3}\right)^n \left(1 + \frac{3^{\frac{7}{3}}}{4^2}\right)^n = \left(\frac{4}{3} + \frac{4.3^{\frac{7}{3}}}{3.4^2}\right)^n =$$

$$= \left(\frac{4}{3} + \frac{3^{4/3}}{4}\right)^n = O(2.4150^n)$$

□

3.2.3 O Algoritmo de Eppstein que determina a Coloração Ótima

O algoritmo que encontra uma coloração ótima em G (Algoritmo 4) percorre os subconjuntos de cada $T \subset V(G)$ em ordem decrescente, removendo os subgrafos de G que são $(k-1)$-coloríveis maximais sucessivamente até que se encontre o conjunto vazio.

Seja S um conjunto que está contido em $V(G)$, tal que $\chi(G[S]) = k$. Em cada iteração, o algoritmo busca um conjunto T tal que $G[T]$ seja $(k-1)$-colorível maximal, pois pelo Lema 2, os vértices em $S \setminus T$ formam um conjunto independente maximal I. Deste modo, uma vez encontrado T, o conjunto I é removido de $G[S]$ e repetem-se estes passos até que o conjunto vazio seja atingido.

Sejam T e S subconjuntos de vértices representados como vetores binários como feito nos algoritmos de Lawler (1976) e Eppstein (2001). Todo conjunto $T \subseteq T'$ é processado depois de T'; isto é, os subconjuntos de vértices de $V(G)$ são percorridos de maneira inversa à dos algoritmos anteriores.

Algoritmo 4: COLORAÇÃO(G) - EPPSTEIN

Entrada: Um grafo G
Saída: Uma coloração ótima de G
Início
  $X \leftarrow$ array calculado como no Algoritmo 3
  $S \leftarrow V(G)$
  Para $T \leftarrow 2^n - 1$ *até* 0
    Se $T \subset S$ *e* $X[S \setminus T] = 1$ *e* $X[T] = X[S] - 1$ **então**
      Atribua a mesma cor a cada vértice de $S \setminus T$.
      $S \leftarrow T$

Como o *loop* do algoritmo tem tempo $O(2^n)$ e as operações dentro deste tomam tempo constante, então o maior gasto computacional está em computar X no Algoritmo 3, que por sua vez, tem tempo $O(2.4150^n)$.

3.3 O Algoritmo de Byskov

Byskov (2002) alcança ganho computacional sobre o resultado de Eppstein (2001). Assim como nos trabalhos supracitados, no algoritmo de Byskov (2002) (Algoritmo 5) há um vetor X que armazena $\chi(G[S])$ para cada $S \subseteq V(G)$. Assim como no Algoritmo 3, o vetor X é inicializado com a marcação dos subconjuntos 3-coloríveis de G em tempo $O(1.3289^n)$.

Aquilo que diferencia seu algoritmo de seus predecessores está na marcação dos subgrafos 4-coloríveis em seguida ao passo da inicialização de X, tal como previamente sugerido por Eppstein (2001). Para cada conjunto independente maximal I, tem-se que para cada

subconjunto de vértices $S \subseteq G - I$, se $\chi(S) = 3$, então $\chi(G[S \cup I]) \leq 4$. O algoritmo é finalizado tal como no Algoritmo 3, isto é, encontrando-se para todo $S \subseteq V(G)$ o número cromático do superconjunto $G[S \cup I]$, para $I \subseteq G - S$.

Novamente, no Algoritmo 5 cada conjunto $S \subseteq V(G)$ está representado como um vetor binário da mesma forma que nos algoritmos prévios, e a função $c(S)$ associa cada S ao subconjunto de vértices que representa.

Algoritmo 5: $\chi(G)$ - Byskov

Entrada: Um grafo G
Saída: O número cromático de G
Início
  $n \leftarrow |V(G)|$
  $X \leftarrow$ array indexado de 0 a $2^n - 1$
  $S \leftarrow 0$ //conjunto vazio
  $X[S] \leftarrow 0$
  Para $S \leftarrow 1$ *até* $2^n - 1$
    Execute o algoritmo de Beigel e Eppstein (2005) em $G[S]$.
    Se $\chi(G[c(S)]) \leq 3$ **então**
      $X[S] \leftarrow \chi(G[c(S)])$
    Senão
      $X[S] \leftarrow \infty$
  Para *todo conjunto independente maximal* $I \subseteq G$
    Para *todo* $S \subseteq (V(G) \setminus I)$
      Se $X[S] = 3$ **então**
        $X[S \cup I] \leftarrow \min\{X[S \cup I], 4\}$
  Para $S \leftarrow 1$ *até* $2^n - 1$
    Se $4 \leq X[S] < \infty$ **então**
      Para *todo conjunto independente maximal* $I \subseteq G - S$ *tal que* $|I| \leq |S|/X[S]$
        $X[S \cup I] \leftarrow \min\{X[S \cup I], X[S] + 1\}$
Devolva $X[2^n - 1]$ //$V(G)$

Teorema 12. *O algoritmo de Byskov (2002) para o problema do número cromático executa em tempo* $O(2.4023^n)$ *e espaço* $O(2^n)$.

Demonstração. Seja $I_k(G)$ o conjunto de todos conjuntos independentes maximais de G de tamanho até k, e seja $I_k \in I_k(G)$. Encontrar os subgrafos $G[S]$ que são 4-coloríveis tem tempo

$$\sum_{I \subseteq G} \sum_{S \subseteq (V(G) \setminus I)} 1 = \sum_{k=1}^{n} \sum_{I_k \in I_k(G)} \sum_{S \subseteq (V(G) \setminus I_k)} 1 = \sum_{k=1}^{n} |I_k(G)| 2^{n-k}$$

Byskov (2004) provou que o limitante superior para o número de conjuntos independentes maximais de tamanho até k é igual a $d^{(d+1)k-n}(d+1)^{n-dk}$ para qualquer valor de $d \geq 3$, tal que $d \in \mathbb{N}^*$. Assim, quando $d = \lfloor n/k \rfloor$, obtém-se o limitante encontrado por Croitoru (1979).

Logo, temos que

$$\sum_{k=1}^{n} |I_k(G)| 2^{n-k} \leq \sum_{k=1}^{n} d^{(d+1)k-n}(d+1)^{n-dk} 2^{n-k}$$

Denominaremos $y(n,k) = d^{(d+1)k-n}(d+1)^{n-dk}2^{n-k}$. A função $y(n,k)$ tem ponto de máximo em $k = n/5$. Desta forma, o somatório pode ser dividido de maneira que em uma das partes, a função é crescente e na outra, é decrescente:

$$y(n,k) = \sum_{k=1}^{\lfloor n/5 \rfloor} y(n,k) + \sum_{k=\lfloor n/5 \rfloor+1}^{n} y(n,k)$$

Byskov (2004) provou que o valor de $d^{(d+1)k-n}(d+1)^{n-dk}$ é justo quando $n/(d+1) \leq k \leq n/d$. Sabemos que no primeiro somatório k vale no máximo $n/5$. Logo,

$$n/(d+1) \leq k \leq n/5$$
$$n/(d+1) \leq n/5$$

Fazendo $d = 5$, temos que $n/6 < n/5$.

Da mesma forma, no segundo somatório, sabemos que k vale no mínimo $\lfloor n/5 \rfloor + 1$. Logo,

$$n/5 < k \leq n/d$$
$$n/5 < n/d$$

Fazendo $d = 4$, temos que $n/5 < n/4$.

Assim,

$$\sum_{k=1}^{\lfloor n/5 \rfloor} y(n,k) = \sum_{k=1}^{\lfloor n/5 \rfloor} 5^{6k-n}6^{n-5k}2^{n-k}$$
$$\sum_{k=\lfloor n/5 \rfloor+1}^{n} y(n,k) = \sum_{k=\lfloor n/5 \rfloor+1}^{n} 4^{5k-n}5^{n-4k}2^{n-k}$$

Portanto,

$$\sum_{k=1}^{n} y(n,k) = \sum_{k=1}^{\lfloor n/5 \rfloor} 5^{6k-n}6^{n-5k}2^{n-k} + \sum_{k=\lfloor n/5 \rfloor+1}^{n} 4^{5k-n}5^{n-4k}2^{n-k}$$

O maior valor do primeiro somatório encontra-se no último termo, enquanto no segundo somatório este valor está no primeiro termo. Isto é, para o primeiro e segundo somatórios, os termos de maior valor são alcançados quando $k = \lfloor n/5 \rfloor$ e $k = \lfloor n/5 \rfloor + 1$, respectivamente.

$$\sum_{k=1}^{\lfloor n/5 \rfloor} 5^{6k-n}6^{n-5k}2^{n-k} \leq 5^{6(n/5)-n}6^{n-5(n/5)}2^{n-(n/5)} =$$
$$= 5^{n/5}2^{4n/5} = (5.2^4)^{n/5} = 80^{n/5} \leq 2.4023^n$$

$$\sum_{k=\lfloor n/5 \rfloor+1}^{n} 4^{5k-n}5^{n-4k}2^{n-k} \leq 4^{5(n/5)-n}5^{n-4(n/5)}2^{n-(n/5)} =$$
$$= 5^{n/5}2^{4n/5} = 80^{n/5} \leq 2.4023^n$$

Portanto, nesta parte o algoritmo tem tempo $O(2.4023^n)$.

Na última parte do algoritmo, isto é, o procedimento semelhante ao de Eppstein (2001), com a diferença de que são considerados os conjuntos independentes maximais de tamanho máximo $|S|/4$ ao invés de $|S|/3$, o tempo é $O(2.3814^n)$, pois

$$\sum_{S\subseteq V} 4^{\frac{5|S|}{4}-(n-|S|)}5^{(n-|S|)-4\frac{|S|}{4}} = \sum_{S\subseteq V} 4^{\frac{9|S|}{4}-n}5^{n-2|S|} \leq$$

$$\leq \sum_{i=0}^{n}\binom{n}{i}4^{\frac{9i}{4}-n}5^{n-2i} = \sum_{i=0}^{n}\binom{n}{i}\frac{4^{\frac{9i}{4}}}{4^n}\frac{5^n}{5^{2i}} =$$

$$= \left(\frac{5}{4}\right)^n \sum_{i=0}^{n}\binom{n}{i}\frac{4^{\frac{9i}{4}}}{5^{2i}} = \left(\frac{5}{4}\right)^n \sum_{i=0}^{n}\binom{n}{i}\left(\frac{4^{\frac{9}{4}}}{5^2}\right)^i =$$

$$= \left(\frac{5}{4}\right)^n \left(1+\frac{4^{\frac{9}{4}}}{5^2}\right)^n = \left(\frac{5}{4}+\frac{5.4^{\frac{9}{4}}}{4.5^2}\right)^n =$$

$$\left(\frac{5}{4}+\frac{4^{5/4}}{5}\right)^n = O(2.3814^n)$$

Finalmente, como a parte mais custosa do Algoritmo 5 está na modificação feita por Byskov (2002), então o número cromático de um grafo G é computado em tempo $O(2.4023^n)$. □

3.4 O Algoritmo de Bodlaender e Kratsch

Dentre todos os algoritmos supracitados, sabe-se que quantidade exponencial de memória é necessária para computar o número cromático de todos os subconjuntos de vértices do grafo de entrada. Por isso, os autores buscaram descrever um algoritmo exato que utiliza quantidade polinomial de memória em sua estrutura.

Diferentemente de seus antecessores, Bodlaender e Kratsch (2006) utilizam um procedimento recursivo que não busca pelos conjuntos independentes maximais (Algoritmo 6). Os autores definiram o Lema 4 como base de seu algoritmo.

Lema 4. *Seja um grafo G tal que $n = |V(G)|$, e seja $0 < \alpha < 1$. Para todo $S \subseteq V(G)$, o número cromático é dado por*

$$\chi(G[S]) = \begin{cases} 1+\min\{\chi(G-S)\}, & \text{para todo } S\subseteq V(G) \text{ se } |S|\geq \alpha.n \\ & \text{e } S \text{ é conjunto independente maximal.} \\ \min\{\chi(G[S])+\chi(G-S)\}, & \text{para todo } S\subseteq V(G) \text{ tal que } (n-\alpha.n)/2 \leq |S| \leq n/2 \end{cases} \tag{3.3}$$

Demonstração. Seja $P = (P_1, P_2, ..., P_k)$ uma coloração ótima de G. Supondo-se que algum $P_i \geq \alpha.n$ para $i \in \{1, ..., k\}$, então há duas possibilidades para P_i: ou esta cor é maximal, ou está contida em outro conjunto maximal P'_i. Dessa forma, pelo Teorema 7, o número cromático do grafo é igual a $1+\min\{\chi(G-S)\}$.

Por outro lado, suponha que todo P_i tem tamanho estritamente menor do que $\alpha.n$, para $i \in \{1, ..., k\}$. Assim, existe um conjunto $S \subseteq V(G)$ onde toda classe de cor P_i está

contida em S ou em seu complemento $V(G) \setminus S$ e, portanto, o número cromático de G é igual a $\min\{\chi(G[S]) + \chi(G - S)\}$.

Considere uma ordenação das cores tal que $|P_1| \leq |P_2| \leq ... \leq |P_k|$. Existe uma partição de P onde $|P_1| + ... + |P_q| \leq n/2$, para algum $q \in \{1, ..., k\}$. Desta forma, o subgrafo induzido por $P_1, ..., P_q$ é q-colorível maximal e como $q < k$, o subgrafo induzido por $P_{q+1}, ..., P_k$ é $(k - q)$-colorível maximal (Corolário 1).

Assim, tanto S e $V(G) \setminus S$ podem ter a mesma quantidade de vértices cada um, ou seja, $n/2$ vértices, como S pode ser menor do que $V(G) \setminus S$. Como toda cor está contida ou em S ou em seu complemento e o tamanho de uma cor neste caso é no máximo $\alpha.n$, então a diferença de tamanho entre S e $V(G) \setminus S$ corresponde a uma cor. Portanto, $|S| = (n - \alpha.n)/2$ e $|V(G) \setminus S| = (n + \alpha.n)/2$. Logo, $n/2 \geq |S| \geq (n - \alpha.n)/2$. □

Algoritmo 6: $\chi(G)$ - Bodlaender e Kratsch

Entrada: Um grafo G e um valor $0 < \alpha < 1$
Saída: O número cromático de G.
Início
  $n \leftarrow |V(G)|$
  $k \leftarrow n$
  Para *todo* $S \subseteq V(G)$
    Se *S é conjunto independente maximal e* $|S| \geq \alpha.n$ **então**
      $k \leftarrow min(k, 1 + \chi(G - S))$
    Se $(n - \alpha.n)/2 \leq |S| \leq n/2$ **então**
      $k \leftarrow min(k, \chi(G[S]) + \chi(G - S))$
Devolva k

A altura máxima de um dos ramos da árvore de soluções do Algoritmo 6 é $O(\log n)$. Como é possível enumerar todos os subconjuntos de vértices de S em espaço polinomial em cada nível da árvore de soluções, e em cada nível, o tamanho de S é no máximo n, então a memória utilizada pelo Algoritmo 6 é $O(n \log n)$.

Os autores provaram que o melhor desempenho de pior caso foi obtido para $\alpha = 0.19903$, e o tempo total alcançado foi $O(5.283^n)$.

Capítulo 4

Algoritmos baseados em Branch-and-Bound e Backtracking

Os métodos *backtracking* e *Branch-and-Bound* solucionam problemas de otimização combinatória a partir da enumeração de possíveis soluções para um problema. Um algoritmo de *backtracking* (conforme pode ser verificado em Kreher e Stinson (1999)) é um método recursivo que gera uma árvore com todas as soluções possíveis da instância de entrada, de maneira que nós folha representam soluções ótimas, e nós internos representam soluções parciais. A árvore de soluções é percorrida por busca em profundidade – passo de ida (*forward*) –, e quando uma solução completa é encontrada ou há falha na referida solução parcial, retorna-se a um ponto da árvore – passo de volta (*backward*) – onde seja possível encontrar soluções alternativas a partir dele.

O método *Branch-and-Bound*, por sua vez, consiste em explorar o espaço de soluções de maneira inteligente, ou seja, evitando que todas as possíveis soluções sejam percorridas. De maneira geral, o algoritmo também gera uma árvore de busca onde cada nó é uma solução parcial da instância de entrada. Isto é, partindo da instância inicial, esta é sucessivamente particionada em instâncias menores. Esta é a etapa de ramificação (*branching*). Em cada nó, tem-se a etapa de *bounding*, onde calcula-se um *limitante inferior*, que é menor ou igual ao valor de uma solução ótima. Se este valor for maior ou igual do que o *limitante superior* do valor da solução ótima global, então o referido nó pode ser descartado. Na etapa de *branching* pode ser aplicado um algoritmo de *backtracking*. [1]

Neste capítulo, são descritos algoritmos baseados nestes métodos.

4.1 Algoritmo de Brown

Brown (1972) propôs um algoritmo que encontra uma coloração exata em um grafo por meio de *backtracking*.

Primeiramente é obtida uma ordenação $(v_1, ..., v_n)$, de maneira que $d_G(v_i) \geq d_G(v_{i+1})$ para todo $i \in \{1, ..., n\}$. Colorem-se os vértices um a um de acordo com esta ordenação. Deste modo, cada nova coloração parcial da árvore de soluções possui i vértices coloridos, para $1 \leq i < n$, e um limitante inferior q em relação ao número cromático da coloração completa. Dado o valor k como limitante superior global para uma coloração de G, tem-se que se uma coloração completa utiliza menos do que k cores ($q < k$), então o valor de k é atualizado. Por

[1] Aqui estamos tratando de problemas de minimização. Para problemas de maximização, basta inverter os limitantes.

outro lado, se em uma coloração parcial não é possível colorir o subgrafo induzido pelos vértices $(v_1, ..., v_i)$ com menos do que k cores, então não é necessário continuar a respectiva coloração. Seja U_i o conjunto de cores disponíveis que podem ser atribuídas ao vértice v_i. Isto é, U_i irá conter as cores no conjunto $\{1, 2, .., q + 1\}$ menos aquelas que são utilizadas pelos vizinhos de i em uma coloração parcial, para algum q. Seja também l_i o total de cores utilizadas por uma coloração parcial com i vértices coloridos, e a variável booleana $atualizarU$ que indica se U_i será atualizado. Dessa maneira, o Algoritmo 7 descreve um pseudocódigo para o método proposto por Brown (1972).

Como algoritmos baseados em *backtracking* e *Branch-and-Bound* podem enumerar todas as soluções possíveis da instância de um problema, o espaço de busca gerado pode se tornar exponencial no tamanho da entrada. Desta maneira, para que o tamanho da árvore de busca seja reduzido, deve-se buscar encontrar boas estratégias nos passos de ida e de volta e nas etapas de *bounding*. Se um bom limitante superior inicial para o número cromático for encontrado, então menor será o valor de k em relação a n.

O passo de ida do Algoritmo 7 consiste em colorir sequencialmente os vértices da ordenação estabelecida inicialmente, enquanto o passo de volta está sinalizado nos trechos marcados com a indicação "BACKTRACK". Isto é, o passo de ida ocorre enquanto houver algum vértice v_i com conjunto U_i diferente de vazio ou enquanto houver vértices a serem coloridos numa solução parcial e o total de cores utilizadas seja menor do que o limitante superior atual. Ambas as operações são feitas em tempo linear em relação a entrada.

Apesar de não ser possível tornar as operações de ida e volta ainda mais eficientes, é possível torná-las mais inteligentes de modo que a solução ótima seja encontrada mais rapidamente. Brown (1972) propõe modificar o passo de ida diminuindo o tamanho do conjunto U_i ao aumentar o número de cores que não podem ser atribuídas ao vértice v_i numa coloração parcial. Isto é, nesta modificação, denominada *look-ahead*, o conjunto U_i será obtido como antes, porém toda cor $c \in \{1, ..., q\}$ que for a única opção de atribuição a um vértice v_j vizinho de v_i, tal que $j > i$, também será removida do conjunto U_i. Experimentos realizados pelo autor sobre grafos aleatórios indicaram que apesar do tempo de execução aumentar neste passo, o número de *backtracks* foi reduzido. Os resultados obtidos para grafos de densidades distintas apontaram que o algoritmo com a modificação foi mais eficiente para grafos mais densos (0,7 a 0,8), enquanto o algoritmo sem a modificação foi mais eficiente para grafos de densidades de valor 0,2 a 0,3 e 0,45 a 0,55 e número de vértices menor ou igual a 30.

Em relação à ordenação de vértices que é definida inicialmente no Algoritmo 7, esta não é modificada uma vez que é estabelecida. Desta forma, Brélaz (1979) propõe utilizar a heurística DSATUR para ordenar os vértices dinamicamente, como descrito na Seção 4.2.

4.2 Algoritmo DSATUR

O algoritmo DSATUR foi inicialmente proposto por Brélaz (1979) como uma heurística gulosa para coloração de vértices, e é descrito no Algoritmo 9. O autor introduziu o conceito de *grau de saturação* de um vértice como a quantidade de cores distintas atribuídas aos vizinhos coloridos de um vértice qualquer de um grafo G.

Teorema 13. *(Goldbarg e Goldbarg (2012)) O algoritmo DSATUR tem tempo* $O(n^2)$.

Algoritmo 7: BROWN(G)

Entrada: Um grafo G
Saída: Uma coloração ótima de G.
Início
$n \leftarrow |V|$.
Obtenha uma ordenação $(v_1, v_2, ..., v_n)$ para $V(G)$, tal que $d_G(v_i) \geq d_G(v_{i+1})$ para todo $i \in \{1, ..., n\}$.
Atribua a cor de índice 1 a v_1.
$i \leftarrow 2$
$k \leftarrow n$
$q \leftarrow 1$
$U_1 \leftarrow \emptyset$
$l_1 \leftarrow 1$
$atualizarU \leftarrow VERDADEIRO$
Enquanto $i > 1$
//Soluções são geradas enquanto o nó raiz da árvore não é alcançado.
Se $atualizarU = VERDADEIRO$ **então**
Determine o conjunto U_i, tal que U_i contém as cores do conjunto $\{1, 2, .., q + 1\}$ menos aquelas que são utilizadas pelos vizinhos do vértice v_i.
Se $U_i = \emptyset$ **então**
BACKTRACK($i - 1$,i,l_i,q)
$atualizarU \leftarrow FALSO$
Senão
Escolha $j \in U_i$, tal que j é mínimo, e atribua a cor j ao vértice v_i.
Remova a cor j do conjunto U_i.
Se *a cor j é menor do que o limitante superior k* **então**
Se *a cor j é maior do que o limitante inferior q* **então**
$q \leftarrow q + 1$
Se $i = n$ **então**
Guarde a solução atual e faça $k \leftarrow q$.
Encontre o menor índice j para o qual a cor do vértice v_j é igual a k.
BACKTRACK($j - 1$,i,$k - 1$,q)
$atualizarU \leftarrow FALSO$
Senão
$l_i \leftarrow q$
$i \leftarrow i + 1$ //um novo vértice é selecionado para ser colorido
$atualizarU \leftarrow VERDADEIRO$
Senão
BACKTRACK($i - 1$,i,l_i,q)
$atualizarU \leftarrow FALSO$
Devolva *Uma função $c : V(G) \rightarrow \{1, ..., k\}$*

Algoritmo 8: BACKTRACK(INDPASSO,I,LIMINF,Q)

Entrada: As variáveis i e q e os valores inteiros $indPasso$ e $limInf$
Início
$i \leftarrow indPasso$
$q \leftarrow limInf$

Algoritmo 9: DSATUR

Entrada: Um grafo G
Saída: Uma coloração ζ de G.
Início
$n = |V(G)|$
$\zeta \leftarrow \emptyset$
I_i é um conjunto que representa uma cor de G, para $1 \leq i \leq n$.
Obtenha uma ordenação $(v_1, v_2, ..., v_n)$ para $V(G)$, de maneira que $d_G(v_i) \geq d_G(v_{i+1})$, para $i \in \{1, ..., n\}$.
Atribua a cor de índice 1 a v_1.
Para *cada* $i \leftarrow 1$ *até* n
$I_i \leftarrow \emptyset$
Enquanto *enquanto houverem vértices a serem coloridos*
Escolha o vértice v ainda não colorido que possui maior grau de saturação. Caso haja mais de um vértice com o maior grau, então escolha v tal que $d_G(v)$ é máximo. Se houver empate entre os vértices mesmo neste critério, então escolha aleatoriamente um dentre estes vértices empatados.

$j \leftarrow 1$
Enquanto *v não tem cor atribuída a si e* $k \leq n$
Se $N_G(v) \cap I_j = \emptyset$ **então**
Acrescente v a I_j.
Acrescente I_j a ζ.
Senão
$j \leftarrow j + 1$
Devolva ζ

Demonstração. Se o grafo estiver implementado como uma lista de adjacências, os graus de cada vértice podem ser determinados em tempo $O(n + m)$, onde $m = |E(G)|$, e a ordenação de acordo com os graus pode ser feita em tempo $O(n \lg n)$.

Cada vértice v pode estar associado a um vetor binário de tamanho n, onde cada posição do vetor está associada a uma cor. Se uma posição estiver valorada com 1, então isto significa que a cor associada a esta posição é utilizada por um dos vizinhos de v. Assim, para encontrar a menor cor a ser atribuída a v, basta buscar pela primeira posição do vetor valorada com 0, o que tem tempo $O(n)$. Toda vez que um vértice é colorido, os vetores de seus vizinhos são atualizados, bem como é atualizado o valor de seu grau de saturação. Esta operação também tem tempo $O(n)$.

Portanto, como as etapas de verificação e atualização descritas acima são realizadas em todo vértice $v \in V(G)$, então o algoritmo tem tempo total $O(n^2)$. □

O trabalho de Turner (1988) apresenta uma versão do DSATUR que executa em tempo $O(m \log n)$.

O Algoritmo DSATUR como heurística gulosa apresenta-se como um procedimento exato para grafos bipartidos, além de ser uma heurística que encontra uma clique maximal em um grafo, e portanto, determina um limitante inferior para $\chi(G)$.

Teorema 14. *O algoritmo DSATUR é exato para grafos bipartidos.*

Demonstração. Seja G um grafo bipartido conexo com pelo menos três vértices, onde $V(G) = X \cup Y$. Iremos provar que o grau de saturação de todo vértice de G é igual a 1.

Seja um vértice $v \in X$ que tenha grau de saturação igual a 2 e sejam v_1 e v_2 os dois vizinhos de v que possuem cores distintas. Suponhamos que pelo menos v_1 ou v_2 possui um vizinho u em X, pois G é conexo. Como G é bipartido, então todo caminho de v a u contendo v_1 e v_2 tem comprimento par e v_1 e v_2 não podem ter aresta entre si, pois se assim fosse, então G teria um caminho de u a v de comprimento ímpar, contrariando o fato de que G é bipartido. Assim, ao contrário da suposição inicial, como v_1 e v_2 não são vizinhos então estes possuem a mesma cor, e portanto, o grau de saturação de v é igual a 1. □

Seja $(v_1, v_2, ..., v_i)$, para $i \leq n$, uma ordenação não-crescente de vértices de $V(G)$ segundo seus graus de saturação, onde v_1 corresponde ao vértice que contém a maior quantidade de vizinhos em G e cada vértice desta ordenação possui uma cor atribuída a si. Seja $v_{i+1} \in G - \{v_1, v_2, ..., v_i\}$ o vértice escolhido com o maior grau de saturação na iteração $i + 1$ do algoritmo. Se este valor for maior do que o grau de saturação do vértice v_i, então isto significa que cada vizinho de v_{i+1} utiliza uma cor distinta, e portanto, tem-se uma clique formada por v_{i+1} e seus vizinhos em $G[\{v_1, v_2, ..., v_i\}]$. Entretanto, se o grau de saturação de v_{i+1} for menor ou igual que o de v_i, então v_{i+1} faz parte de outra aresta ou clique maximal em relação à encontrada até então, visto que ele pode ser colorido com a mesma cor já utilizada por outro vértice do subgrafo $G[\{v_1, v_2, ..., v_i\}]$.

Uma modificação do Algoritmo 9 para encontrar uma clique maximal pode ser feita criando-se uma lista que armazena o primeiro vértice da ordenação estabelecida inicialmente, e que acrescenta vértices sequencialmente enquanto o grau de saturação do vértice encontrado na iteração atual for maior do que o encontrado na iteração anterior. Ao final, os vértices contidos nesta lista correspondem a uma clique maximal.

Brélaz (1979) propôs adaptar o algoritmo DSATUR para torná-lo exato tendo como base o trabalho de Brown (1972) descrito na seção anterior. Isto é, a enumeração de soluções é feita como no Algoritmo 7, contudo a ordem em que os vértices são percorridos é alterada dinamicamente. Ou seja, enquanto no Algoritmo 7 os vértices são escolhidos sequencialmente de acordo com a ordem estabelecida inicialmente, na adaptação de Brélaz (1979) esta operação é substituída pelo índice i do vértice que contém o maior grau de saturação. No caso de dois vértices terem o mesmo grau de saturação, então o critério de desempate é determinado pelo vértice que possuir maior quantidade de vizinhos. Se novamente houver empate, então é escolhido aleatoriamente um dentre os vértices que empataram entre si.
Um exemplo da árvore de busca do algoritmo é apresentado na Figura 5, para o grafo da Figura 4. Neste exemplo, cada cor que está riscada no conjunto U_i indica uma cor que não pode ser utilizada pelo vértice v_i na iteração atual. Cada nó da árvore e seus respectivos antepassados representam uma coloração parcial. Por exemplo: os nós onde atribuem-se as cores 1, 2 e 2 aos vértices A, F e B, respectivamente, representam uma 2-coloração parcial.

Neste exemplo, uma coloração inicial com 4 cores foi encontrada nos passos (1) a (7). Assim, o limitante superior é atualizado para valor igual a 4, e nos próximos passos, será

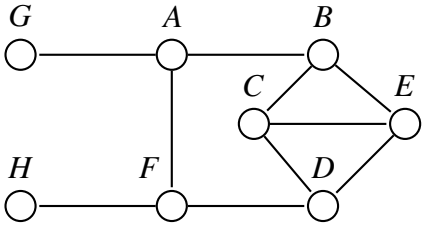


Figura 4: Grafo de exemplo que será colorido pelo Algoritmo 7

procurada uma solução melhor do que atual, ou seja, uma solução que contenha 3 cores. Como o vértice E é aquele que obtém a maior cor no passo (6), então a etapa de *backtrack* começa a partir do passo anterior, isto é, no passo (5). Visto que não há outra cor possível para o vértice D neste passo, então tenta-se atribuir uma nova cor para o vértice C. A cor 3 está disponível, então esta cor é atribuída a C e o grafo é recolorido a partir deste ponto nos passos (9) a (11). Como não é possível colorir o vértice E com menos do que 4 cores, então a coloração parcial é encerrada aí e o *backtrack* ocorre no passo (10).
Como percebe-se nos passos (13) a (18), o valor do número cromático da coloração foi melhorado quando B foi recolorido com a cor 3. Como não é possível melhorar esta solução, então o número cromático do grafo é 3.
A execução do Algoritmo 7 adaptado com uso do DSATUR produziu as soluções representadas na Figura 6 para o grafo da Figura 4. As cores de índices 1, 2, 3 e 4 estão representadas pelos padrões “linhas horizontais”, “linhas verticais”, “pontilhado” e “quadriculado”, respectivamente. A melhor solução encontrada neste exemplo é a solução 2.

Brélaz (1979) observou que ao elegerem-se os vértices por ordem de saturação para serem coloridos, isto significa que a prioridade é dada àqueles com menos opções possíveis de cores e, com isso, é possível obter uma coloração com limitante superior mais próximo do número cromático com menos iterações.

Testes deste algoritmo foram feitos em grafos aleatórios de até 100 vértices e densidades variadas (0,3 a 0,7). Os resultados obtidos por Brélaz (1979) foram comparados com o algoritmo *look-ahead* de Brown (1972), concluindo que este último só é útil para grafos pequenos (menos de 60 vértices), enquanto o procedimento de Brélaz (1979) (incluindo modificação *look-ahead* como Brown (1972)) mostrou-se em geral mais eficiente, bem como reduziu o número de *backtracks* gerado.

Trabalhos baseados em Brélaz (1979)

Modificações do trabalho de Brélaz (1979) foram feitas no critério de desempate quando o maior grau de saturação pertence a mais de um vértice. Dentre eles destacam-se Sewell (1996) e San Segundo (2012).

- Sewell (1996) propôs escolher o vértice que compartilha mais cores disponíveis com seus vizinhos no subgrafo não colorido, pois ao ter uma cor atribuída a si, as opções para seus vizinhos tornam-se ainda mais restritas.

- San Segundo (2012) adaptou o critério de desempate de Sewell (1996). O autor propôs escolher o vértice que compartilha mais cores disponíveis com os outros vértices que estão empatados.

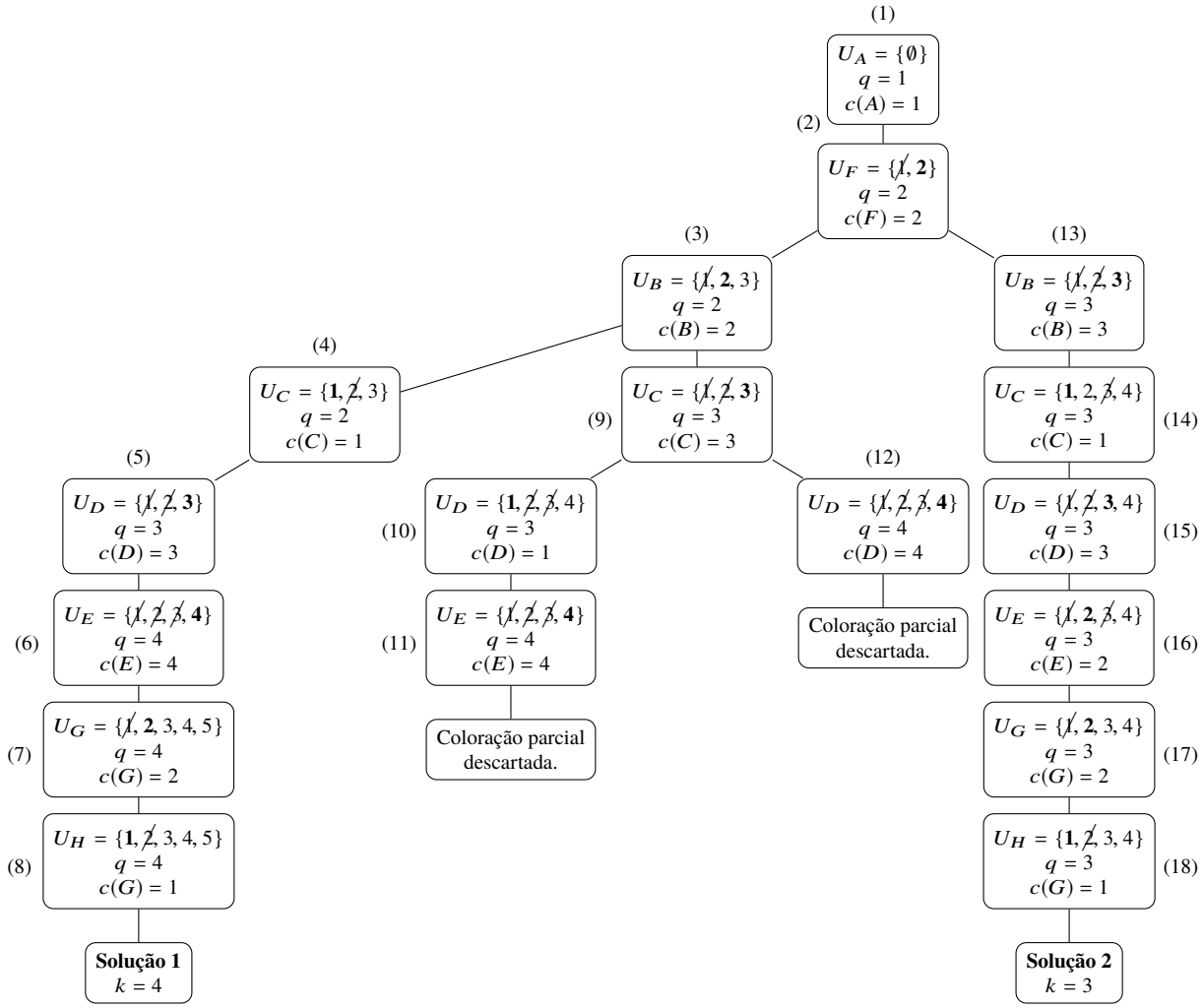


Figura 5: Árvore de busca de exemplo de execução do Algoritmo 7 adaptado por Brélaz (1979)

San Segundo (2012) executou seu algoritmo em grafos aleatórios de até 80 vértices e densidades variadas e instâncias contidas no *benchmark* DIMACS[2], que inclui grafos baseados em problemas de busca (problema das *n* rainhas, por exemplo), grafos geométricos aleatórios e grafos modelados a partir de problemas do mundo real[3]. Os resultados obtidos foram comparados com execuções dos algoritmos de seus predecessores nas mesmas instâncias. O algoritmo de San Segundo (2012) mostrou-se mais eficiente em grafos mais densos (valores de densidades acima de 0,7), enquanto em grafos menos densos os três algoritmos obtiveram resultados parecidos. Para as instâncias do DIMACS, o algoritmo de San Segundo (2012) é superior em relação aos demais (exceto em apenas cinco delas).

A adaptação mais recente para o algoritmo de Brélaz (1979) consiste no trabalho de Furini et al. (2017). Os autores observam que nos outros trabalhos, uma clique maximal é encontrada e colorida antes do processo de coloração iniciar, e seu tamanho determina um bom limitante inferior inicial. Além disso, o limitante superior determinado na primeira solução encontrada pelo algoritmo é recalculado de maneira iterativa, ou seja, é decrementado de uma a uma unidade até que encontre-se o valor para o qual a coloração é ótima. Com isso, ao invés de modificar as estratégias de critério de desempate como seus antecessores fizeram, Furini et al.

[2]Disponível em http://dimacs.rutgers.edu/Challenges/
[3]Disponível em http://mat.gsia.cmu.edu/COLOR/instances.html

(a) Solução 1 (b) Solução 2

Figura 6: Colorações obtidas por meio de execução do Algoritmo 7 no grafo da Figura 5

(2017) propõem atualizar o valor do limitante superior global dinamicamente. Furini et al. (2017) concluíram que seu algoritmo apresenta resultados melhores para grafos entre 75 e 80 vértices em relação ao trabalho de San Segundo (2012), porém a diferença de custo computacional é pequena.

4.3 Árvores de Zykov

Zykov (1962) propôs uma enumeração de soluções que altera a estrutura do grafo a cada iteração. Antes de entender a estrutura do método proposto pelo autor, é necessário definir as modificações sobre os vértices de um grafo G, expostas na Definição 2. Dados dois vértices não adjacentes $x, y \in V(G)$, um dos novos grafos obtidos a partir de G terá x e y contraídos a um único vértice, enquanto o outro terá uma aresta entre x e y adicionada a si.

Definição 2. *Dados dois vértices não adjacentes $x, y \in V(G)$, uma* contração *em G produz um grafo G'_{xy} que é dado por*

$$V(G'_{xy}) = V(G) \setminus \{x, y\} \cup \{z\}$$
$$E(G'_{xy}) = \begin{Bmatrix} \{u, v\} \in E(G) \text{ tal que } x \notin \{u, v\} \text{ e } y \notin \{u, v\} \\ \cup \\ \{u, z\} \text{ tal que } \{u, x\} \in E(G) \text{ ou } \{u, y\} \in E(G) \end{Bmatrix}$$

Dados dois vértices não adjacentes $x, y \in V(G)$, uma adição *em G produz um grafo G''_{xy} que é dado por*

$$V(G''_{xy}) = V(G)$$
$$E(G''_{xy}) = E(G) \cup \{\{x, y\}\}$$

Em outras palavras, uma coloração de G que atribui a mesma cor a x e y corresponde a uma coloração de G'_{xy}, enquanto uma coloração de G que atribui cores distintas a x e y corresponde a uma coloração de G''_{xy}.

Teorema 15. *O número cromático do grafo G é dado pela recorrência*

$$\chi(G) = \min\{\chi(G'_{xy}), \chi(G''_{xy})\} \text{ tal que } x, y \in V(G) \text{ e } \{x, y\} \notin E(G)$$

Apesar da estrutura de G ser alterada cada vez que uma operação de contração ou de adição é executada, os vértices envolvidos nestas operações são sempre aqueles que não são

vizinhos em G. Portanto, todas as outras relações de vizinhança do grafo original são mantidas em cada grafo modificado pois, ou uma nova aresta é adicionada e as outras são preservadas, ou as arestas incidentes em x e y no grafo G estão incidentes no vértice z no grafo modificado.

Como uma clique não contém pares de vértices que não são vizinhos, então contrações de vértices e adições de aresta são feitas enquanto um dos grafos G'_{xy} ou G''_{xy} não forem cliques. Dessa maneira, sucessivas aplicações da recorrência contida no Teorema 15 originam uma árvore binária denominada *árvore de Zykov*, onde nós folha são cliques, e o número cromático é dado pelo tamanho da menor clique dessa árvore. Um exemplo de uma árvore de Zykov está ilustrado na Figura 7, onde o grafo G dado como entrada está representando na raiz da árvore.

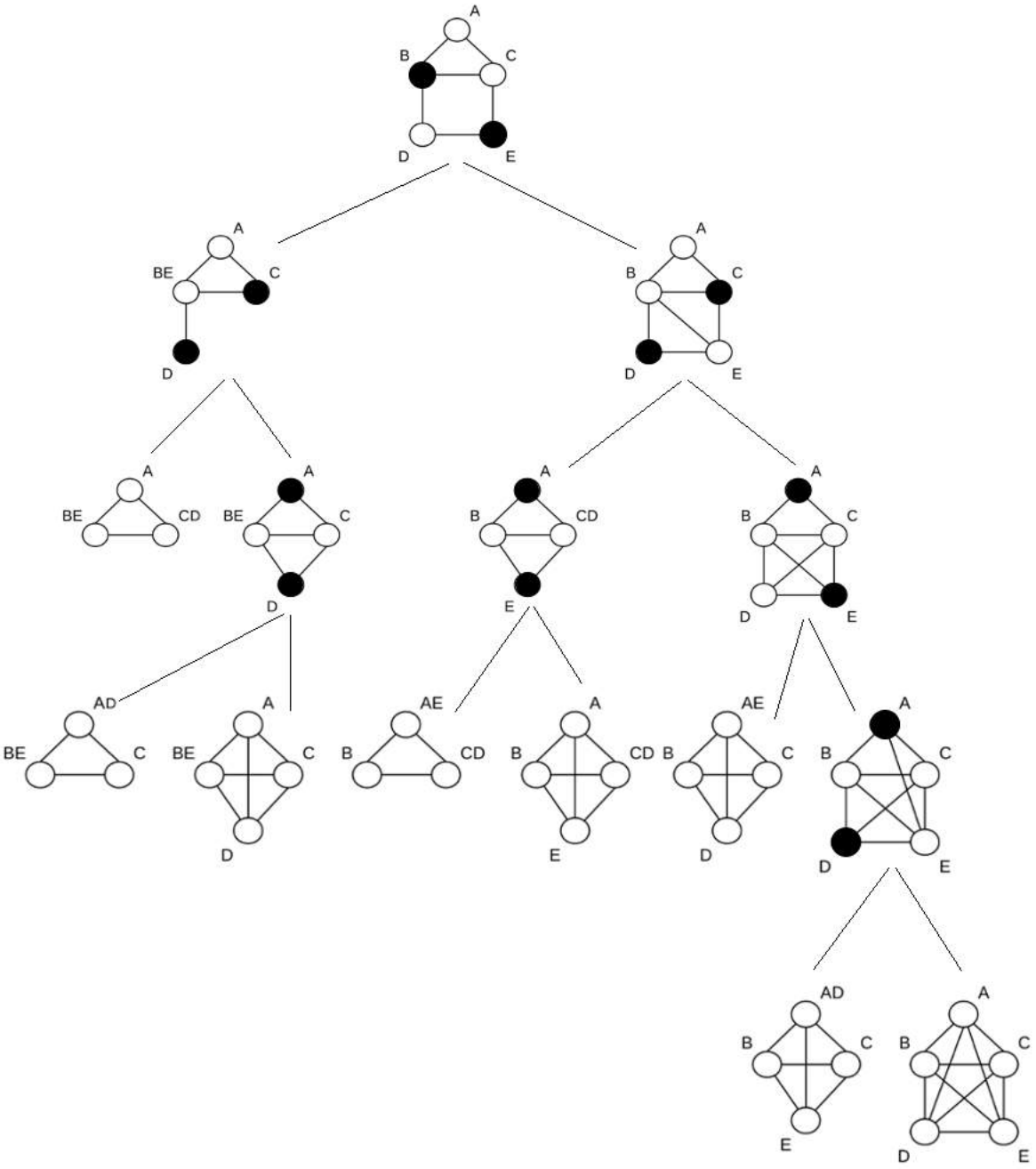


Figura 7: Árvore de Zykov

Neste exemplo, cada par de vértices destacado em preto em cada etapa da recorrência representa um par de vértices não adjacentes. Cada uma das oito cliques obtidas representa uma coloração possível de G, e a coloração ótima – representada na Figura 8 – encontra-se na clique de tamanho 3 constituída pelos vértices $\{A, BE, CD\}$. O número cromático de G, portanto é, 3.

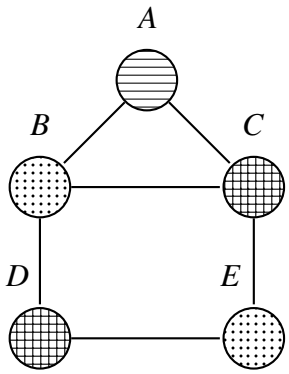


Figura 8: Coloração ótima de G

Como se pode perceber no exemplo, árvores de Zykov tem exatamente um ramo onde são feitas apenas operações de adição de aresta, onde o número da coloração resultante é igual a n (ramo mais à direita). Por outro lado, há também exatamente um ramo onde são feitas apenas operações de contração de aresta, onde o número da coloração resultante é um limitante superior para o número cromático (ramo mais à esquerda).

Em algoritmos baseados em árvores de Zykov, o ramo mais a esquerda torna-se o limitante superior inicial q para a coloração. Este limitante é atualizado à medida que uma solução com menos cores é encontrada. Uma instância só é ramificada se não contém subgrafos que são cliques de tamanho q, pois como sabe-se, o tamanho de qualquer clique contida em um grafo G é limitante inferior para $\chi(G)$.

Os Algoritmos 10 e 11 contém a descrição de um algoritmo *Branch-and-Bound* para o problema de encontrar o número cromático de G por meio do algoritmo de Zykov (1962), onde q é a variável global que representa o limitante superior, e l é o tamanho do grafo que é passado como parâmetro (Araújo Neto e Gomes, 2014).

Algoritmo 10: Cor(G)

Entrada: Um grafo G
Saída: O número cromático de G.
Início
  $n \leftarrow |V(G)|$
  Se *G é grafo completo tal que $|V(G)| = n$* **então**
    $q \leftarrow \min\{n, q\}$
  Senão se *G não contém clique de tamanho q* **então**
    Escolha $x, y \in V(G)$ tais que $\{x, y\} \notin E(G)$.
    Cor(G'_{xy}) //grafo obtido por meio de contração de aresta
    Cor(G''_{xy}) //grafo obtido por meio de adição de aresta
Devolva q

Algoritmo 11: ZYKOV(G)

Entrada: Um grafo G
Saída: O número cromático de G.
Início
$n \leftarrow |V(G)|$
$\chi(G) \leftarrow Cor(G)$
Devolva $\chi(G)$

A Figura 9 contém a árvore de busca do Algoritmo 11 para o grafo da Figura 7. Como percebe-se no exemplo, as subárvores dos Passos (4) e (5) não foram exploradas, pois estas admitem cliques de tamanho maior ou igual a 3, que foi o limitante superior encontrado no Passo (3) da Figura 9.

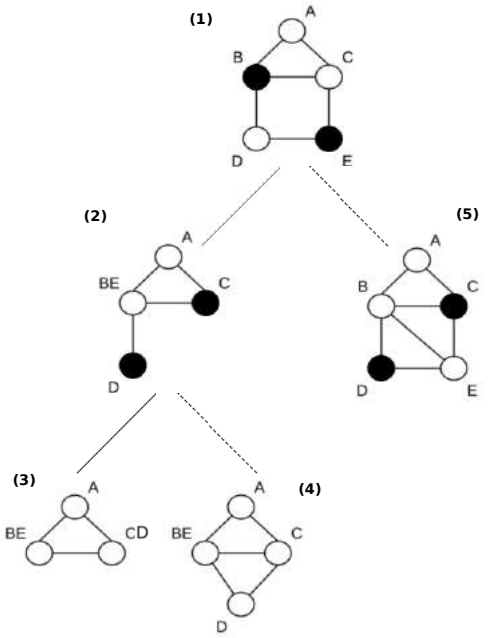


Figura 9: Árvore de Zykov de tamanho reduzido

Marin (2005) explica como estimar o espaço e tempo de algoritmos baseados em árvores de Zykov, considerando que a chamada recursiva da operação de adição é feita antes da operação de contração no Algoritmo 10.

Neste caso, a altura de uma árvore de Zykov é determinada pela quantidade de arestas que faltam para que o grafo G seja completo, ou seja, a quantidade de arestas $\overline{m}$ do grafo complementar de G. Como toda árvore de Zykov é estritamente binária, então o número máximo de nós é dado por

$$\sum_{i=0}^{\bar{m}} 2^i = \frac{2^{\bar{m}+1}-1}{2-1} = 2^{\bar{m}+1} - 1 = 2^{\binom{n}{2}-m+1} - 1$$

O grafo complementar de G será completo no pior caso, ou seja, $\bar{m} = \frac{n(n-1)}{2}$. Assim, tem-se que a complexidade de tempo é $O(2^{n^2})$.

Para estimar o espaço, tem-se que se a altura máxima da árvore é $O(n^2)$ e em cada nível o grafo todo é armazenado, então o espaço total é $O(n^2(n+m))$.

Dentre os trabalhos presentes na literatura que estudam árvores de Zykov destacam-se Corneil e Graham (1973) e McDiarmid (1979).

A ideia de Corneil e Graham (1973) consiste em primeiramente determinar um limitante superior para o número cromático, denotado q, e então determinar um subgrafo de tamanho q onde os vértices tem alta densidade de arestas entre si (que os autores denominam-no *cluster*), para então eleger dois vértices não adjacentes neste *cluster*. O par de vértices não adjacentes é escolhido neste *cluster*, pois isto facilita a busca por cliques na etapa de *bounding* do algoritmo. O algoritmo de Corneil e Graham (1973) utiliza quantidade de memória $O(n^3)$.

McDiarmid (1979) focou-se em fazer análise de pior caso do algoritmo de Zykov (1962), e confrontou os resultados de Corneil e Graham (1973), afirmando que algoritmos baseados em árvores de Zykov não são mais eficientes do que algoritmos que geram todos os conjuntos independentes maximais do grafo de entrada. Os autores concluíram que assintoticamente, para quase todo grafo, toda árvore de Zykov tem tamanho $O(e^{cn\sqrt{\log n}})$, onde c é uma constante maior que zero.

Atualmente, a recorrência de Zykov (1962) é utilizada na etapa de *branching* de algoritmos *Branch-and-Price*, como descrito no Capítulo 5.

4.3.1 Polinômio cromático

A recorrência proposta por Zykov (1962) permite obter o *polinômio cromático* de um grafo G. Isto é, para $n = |V(G)|$ e algum $k \leq n$, o polinômio cromático de G, denotado $P(G, k)$, é o número de k-colorações próprias G.

O polinômio cromático de um grafo completo K_n é igual a $k(k-1)(k-2)...(k-n-1)$. Como cada nó folha de uma árvore de Zykov é um grafo completo que representa uma coloração própria possível de G, então $P(G, k)$ é determinado pela soma dos polinômios cromáticos de cada um destes grafos.

Seja o grafo C_4 da Figura 10. O polinômio cromático deste grafo é igual a $k^4 - 3k^3 + 4k^2 - 2k$, pois

$$P(C_4, k) = P(K_4, k) + 2P(K_3, k) + P(K_2, k) =$$

$$= k(k-1)(k-2)(k-3) + 2k(k-1)(k-2) + k(k-1) =$$
$$= (k^4 - 6k^3 + 11k^2 - 6k) + (2k^3 - 6k^2 + 4k) + (k^2 - k) =$$
$$= k^4 - 4k^3 + 6k^2 - 3k$$

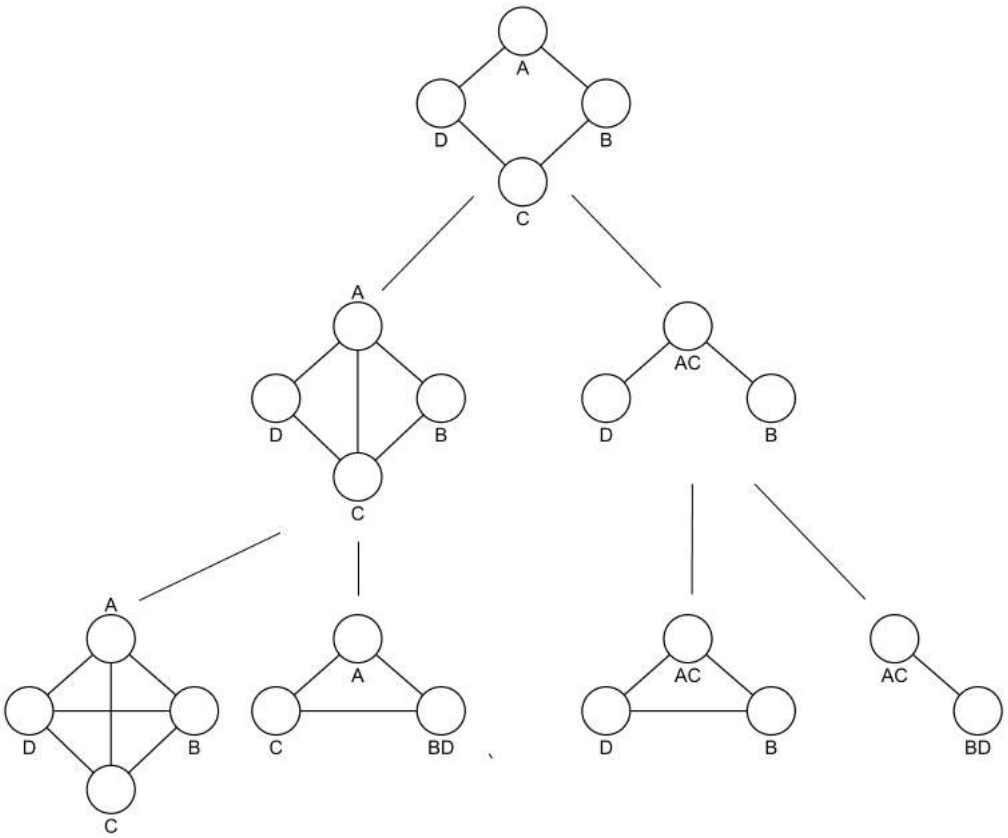


Figura 10: Grafo C_4

Mais detalhes podem ser consultados em Dong et al. (2005).

Capítulo 5

Algoritmos baseados em Programação Linear

Neste capítulo, apresentamos conceitos de Programação Linear e formulações do problema de coloração de vértices por meio de Programação Linear Inteira. Além disso, são citados alguns dos principais trabalhos da literatura que solucionam o problema desta maneira.

Dentre estes algoritmos, o método que apresenta o melhor desempenho para resolução dos mesmos é o método *Branch-and-Price*, que será apresentado na Seção 5.6. Para isso, é necessária a compreensão do algoritmo *Simplex* (Dantzig, 1963) e do método *Geração de Colunas*, pois *Branch-and-Price* consiste em uma adaptação do *Branch-and-Bound* associado a *Geração de Colunas*.

5.1 Formulação

Quando temos um problema de otimização onde é dada uma função linear sujeita a um conjunto de restrições, também lineares, então temos um *programa linear*. Quando o objetivo é encontrar o menor valor possível para a função, então temos um programa linear de *minimização*.

Sejam $n, m \in \mathbb{N}^*$, a matriz $A \in \mathbb{Q}^{n \times m}$ e $c \in \mathbb{Q}^n$, $x \in \mathbb{Q}^n_+$ e $b \in \mathbb{Q}^m$ vetores representados como matrizes coluna, de modo que c^T denota o vetor c transposto. Definimos as formulações canônicas e padrão de um programa linear como abaixo.

Definição 3. *Um programa linear na forma canônica, definido por A, b e c, é dado por*

$$\begin{aligned} &\textit{Minimizar } z = c^T x \\ &\textit{sujeito a } Ax \geq b \\ &x \geq 0 \end{aligned}$$

Definição 4. *Um programa linear na forma padrão, definido por A, b e c, é dado por*

$$\begin{aligned} &\textit{Minimizar } z = c^T x \\ &\textit{sujeito a } Ax = b \\ &x \geq 0 \end{aligned}$$

Um programa linear na formulação canônica pode ser convertido na formulação padrão com o acréscimo de *variáveis de folga* nas desigualdades do programa linear. Neste caso, $Ax \geq b$

é substituído por $Ax - Ix_S = b$, onde x_S é o conjunto de variáveis de folga e I é a matriz identidade.
Seja $b_i \in b$ e seja a seguinte restrição de maior ou igual:

$$x_1 + ... + x_n \geq b_i$$

A restrição pode ser transformada em uma igualdade com o acréscimo de uma variável de folga x_{S_1} de coeficiente negativo, ou seja:

$$x_1 + ... + x_n - x_{S_1} = b_i$$

Da mesma forma, seja uma restrição de menor ou igual:

$$x_1 + ... + x_n \leq b_i$$

A restrição pode ser transformada em uma igualdade com o acréscimo de uma variável de folga x_{S_1} de coeficiente positivo, ou seja:

$$x_1 + ... + x_n + x_{S_1} = b_i$$

Assim, para x_S sendo o conjunto de variáveis de folga acrescidas ao programa linear dado como entrada, a função objetivo é escrita como:

$$\text{Minimizar } c^T x + 0^T x_S$$

Maiores detalhes podem ser verificados Matoušek e Gärtner (2007).

5.1.1 Programação Linear Inteira

Um *programa linear inteiro* é um programa linear que acrescenta restrições de integralidade aos valores do vetor x. Assim, equivalentemente a um programa linear, tem-se os vetores $c \in \mathbb{Q}^n$, $b \in \mathbb{Q}^m$ e agora $x \in \mathbb{Z}_+^n$, e a matriz $A \in \mathbb{Q}^{n \times m}$.

Definição 5. *A formulação canônica do programa linear definido por A, b e c é expressa como:*

$$\begin{aligned} \textit{Minimizar } z &= c^T x \\ \textit{sujeito a } Ax &\geq b \\ x &\in \mathbb{Z}_+^n \end{aligned}$$

Definição 6. *A formulação padrão do programa linear definido por A, b e c é expressa como:*

$$\begin{aligned} \textit{Minimizar } z &= c^T x \\ \textit{sujeito a } Ax &= b \\ x &\in \mathbb{Z}_+^n \end{aligned}$$

O valor da função objetivo para a solução ótima não inteira de um programa linear inteiro de minimização é um limitante inferior para o valor referente à solução ótima inteira, e a diferença entre estes valores é denominada intervalo (*gap*) de integralidade.

O problema da Programação Linear é polinomial, enquanto a Programação Linear Inteira é da classe $\mathcal{NP}$-Difícil (Papadimitriou e Steiglitz, 1998).

5.2 Dualidade

Definição 7. *Seja P o programa linear abaixo:*

$$\begin{aligned} &\textit{Minimizar } c^T x \\ &\textit{sujeito a } Ax \geq b \\ &\qquad x \geq 0 \end{aligned}$$

O programa dual de P é definido como

$$\begin{aligned} &\textit{Maximizar } b^T y \\ &\textit{sujeito a } A^T y \leq c \\ &\qquad x \geq 0 \end{aligned}$$

O programa P é denominado *primal*. Dentre as propriedades e relações entre um programa linear e seu respectivo dual, as principais são aquelas que relacionam os valores das soluções de cada um.

Definição 8. *Se existir uma valoração das variáveis do vetor x que satisfaça todas as restrições de um programa linear, esta é denominada* solução viável.

Definição 9. *Uma solução viável x é denominada* solução ótima *se x admite o menor valor possível na função objetivo.*

Teorema 16. ***(Teorema da dualidade fraca)*** *Sejam x e y soluções viáveis de um programa linear e seu dual associado, respectivamente. Então,*

$$b^T y \leq c^T x$$

Demonstração. Pelo conjunto de restrições do dual, $c \geq A^T y$. Como $A^T y = y^T A$, logo,

$$c^T x \geq y^T A\, x$$

Como $Ax \geq b$ em P, então

$$c^T x \geq y^T A\, x \geq b^T y$$

□

5.3 O Algoritmo Simplex

O Algoritmo Simplex (Dantzig, 1963) é o algoritmo mais conhecido na literatura em relação a soluções para programas lineares. Em linhas gerais, o algoritmo busca satisfazer o sistema de equações composto pelo conjunto de restrições do programa linear dado como entrada, partindo de uma solução viável inicial e tentando melhorá-la enquanto for possível a cada iteração.

Para entender os passos do Simplex, primeiramente é necessário caracterizar uma solução e seu respectivo valor. Os teoremas não demonstrados neste capítulo foram provados por Matoušek e Gärtner (2007).
Seja uma matriz $A \in \mathbb{Q}^{n \times m}$, para $n, m \in \mathbb{N}^*$, e seja o conjunto $X \subseteq \{1, ..., m\}$. A matriz A_X é a matriz formada pelas colunas de A indexadas pelos elementos de X. Da mesma maneira, seja o vetor b de tamanho m. b_X é o vetor obtido pelos elementos de b indexados por X.

Definição 10. *Sejam n e m o número de variáveis e restrições de um programa linear, respectivamente, e seja a matriz A. Uma* solução básica viável *é uma solução viável onde existe um conjunto $B \subseteq \{1, ..., m\}$ tal que a matriz A_B obtida a partir de A é não-singular.*

A matriz A_B tem dimensão $m \times m$ e é denominada *básica*. Cada coluna de A está associada a uma variável do programa linear dado como entrada. Desta forma, x_B é o conjunto de variáveis associadas as colunas de A_B. As variáveis de x_B são denominadas *básicas*.

Seja $N \subseteq \{1, ..., n\} \setminus B$. A matriz A_N tem dimensão $m \times (n - m)$ e é denominada *não-básica*. Da mesma forma, o conjunto de variáveis x_N associado as colunas de A_N é denominado conjunto de variáveis *não-básicas*.

Em uma solução básica viável, $x_i = 0$ para todo $i \in N$.

Teorema 17. *Se um programa linear admite solução ótima, então existe uma solução básica viável que é ótima.*

Dessa forma, uma vez caracterizada uma solução básica viável, a matriz de restrições pode ser reescrita com base na partição de A em A_B e A_N. Denotamos como $A = [A_B|A_N]$. Assim a equação $Ax = b$ pode ser reescrita como

$$A_B x_B + A_N x_N = b$$

A ideia principal do Algoritmo Simplex consiste em partir de uma solução básica viável x_B para o programa fornecido como entrada, e alterá-la (obtendo assim, uma nova solução) enquanto for possível, de maneira que o valor da função objetivo obtido por meio da referida solução diminua. A seguir, tem-se uma breve explicação de como essa alteração ocorre.

Visto que o valor de uma solução básica viável é determinado pelo valor das variáveis básicas, então x_B será isolada na equação $A_B x_B + A_N x_N = b$. Para isso, é necessário multiplicar esta equação por A_B^{-1}, pois $A_B^{-1} A_B = I$.

$$A_B^{-1} A_B x_B + A_B^{-1} A_N x_N = A_B^{-1} b$$

$$x_B = A_B^{-1} b - A_B^{-1} A_N x_N$$

O valor de toda variável não-básica é igual a zero em uma solução básica viável. Contudo, ao incrementar-se em uma unidade o valor de uma das variáveis não-básicas desta solução, isto irá afetar o valor das variáveis básicas e da função objetivo, que terão seus valores decrementados.

Para entender esta alteração, a função objetivo será reescrita em função de x_B. Seja $c = [c_B|c_N]$ o vetor de custos, isto é, os coeficientes associados as variáveis na função objetivo. c_B representa os custos das variáveis básicas e c_N, das não-básicas.

$$\begin{aligned} z &= c_B^T x_B + c_N^T x_N \\ &= c_B^T(A_B^{-1}b - A_B^{-1}A_N x_N) + c_N^T x_N \\ &= c_B^T A_B^{-1} b - c_B^T A_B^{-1} A_N x_N + c_N^T x_N \\ &= c_B^T A_B^{-1} b + x_N(c_N^T - c_B^T A_B^{-1} A_N) \end{aligned}$$

Denominando $\lambda^T = c_B^T A_B^{-1}$ e $\overline{c} = c_N^T - \lambda^T A_N$, de maneira que, $z = \lambda^T b + x_N \overline{c}$, temos que

- $\lambda^T b$ é o valor da solução básica;
- $\overline{c}$ é chamado de vetor de *custos reduzidos*. Se o menor custo reduzido for menor do que zero, então a solução básica atual não é ótima, pois existe alguma variável não-básica x_i, para $i \in N$, onde seu respectivo valor $\overline{c}_i$ reduz o valor da solução básica atual $\lambda^T b$ [1], e
- λ^T é denominado *vetor de multiplicadores duais*. Se $\lambda^T b$ é o valor da solução ótima do programa linear P dado como entrada, isto é, não há custo reduzido em $\overline{c}$ menor do que zero, então esta também é a solução ótima do seu respectivo dual, pois

$$c^T x = c_B^T A_B^{-1} b = \lambda^T b = b^T \lambda$$

Além disso, numa solução ótima o custo reduzido das variáveis básicas é igual a zero, pois

$$\overline{c}_B = c_B^T - \lambda^T A_B = c_B^T - c_B^T A_B^{-1} A_B = c_B^T - c_B^T = 0$$

Como o custo reduzido das variáveis não-básicas é maior ou igual a zero, isto é,

$$\overline{c}_N = c_N^T - \lambda^T A_N \geq 0 \Rightarrow c_N^T \geq \lambda^T A_N$$

então c^T pode ser reescrito como

$$c^T = [c_B^T | c_N^T] \geq [c_B^T | \lambda^T A_N] = [c_B^T A_B^{-1} A_B | \lambda^T A_N] = [\lambda^T A_B | \lambda^T A_N] = \lambda^T A$$

Portanto, como $A^T \lambda \leq c$, então $\lambda^T b$ é uma solução viável e ótima para o dual do programa P.

A representação tabular de um programa linear a partir da divisão das variáveis entre básicas e não-básicas (e que será utilizada no Algoritmo Simplex) é dada como abaixo

	x_B^T	x_N^T	$c^T x$
	c_B^T	c_N^T	
x_B	A_B	A_N	b

Tabela 5.1: Representação tabular de um programa linear a partir da separação das variáveis em básicas e não-básicas

[1] Iremos denominar este passo como etapa de *pricing* para referenciá-lo em outros algoritmos.

Como nem todo programa linear possui solução viável ou solução ótima limitada, antes de finalmente descrever o Algoritmo Simplex é necessário definir o conceito de programa linear *ilimitado*.

Definição 11. *Um programa linear é denominado* ilimitado *ou* sem solução limitada *se para toda solução viável x, há outra solução viável x' tal que $c^T x' < c^T x$.*

Assim sendo, podemos finalmente expôr o algoritmo Simplex como abaixo (Algoritmo 12). Um exemplo de execução da primeira iteração do algoritmo é apresentado a seguir.

Maiores detalhes sobre os passos do algoritmo podem ser consultados em Matoušek e Gärtner (2007).

Algoritmo 12: SIMPLEX

Entrada: Matriz A de dimensão $m \times n$, e vetores b e c de dimensões m e n, respectivamente.

Saída: A solução ótima do programa linear definido por A, b e c, tal que $Ax = b$ e $c^T x$ é mínimo.

Início

Monte a representação tabular do programa como na Tabela 5.1.

Encontre uma solução básica viável inicial x. Se não for possível determiná-la, então devolva "*programa linear sem solução*".

Faça

Compute A_B^{-1}, λ^T e $\bar{c} = c_N^T - \lambda^T A_N$.

Seja $i \in N$ tal que $\overline{c}_i$ é mínimo. Selecione x_i associado a $\bar{c}_i$ para fazer parte da solução básica atual x.

Escolha $j \in \{1, ..., m\}$ tal que $\frac{\overline{b}_j}{y_{ji}} \geq 0$ e $\frac{\overline{b}_j}{y_{ji}}$ é mínimo, para $Y = A_B^{-1} A_N$, $\overline{b} = A_B^{-1} b$, $\overline{b}_j \in \overline{b}$ e y_{ji} sendo um valor da matriz Y. Se nenhum destes valores atende a condição de não-negatividade, então devolva *programa linear ilimitado*.

A linha j da representação tabular do programa linear dado como entrada está associada a uma variável básica x_k, para $k \in B$. A variável x_i irá substituir x_k no conjunto de variáveis básicas pois x_k é a variável que mais limita o aumento do valor de x_i, conforme o teste de razão feito em $\frac{\overline{b}_j}{y_{ji}}$.

Atualize os índices dos conjuntos B e N, isto é, remova j de B e adicione-o em N, e remova i de N e adicione-o em B.

enquanto *houver $\overline{c}_i \in \overline{c}$, tal que $\overline{c}_i < 0$ para algum $i \in N$*;

Devolva x

Exemplo Seja o seguinte programa linear P de minimização e seu respectivo dual D[2].

[2]Exemplo disponível em http://www.cengage.com/resource_uploads/downloads/1305658000_528714.pdf

$$\begin{aligned}\text{Minimizar } z &= 0.12x_1 + 0.15x_2\\ \text{sujeito a } 60x_1 + 60x_2 &\geq 300\\ 12x_1 + 6x_2 &\geq 36\\ 10x_1 + 30x_2 &\geq 90\\ x_i &\geq 0, \text{ para } i \in \{1,2\}\end{aligned}$$

$$\begin{aligned}\text{Maximizar } w &= 300y_1 + 36y_2 + 90y_3\\ \text{sujeito a } 60y_1 + 12y_2 + 10y_3 &\leq 0.12\\ 60y_1 + 6y_2 + 30y_3 &\leq 0.15\\ y_i &\geq 0, \text{ para } i \in \{1,2,3\}\end{aligned}$$

Primeiramente é necessário deixar o programa dual D na forma padrão, isto é:

$$\begin{aligned}\text{Maximizar } w &= 300y_1 + 36y_2 + 90y_3 + 0y_4 + 0y_5\\ \text{sujeito a } 60y_1 + 12y_2 + 10y_3 + y_4 &= 0,12\\ 60y_1 + 6y_2 + 30y_3 + y_5 &= 0,15\\ y_i &\geq 0, \text{ para } i \in \{1,2,3,4,5\}\end{aligned}$$

A Tabela 5.2 contém a representação tabular do programa D. Como a matriz formada pelas colunas referentes as variáveis y_4 e y_5 é uma matriz não-singular, então esta será a matriz A_B inicial. Assim, temos que $B = \{4,5\}$ e $N = \{1,2,3\}$.

	y_1	y_2	y_3	y_4	y_5	
	-300	-36	-90	0	0	w
y_4	60	12	10	1	0	0,12
y_5	60	6	30	0	1	0,15

Tabela 5.2: Representação tabular do programa linear P

Temos que:

$$c_B^T = \begin{bmatrix} 0 & 0 \end{bmatrix}$$

$$A_B^{-1} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

$$\lambda^T = \begin{bmatrix} 0 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 0 \end{bmatrix}$$

$$\lambda^T A_N = \begin{bmatrix} 0 & 0 \end{bmatrix} \begin{bmatrix} 60 & 12 & 10 \\ 60 & 6 & 30 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix}$$

$$\overline{c} = \begin{bmatrix} -300 & -36 & -90 \end{bmatrix} - \begin{bmatrix} 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} -300 & -36 & -90 \end{bmatrix}$$

$$Y = A_B^{-1} A_N = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 60 & 12 & 10 \\ 60 & 6 & 30 \end{bmatrix} = \begin{bmatrix} 60 & 12 & 10 \\ 60 & 6 & 30 \end{bmatrix}$$

$$\overline{b} = A_B^{-1} b = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 0,12 & 0,15 \end{bmatrix} = \begin{bmatrix} 0,12 & 0,15 \end{bmatrix}$$

A variável y_1 irá entrar na base, pois $\min\{\overline{c}\} = \overline{c}_1 = -300$

A variável y_4 irá sair da base, pois $j = 1$, que corresponde a primeira linha da representação tabular, que está associada a y_4: $\min\left\{\frac{\overline{b}_1}{y_{11}}, \frac{\overline{b}_2}{y_{21}}\right\} = \min\left\{\frac{0,12}{60}, \frac{0,15}{60}\right\} = \frac{0,12}{60}$

Portanto, $B = \{1, 5\}$ e $N = \{4, 2, 3\}$. Como $\min \overline{c} < 0$, então o algoritmo irá para a próxima iteração, pois a solução básica atual ainda não é ótima.

Ao final da execução do algoritmo sobre o programa D, este terá solução ótima com valor $w = 33/50$, onde $B = \{1, 3\}$ e $N = \{4, 2, 5\}$ e $y_1 = 7/4000$, $y_3 = 3/2000$ e $y_2 = y_4 = y_5 = 0$.

A solução ótima do programa primal, por sua vez, corresponde a $x_1 = 3$ e $x_2 = 2$, e como $c^T x = b^T y$, então

$$\begin{bmatrix} 0,12 & 0,15 \end{bmatrix} \begin{bmatrix} 3 \\ 2 \end{bmatrix} = \begin{bmatrix} 300 & 36 & 90 \end{bmatrix} \begin{bmatrix} 7/4000 \\ 0 \\ 3/2000 \end{bmatrix} = 33/50$$

Para maiores detalhes sobre como obter a solução ótima do programa primal a partir do seu dual e vice-versa, verificar em Matoušek e Gärtner (2007).

5.4 Branch-and-Bound para PLI

Programas lineares inteiros podem ser solucionados por meio de *Branch-and-Bound* (conforme pode ser verificado em Papadimitriou e Steiglitz (1998)). Por exemplo, dado o seguinte programa linear inteiro:

$$\begin{aligned} \textbf{Instância Inicial:}\ & \text{Minimizar } z = c^T x = c(x) \\ & \text{sujeito a: } Ax \leq b \\ & x \in \mathbb{Z}_+^n \end{aligned}$$

Seja y uma solução viável ótima não inteira da Instância Inicial. Se os valores das variáveis de y forem inteiros, então não é necessário criar novas instâncias; caso contrário, seja y_1 uma das variáveis de y que não obteve valor inteiro nesta solução. Neste caso, criam-se duas novas instâncias como abaixo:

$$\begin{aligned} \textbf{Instância } \boldsymbol{L_1}\text{:}\ & \text{Minimizar } z = c^T x = c(x) \\ & \text{sujeito a: } Ax \leq b \\ & x \in \mathbb{Q}_+^n \\ & y_1 \leq \lfloor y_1 \rfloor \end{aligned}$$

$$\begin{aligned} \textbf{Instância } \boldsymbol{R_1}\text{:}\ & \text{Minimizar } z = c^T x = c(x) \\ & \text{sujeito a: } Ax \leq b \\ & x \in \mathbb{Q}_+^n \\ & y_1 \geq \lceil y_1 \rceil \end{aligned}$$

Escolhe-se uma das novas instâncias para ser solucionada, por exemplo, a Instância L_1. Sua respectiva solução é denotada por w. Desta maneira, se w tiver algum valor não inteiro, a etapa de ramificação ocorre da mesma maneira como na etapa anterior (em que a Instância Inicial foi subdividida). Caso contrário, a solução encontrada não é viável ou é inteira.

Em algum ponto da árvore de soluções gerada pelo algoritmo, a melhor solução encontrada até então – aquela onde a função objetivo possui o menor valor – tem custo z_m. Assim, para uma instância J e sua respectiva solução (denotada por $\overline{x}$), se $c(\overline{x}) \geq z_m$, então a referida instância não é ramificada.

Sejam X o conjunto de soluções parciais, x_* a melhor solução até então, e z_m o custo de x_*. Dessa maneira, um pseudocódigo para o algoritmo *Branch-and-Bound* é descrito no Algoritmo 13, onde a etapa de encontrar a solução ótima não inteira de uma instância é executada pelo algoritmo Simplex.

Algoritmo 13: BRANCH-AND-BOUND(G)

Entrada: Matriz A de dimensão $m \times n$, e vetores b e c de dimensões m e n, respectivamente.
Saída: A solução ótima do programa linear definido por A, b e c, tal que $Ax = b$, $c^T x$ é mínimo e x é de inteiros.
Início
  $P \leftarrow$ programa linear formulado a partir de A, b e c.
  $X \leftarrow \emptyset$
  Adicione P ao conjunto X.
  $z_m \leftarrow 0$
  $x_* \leftarrow \emptyset$
  Enquanto $X \neq \emptyset$
    Selecione uma instância de X para ser solucionada, e remova-a de X.
    Encontre uma solução $\overline{x}$ para a instância selecionada.
    Se $\overline{x}$ *é solução viável* **então**
      Se $\overline{x}$ *é solução inteira* **então**
        Se $z_m = 0$ *ou* $c(\overline{x}) < z_m$ **então**
          $x_* \leftarrow \overline{x}$
          $z_m \leftarrow c(\overline{x})$
      Senão se $c(\overline{x}) < z_m$ **então**
        Seja $\overline{x}_i$ uma variável que não obteve valor inteiro nesta solução. Crie duas novas instâncias a partir da instância selecionada, tais que em uma delas será acrescentada a restrição $\overline{x}_i \leq \lfloor \overline{x}_i \rfloor$, e na outra será acrescentada a restrição $\overline{x}_i \geq \lceil \overline{x}_i \rceil$. Adicione as novas instâncias a X.
Devolva x_*.

Exemplo A Figura 11 mostra um exemplo da árvore de soluções do Algoritmo 13 aplicado sobre o programa linear inteiro abaixo, onde cada nó indica a solução e respectivo valor obtido para determinada instância.

$$\begin{aligned} \text{Minimizar } & z = 2x_1 + 10x_2 + 8x_3 \\ \text{sujeito a } & x_1 + x_2 + x_3 \geq 6 \\ & x_1 + 2x_3 \geq 8 \\ & -x_1 + 2x_2 + 2x_3 \geq 4 \\ & x_i \in \mathbb{Z}_+^n, \text{ para } i \in \{1, 2, 3\} \end{aligned}$$

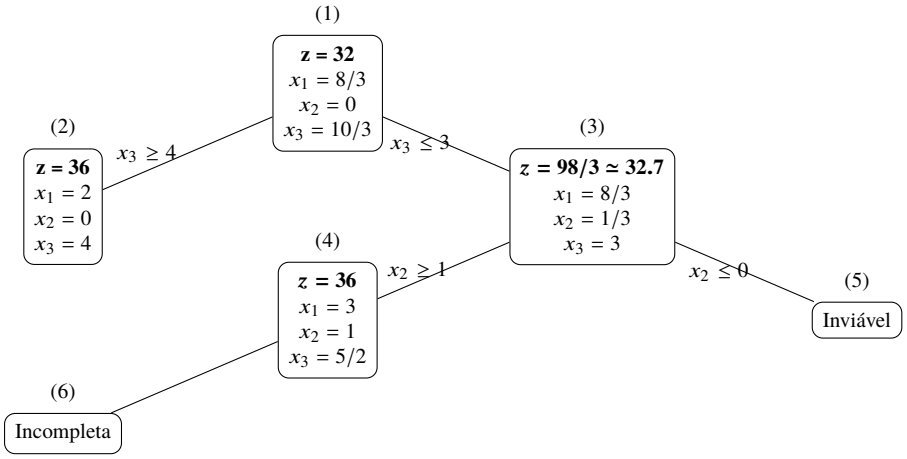


Figura 11: Árvore de busca de exemplo de execução do Algoritmo 13

Neste exemplo, a solução inicial está apresentada no nó (1). Uma solução inteira com $z = 36$ foi obtida em (2) quando acrescentou-se a restrição $x_3 \geq 4$ ao problema, portanto o valor desta solução torna-se um limitante inferior para as outras iterações, onde $z_m = 36$ e $x_* = (x_1 = 2, x_2 = 0, x_3 = 4)$. Esta é a solução ótima do programa linear dado como entrada neste exemplo, pois não foi possível obter uma solução melhor do que esta a partir de (3). Isto é, como o valor de z obtido em (4) foi igual a z_m e o valor de uma solução não inteira é limitante inferior para a solução inteira, então qualquer instância obtida a partir desta não será melhor do que a melhor solução até então, portanto não foi necessário continuar a execução a partir do nó (4).

5.5 Formulações para o problema de coloração de grafos

Dado um grafo G tal que $n = |V(G)|$ e $m = |E(G)|$, uma primeira formulação do problema de coloração de grafos como um problema de Programação Linear Inteira é expressa como abaixo:

$$\text{Minimizar} \sum_{j=1}^{n} x_j \tag{5.5a}$$

$$\text{sujeito a:} \sum_{j=1}^{n} y_{vj} = 1, \quad \text{para todo } v \in V(G) \text{ e todo } j \in \{1, ..., n\} \tag{5.5b}$$

$$y_{vj} + y_{uj} \leq x_j, \quad \text{para todo } \{v, u\} \in E(G) \text{ e todo } j \in \{1, ..., n\} \tag{5.5c}$$

$$y_{vj} \in \{0, 1\}, \quad \text{para todo } v \in V(G) \text{ e todo } j \in \{1, ..., n\} \tag{5.5d}$$

$$x_j \in \{0, 1\}, \quad \text{para todo } j \in \{1, ..., n\} \tag{5.5e}$$

Nesta formulação, tem-se uma variável binária x_j para cada cor, de modo que x_j recebe valor 1 se a cor j é utilizada na solução, e 0 caso contrário. Tem-se também as variáveis binárias y_{vj} que indicam se a cor j está atribuída ao vértice $v \in V(G)$ na solução. O problema é de minimização, conforme declarado na função objetivo (5.5a), visto que a quantidade de cores

utilizadas pela solução ótima deve ser mínima. A restrição 5.5b determina que um vértice v deve ter exatamente uma cor atribuída a si, e cada restrição em 5.5c determina que vértices vizinhos v e u não podem receber a mesma cor j. Os conjuntos de restrições 5.5d e 5.5e representam as restrições de que as variáveis são binárias.
Esta formulação contém $n(n+1)$ variáveis, para $n = |V(G)|$, e no máximo $\frac{3n^2+3n}{2}$ restrições, pois o número exato de restrições é $n^2 + n + m$, e o número máximo de arestas em um grafo é igual a $\frac{n(n-1)}{2}$, logo,

$$n^2 + 2n + m \leq n^2 + 2n + \frac{n(n-1)}{2} = \frac{2n^2 + 4n + n(n-1)}{2} = \frac{n(2n + 4 + n - 1)}{2} = \frac{3n^2 + 3n}{2}$$

Assim, a quantidade de variáveis e restrições é polinomial em relação ao tamanho da entrada.

Dizemos que a solução de um problema de otimização é simétrica quando esta solução pode ser representada de diferentes maneiras ao permutarem-se os valores atribuídos às variáveis. Por exemplo, as colorações da Figura 12 são soluções simétricas.

(a) Solução 1 (b) Solução 2

Figura 12: Exemplo de soluções simétricas

Esta formulação pode ser representada por duas matrizes $Y \in \mathbb{Z}^{n \times n}$ e $X \in \mathbb{Z}^n$ tais que, para $y_{vj} \in Y$ e $x_j \in X$

$$y_{vj} = \begin{cases} 1, & \text{se a cor } j \text{ está atribuída ao vértice } v \\ 0, & \text{caso contrário} \end{cases} \tag{5.6}$$

$$x_j = \begin{cases} 1, & \text{se a cor } j \text{ está atribuída a algum vértice} \\ 0, & \text{caso contrário} \end{cases} \tag{5.7}$$

Sendo assim, a simetria entre as possíveis soluções do programa linear formulado em 5.5 é um fator que pode dificultar o uso desta formulação na prática, pois para uma k-coloração onde cada cor está associada a um valor do conjunto $\{1, ..., n\}$, temos que uma mesma coloração pode ser representada como uma permutação das colunas da matriz Y. Ou seja, há $P(n, k) = \frac{n!}{(n-k)!}$ soluções simétricas entre si, o que torna a árvore de soluções do algoritmo *Branch-and-Bound* ainda maior, e portanto, é prejudicial para o desempenho do algoritmo. Lewis (2016) expõe alternativas para contornar essa simetria. Entretanto, o programa linear formulado em 5.5 ainda apresenta-se pouco eficiente em relação a outras formulações, como a formulação exibida na subseção 5.5.1.

5.5.1 Formulação por conjuntos independentes maximais

O problema de coloração de vértices consiste em encontrar uma cobertura de conjuntos, onde os vértices devem estar cobertos na menor quantidade de conjuntos independentes possível.

Com isso, Mehrotra e Trick (1995) propuseram uma formulação em programa linear como a seguir, dado um grafo G.

$$\text{Minimizar} \sum_{S \in \mathbf{S}} x_S \tag{5.8a}$$

$$\text{sujeito a:} \sum_{\{S : v \in S\}} x_S \geq 1, \qquad \text{para todo } v \in V(G) \tag{5.8b}$$

$$x_S \in \{0, 1\}, \qquad \text{para todo } S \in \mathbf{S} \tag{5.8c}$$

Neste programa linear, $\mathbf{S}$ é o conjunto de todos os conjuntos independentes maximais de G e x_S é a variável binária correspondente a um conjunto $S \in \mathbf{S}$, de maneira que x_S recebe valor 1 se o conjunto S faz parte da solução, e 0 caso contrário. A função objetivo 5.8a diz que o número cromático é determinado pela menor partição dos vértices em conjuntos independentes. Cada restrição em 5.8b indica que cada vértice v pertence a pelo menos um conjunto $S \in \mathbf{S}$. Esta formulação pode ser obtida a partir da descrita em 5.5 por meio de decomposição linear de Dantzig-Wolfe, conforme explicado por Schindl (2004).

O número de variáveis desta formulação pode ser muito maior em relação à formulação anterior, visto que como dito anteriormente, o número de conjuntos independentes maximais em um grafo pode ser exponencial em relação a n. Entretanto, diferentemente da formulação 5.5, a formulação 5.8 evita o problema de simetria nas soluções (Lewis, 2016).

Coloração fracionária

Ao obter uma solução ótima não inteira para o programa linear 5.8, tem-se uma *coloração fracionária* de G. Isto é, como cada conjunto independente representa uma cor, será atribuído um conjunto de cores a cada vértice ao invés de apenas uma, e para vértices que são vizinhos, o conjunto de cores destes vértices será disjunto. O valor da solução ótima de uma coloração fracionária é denominado *número cromático fracionário* de G, denotado por $\chi_f(G)$. Por exemplo, para o grafo C_5 (Figura 13) e os conjuntos independentes $S_1 = \{AC\}$, $S_2 = \{AD\}$, $S_3 = \{BD\}$, $S_4 = \{BE\}$ e $S_5 = \{CE\}$, tem-se a formulação como em 5.9.

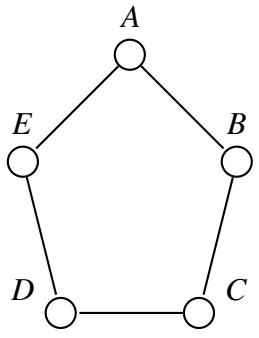


Figura 13: Grafo C_5

$$
\begin{aligned}
\text{Minimizar } & \sum_{i=1}^{5} x_{S_i} \\
\text{sujeito a: } & x_{S_1} + x_{S_2} \geq 1 \\
& x_{S_3} + x_{S_4} \geq 1 \\
& x_{S_1} + x_{S_5} \geq 1 \\
& x_{S_2} + x_{S_3} \geq 1 \\
& x_{S_4} + x_{S_5} \geq 1 \\
& x_{S_i} \geq 0, \qquad \text{para todo } i \in \{1, ..., 5\}
\end{aligned}
\tag{5.9}
$$

Na solução ótima deste programa linear, cada variável x_{S_i} tem valor igual a 0,5, o que significa que cada cor contribui com 0,5 na coloração total. Dessa forma, $\chi_f(G)$ para este exemplo é igual a 2,5. A representação da solução no grafo C_5 é ilustrada na Figura 14, onde as cores de índices 1, 2, 3, 4 e 5 estão representadas pelos padrões "linhas horizontais", "quadriculado", "pontilhado", "linhas verticais" e "hachurado" e representam os conjuntos S_1, S_2, S_3, S_4 e S_5, respectivamente.

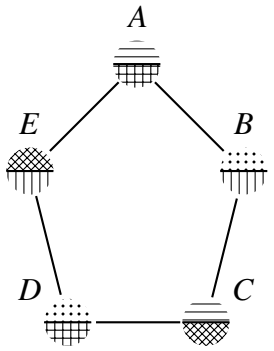


Figura 14: Exemplo de uma coloração fracionária

Lund e Yannakakis (1994) provaram que o *gap* de integralidade entre $\chi_f(G)$ e $\chi(G)$ é $O(\log n)$. Contudo, determinar $\chi_f(G)$ também é um problema NP-Difícil (Held et al., 2011).

5.6 O método Geração de Colunas

Formulações com grande quantidade de variáveis como as citadas acima, apesar de comprometerem o desempenho de técnicas que solucionam programas lineares inteiros, ainda são utilizadas pela literatura, pois segundo Barnhart et al. (1998), podem apresentar limitantes melhores do que formulações com menos variáveis, bem como podem eliminar simetrias prejudiciais ao *Branch-and-Bound*. Além disso, muitas vezes é inevitável que o programa linear inteiro formulado a partir do problema que será solucionado tenha muitas variáveis. Contudo, vale ressaltar que muitas vezes, grande parte das variáveis do programa pode não fazer parte da solução ótima. Assim, uma maneira de contornar este problema é iniciar o procedimento de solução do programa de entrada com um pequeno subconjunto das variáveis, e inserir novas variáveis iterativamente.

O método de Geração de Colunas (*Column Generation*) é um procedimento de solução de programas lineares com muitas variáveis que parte dessa ideia. O método leva este nome pois

na representação tabular de um programa linear no algoritmo *Simplex*, as colunas representam as variáveis.

Um algoritmo baseado em Geração de Colunas decompõe o programa linear em dois outros programas lineares, denominados *problema mestre* e *subproblema gerador de colunas*, este também denominado como etapa de *pricing* do algoritmo. O programa linear referente ao problema mestre é uma reformulação do programa original, mais restrita. O subproblema gerador de colunas, por outro lado, é aquele que irá determinar as variáveis que serão gradativamente acrescentadas ao problema mestre.

Mehrotra e Trick (1995) observam que a eficácia da técnica Geração de Colunas depende de uma boa exploração das características do problema a ser solucionado. O problema mestre e os subproblemas geradores de colunas podem ser obtidos a partir da decomposição linear de Dantiz-Wolfe, descrito no trabalho de Andrade et al. (2006).

Dado um programa linear inteiro com solução ótima, seja n sua quantidade de variáveis. O método *Geração de Colunas* é genericamente descrito como no Algoritmo 14.

Algoritmo 14: Geração de Colunas

Entrada: Matriz A de dimensão $m \times n$, e vetores b e c de dimensões m e n, respectivamente.

Saída: A solução ótima inteira do programa linear definido por A, b e c, tal que $Ax = b$ e $c^T x$ é mínimo.

Início

Seja P o programa linear formulado a partir de A, b e c.

Obtenha o problema mestre de P pela decomposição linear de Dantzig-Wolfe.

Seja o *problema mestre restrito*, denotado PMR, o programa linear obtido do problema mestre tal que o conjunto das variáveis de PMR corresponde a um subconjunto das variáveis do problema mestre e há pelo menos uma solução válida não inteira para o PMR a partir desse subconjunto.[3]

Faça

Resolva o PMR utilizando Simplex.

Seja $N \subseteq \{1, ..., n\}$ o conjunto de índices das variáveis não-básicas tal que N contém tanto os índices das variáveis não-básicas de P que já estão em PMR como aquelas que não estão. Formule o subproblema gerador de colunas, denotado SGC, a partir da etapa de *pricing* do Simplex, isto é,

$$\overline{c} = c_N^T - \lambda^T A_N$$

$$\overline{c}_i \in \overline{c} \text{ tal que } \overline{c}_i \text{ é mínimo e } i \in N$$

Resolva SGC. Se $\overline{c}_i < 0$, então acrescente a variável encontrada em SGC a PMR. Caso contrário, a solução de PMR também é solução ótima para o programa linear dado como entrada.

enquanto *houver* $\overline{c}_i < 0$ *em* $\overline{c}$;

Devolva x

---

[3] Barnhart et al. (1998) explicam como determinar este subconjunto de variáveis.

As restrições do subproblema SGC podem ser definidas manualmente ou a estrutura do mesmo pode ser alterada uma vez que se perceba que estas modificações permitem que as colunas sejam geradas de maneira mais eficiente.

A Seção 5.7.1 contém uma explicação sobre como a Geração de Colunas é aplicada na formulação 5.8.

5.6.1 Branch-and-Price

Um algoritmo baseado em *Branch-and-Price* soluciona um programa linear inteiro combinando os métodos *Branch-and-Bound* e Geração de Colunas. A etapa de ramificação de uma instância pode ocorrer como no Algoritmo 13. Entretanto, há formulações que estabelecem outras regras de ramificação para explorar melhor o problema que será solucionado.
Seja X o conjunto de instâncias criado no Algoritmo 13, onde cada elemento é uma instância do problema mestre restrito. Um fluxograma da ideia do funcionamento de um algoritmo baseado em *Branch-and-Price* é mostrado na Figura 15. Neste fluxograma, está implícita a etapa de *bounding* do *Branch-and-Bound* no momento em que a instância é ramificada ou não.

5.7 Algoritmos baseados em PLI

5.7.1 O Algoritmo de Mehrotra e Trick (1995)

O primeiro trabalho que utilizou o método *Branch-and-Price* para o problema da coloração foi proposto por Mehrotra e Trick (1995). Ali, utiliza-se a formulação 5.8 como problema mestre, desconsiderando as restrições de integralidade.
Dado um grafo G, seja $\mathbf{S}$ o conjunto de conjuntos independentes maximais de G. Para determinar o conjunto inicial de variáveis $\bar{\mathbf{S}} \in \mathbf{S}$ para o programa linear referente problema mestre restrito, isto é,

$$\text{Minimizar} \quad \sum_{s \in \bar{\mathbf{S}}} x_S \tag{5.10a}$$

$$\text{sujeito a:} \quad \sum_{\{S : v \in S\}} x_S \geq 1, \qquad \text{para todo } v \in V(G) \tag{5.10b}$$

$$x_S \geq 0, \qquad \text{para todo } S \in \bar{\mathbf{S}}$$

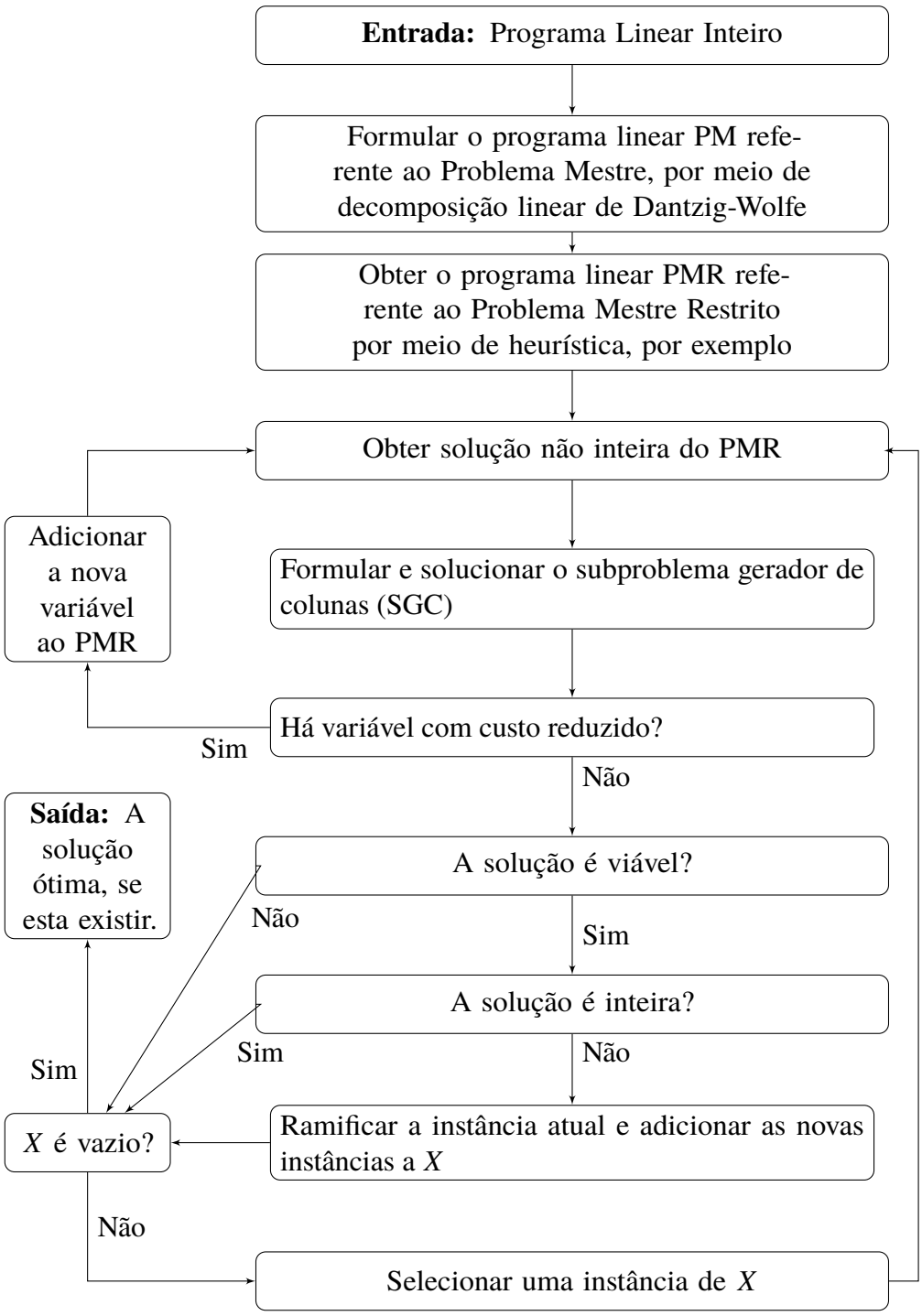


Figura 15: Fluxograma simplificado do algoritmo *Branch-and-Price*

aplica-se a heurística gulosa para o problema do conjunto independente de peso máximo descrita por (Algoritmo 15) repetidas vezes no grafo de entrada, até que todos os vértices estejam cobertos por algum conjunto.

Algoritmo 15: HEURÍSTICA GULOSA PARA O CIPM(G)

Entrada: Um grafo G
Saída: Um conjunto independente ponderado.
Início
  $I \leftarrow \emptyset$
  Faça
    Escolha um vértice $v \in V(G)$ de peso máximo.
    Adicione v a I.
    Remova $N_G(v)$ de $V(G)$.
  enquanto $V(G) \neq \emptyset$;
Devolva I

Procura-se a solução ótima não inteira para o problema mestre restrito, obtendo-se assim o vetor de multiplicadores duais λ para a formulação do subproblema gerador de colunas.

Como visto anteriormente, o subproblema gerador de colunas é obtido a partir da etapa de *pricing* do Simplex no problema mestre, ou seja, a etapa de encontrar o menor custo reduzido do vetor $c_N^T - \lambda^T A_N$.

No problema mestre (5.8) toda variável de c_N^T tem coeficiente de valor igual a 1 associada a si, e cada variável está associada a uma coluna de A_N, que por sua vez, descreve um conjunto independente S representado como vetor binário $z \in \{0,1\}^n$. Logo, o subproblema gerador de colunas será encontrar o menor valor $1 - \lambda^T z$, ou seja, para λ_v sendo o valor dual obtido para o vértice v na solução ótima não inteira do problema mestre restrito, o subproblema gerador de colunas será

$$\min\{1 - \sum_{v \in V(G)} \lambda_v z_v\} \equiv 1 - \max\{\sum_{v \in V(G)} \lambda_v z_v\}$$

Assim, o subproblema gerador de colunas consistirá no problema do conjunto independente máximo ponderado, isto é,

$$\text{Maximizar} \sum_{v \in V(G)} \lambda_v z_v \tag{5.11a}$$

$$z_u + z_v \leq 1, \qquad \text{para todo } \{u,v\} \in E(G) \tag{5.11b}$$

$$z_v \in \{0,1\}, \qquad \text{para todo } v \in V(G)$$

Nesta formulação, cada restrição em 5.11b denota que vértices adjacentes não devem fazer parte do mesmo conjunto independente máximo. Se a solução ótima do subproblema gerador é maior do que 1, então todo z_v com valor 1 corresponde a um vértice do conjunto independente que deve ser adicionado a $\bar{\mathbf{S}}$, ou seja, a variável que será adicionada é $S = \{v \in V(G) \text{ tal que } z_v = 1\}$. Caso contrário, então não há conjunto independente que possa melhorar a solução atual do problema mestre.

Exemplo Um exemplo pequeno do método de Geração de Colunas a fim de encontrar uma coloração ótima no grafo G da Figura 16 e seus respectivos conjuntos independentes maximais $S_1 = \{0,3\}$, $S_2 = \{0,5\}$, $S_3 = \{0,4\}$, $S_4 = \{1,5\}$, $S_5 = \{2,3\}$ e $S_6 = \{2,5\}$ é dado a seguir.

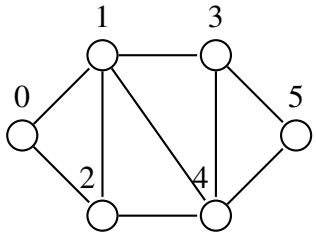


Figura 16: Grafo de exemplo que será colorido pelo algoritmo de Mehrotra e Trick (1995)

O problema mestre restrito constitui-se das variáveis referentes aos conjuntos independentes maximais S_1, S_3, S_4 e S_6:

$$\begin{aligned}
\text{Minimizar } & x_{S_1} + x_{S_3} + x_{S_4} + x_{S_6} \\
\text{sujeito a: } & x_{S_1} + x_{S_3} \geq 1 \\
& x_{S_4} \geq 1 \\
& x_{S_6} \geq 1 \\
& x_{S_1} \geq 1 \\
& x_{S_3} \geq 1 \\
& x_{S_4} + x_{S_6} \geq 1 \\
& x_{S_1}, x_{S_3}, x_{S_4}, x_{S_6} \geq 0
\end{aligned} \tag{5.12}$$

A solução ótima deste programa é dada por $x_{S_1} = 1$, $x_{S_3} = 1$, $x_{S_4} = 1$ e $x_{S_6} = 1$, o valor ótimo é igual a 4 e $\lambda^T = (0, 1, 1, 1, 1, 0)$. Dessa forma, o subproblema gerador de colunas corresponde a:

$$\begin{aligned}
\text{Maximizar } & 0v_0 + 1v_1 + 1v_2 + 1v_3 + 1v_4 + 0v_5 \\
\text{sujeito a: } & v_0 + v_1 \leq 1 \\
& v_0 + v_2 \leq 1 \\
& v_1 + v_2 \leq 1 \\
& v_1 + v_3 \leq 1 \\
& v_1 + v_4 \leq 1 \\
& v_2 + v_4 \leq 1 \\
& v_3 + v_4 \leq 1 \\
& v_3 + v_5 \leq 1 \\
& v_4 + v_5 \leq 1 \\
& v_i \in \{0, 1\}, \qquad \text{para todo } i \in \{0, \ldots, 5\}
\end{aligned} \tag{5.13}$$

A solução ótima do programa 5.13 é dada por $v_0 = v_1 = v_4 = v_5 = 0$ e $v_2 = v_3 = 1$. Como o valor ótimo deste programa é igual a 2, então a variável correspondente ao conjunto $S_5 = \{2, 3\}$ será adicionada ao problema restrito na próxima iteração:

$$\begin{aligned} \text{Minimizar } & x_{S_1} + x_{S_3} + x_{S_4} + x_{S_5} + x_{S_6} \\ \text{sujeito a: } & x_{S_1} + x_{S_3} \geq 1 \\ & x_{S_4} \geq 1 \\ & x_{S_5} + x_{S_6} \geq 1 \\ & x_{S_1} + x_{S_5} \geq 1 \\ & x_{S_3} \geq 1 \\ & x_{S_4} + x_{S_6} \geq 1 \\ & x_{S_1}, x_{S_3}, x_{S_4}, x_{S_5}, x_{S_6} \geq 0 \end{aligned} \tag{5.14}$$

A solução ótima deste programa, por sua vez, é dada por $x_1 = x_6 = 0$ e $x_3 = x_4 = x_5 = 1$, o valor ótimo é igual a 3 e $\lambda^T = (0, 0, 0, 1, 1, 1)$. Portanto, o subproblema gerador de colunas corresponde a:

$$\begin{aligned} \text{Maximizar } & 0v_0 + 0v_1 + 0v_2 + 1v_3 + 1v_4 + 1v_5 \\ \text{sujeito a: } & v_0 + v_1 \leq 1 \\ & v_0 + v_2 \leq 1 \\ & v_1 + v_2 \leq 1 \\ & v_1 + v_3 \leq 1 \\ & v_1 + v_4 \leq 1 \\ & v_2 + v_4 \leq 1 \\ & v_3 + v_4 \leq 1 \\ & v_3 + v_5 \leq 1 \\ & v_4 + v_5 \leq 1 \\ & v_i \in \{0, 1\}, \qquad \text{para todo } i \in \{0, ..., 5\} \end{aligned} \tag{5.15}$$

A solução ótima de 5.15 é dada por $v_0 = v_1 = v_2 = v_4 = v_5 = 0$ e $v_3 = 1$. Porém, como o respectivo valor da solução ótima é igual a 1, então a solução atual do problema mestre restrito também é ótima. Como esta solução é inteira e a instância inicial não foi ramificada, então esta também é a solução ótima global. A respectiva coloração ótima de G está representada na Figura 17.

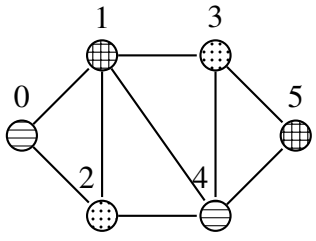


Figura 17: Coloração ótima obtida no grafo da Figura 16

Etapa de ramificação À medida que o conjunto de instâncias é gerado na árvore de solução do algoritmo de Mehrotra e Trick (1995), as restrições que são acrescentadas a cada instância para forçar a integralidade do valor das variáveis são estabelecidas conforme a recorrência de Zykov apresentada no Capítulo 4. Isto é, seja uma instância deste conjunto que obteve solução não inteira e sejam dois conjuntos independentes maximais S_1 e S_2. Como esta instância também representa uma coloração fracionária, então há dois vértices u e v tais que, $u \in S_1 \cap S_2$ e $v \in S_1 \setminus S_2$ e consequentemente, ao menos um dentre x_{s_1} ou x_{s_2} tem valor fracionário. Isto significa que tanto o vértice u pode ser colorido com a mesma cor dos vértices de S_1 (e portanto, com a mesma cor do vértice v), como u pode receber a mesma cor dos vértices de S_2. Portanto, em uma das novas instâncias será acrescentada uma restrição onde u e v devem receber a mesma cor, enquanto na outra será acrescentada uma restrição onde u e v devem receber cores diferentes. Mehrotra e Trick (1995) observam que esta maneira de ramificar uma instância na etapa de *branching* mantém a estrutura do problema pois não adiciona novas restrições ao problema mestre nem nos subproblemas geradores.

Como visto no Capítulo 4, a árvore de Zykov é binária. Logo, a árvore de soluções gerada pelo algoritmo *Branch-and-Price* que utiliza essa forma de ramificação também é uma árvore binária, pois cada instância é ramificada duas vezes.

Um exemplo de ramificação pode ser verificado no programa linear 5.9 para o grafo C_5 da Figura 13, e respectivos grafos obtidos pela recorrência de Zykov (1962) (Figura 18). Neste exemplo, a variável x_{S_1} terá seu valor ramificado, isto é, em uma das instâncias os vértices A e C terão a mesma cor (programa linear referente ao grafo (a)), enquanto na outra instância, A e C terão cores distintas (programa linear referente ao grafo (b)). Em (a), as variáveis x_{S_1}, x_{S_3} e x_{S_4} representam os conjuntos AC, D e BE, respectivamente. Em (b), as variáveis x_{S_2}, x_{S_3} e x_{S_4} e x_{S_5} representam os conjuntos AD, BD, BE e CE, respectivamente.

(a) G'_{AC} (b) G''_{AC}

Figura 18: Grafos obtidos pela ramificação de Zykov no grafo da Figura 13

$$
\begin{aligned}
\text{Minimizar } & x_{S_1} + x_{S_3} + x_{S_4} \\
\text{sujeito a: } & x_{S_1} \geq 1 \\
& x_{S_4} \geq 1 \\
& x_{S_3} \geq 1 \\
& x_{S_4} \geq 1 \\
& x_{S_i} \geq 0, \\
& \text{para todo } i \in \{1, 3, 4\}
\end{aligned}
$$

$$
\begin{aligned}
\text{Minimizar } & \sum_{i=2}^{5} x_{S_i} \\
\text{sujeito a: } & x_{S_2} \geq 1 \\
& x_{S_3} + x_{S_4} \geq 1 \\
& x_{S_5} \geq 1 \\
& x_{S_2} + x_{S_3} \geq 1 \\
& x_{S_4} + x_{S_5} \geq 1 \\
& x_{S_i} \geq 0, \\
& \text{para todo } i \in \{2, ..., 5\}
\end{aligned}
$$

Conclusões

De acordo com Held et al. (2011), o método proposto por Mehrotra e Trick (1995) e trabalhos subsequentes baseados neste método apresentam os melhores resultados experimentais no que diz respeito a coloração de grafos.

Os autores observam que melhoras podem ser alcançadas por meio de modificações no algoritmo que soluciona o subproblema gerador de colunas, isto é, o problema do conjunto independente máximo ponderado. Em seu trabalho, o método que soluciona esse problema é um algoritmo recursivo baseado na recorrência onde dado um grafo G e um vértice $v \in V(G)$, o conjunto independente máximo ponderado $CIPM$ de G não contém o vértice v, ou contém v e vértices de sua anti-vizinhança, isto é,

$$CIPM(G) = \max\{CIPM(G - v), CIPM(G[\{v\} \cup \overline{N}_G(v)])\}$$

Mehrotra e Trick (1995) observam que a ordem em que os vértices são percorridos nesta etapa é fundamental para que a rotina se torne mais eficiente. Neste caso, os autores notaram que considerar apenas o maior grau ou maior peso de um vértice durante a ordenação não é tão eficiente quanto considerar os dois simultaneamente.

Outras considerações feitas pelos autores dizem respeito a outras regras de ramificação de uma instância que obteve solução não inteira, bem como a geração de múltiplas variáveis ao invés de apenas uma a cada etapa de geração de colunas.

Trabalhos posteriores

Adaptações do método de Mehrotra e Trick (1995) foram propostas nos trabalhos de Gualandi e Malucelli (2012) e Malaguti et al. (2011), que conforme sugerido por seus predecessores, buscaram encontrar algoritmos mais eficientes para resolução do subproblema gerador de colunas. Malaguti et al. (2011) utilizam uma heurística baseada em busca tabu na resolução do problema do conjunto independente de peso máximo. Inicialmente, um número de iterações máximo é fixado para que a heurística proposta seja executada. Caso esta heurística tenha atingido esse número e uma coluna não tenha sido gerada, então a instância CIPM é finalmente solucionada como um programa linear. Gualandi e Malucelli (2012), por sua vez, propuseram a utilização de Programação com Restrições (*Constraint Programming*) na etapa de *pricing* do *Branch-and-Price*.

Resultados comparativos entre algoritmos baseados em *Branch-and-Price* e baseados em DSATUR podem ser verificados nos trabalhos de San Segundo (2012) e Furini et al. (2017). Tendo como base a implementação de Malaguti et al. (2011), os autores concluíram que algoritmos baseados em *Branch-and-Price* são atualmente os mais eficientes para instâncias difíceis do DIMACS, enquanto para grafos aleatórios de 60 a 80 vértices e densidades entre 0,1 e 0,9, as implementações baseadas em DSATUR apresentam-se mais eficientes.

5.8 Algoritmo do SageMath

Durante o processo de busca por *softwares* que solucionam instâncias do problema de coloração de grafos, nos deparamos com a implementação presente na ferramenta matemática de código-aberto SageMath [4]. A plataforma é abrangente em pacotes para cálculo de funções e algoritmos em grafos, incluindo o problema de coloração. As rotinas que encontram o número

[4]Disponível em http://www.sagemath.org/pt/

cromático e a coloração ótima de um grafo são baseadas em Programação Linear Inteira. Uma descrição simplificada do algoritmo que encontra o número cromático de um grafo G é dada no Algoritmo 16.

Algoritmo 16: $\chi(G, k)$ - SageMath

```
Entrada: Um grafo G e um inteiro k
Saída: O número cromático de G
Início
    Se k não é fornecido como entrada então
        Se V(G) é conjunto independente então
            Devolva 1
        Se G é bipartido então
            Devolva 2
        k ← max{ω(G), |V(G)|/α(G)}
        otima ← 0
        Enquanto otima ≠ 1
            t ← χ(G, k)
            Se t ≠ 0 então
                otima ← 1
                Devolva k
            Senão
                k ← k + 1
    Senão
        Remova todo vértice v ∈ V(G), tal que d_G(v) < k.
        Formule uma instância PLI como em 5.5, tendo k como limitante superior.
    Se A instância PLI tem solução então
        Devolva 1
    Senão
        Devolva 0
Devolva k
```

Como descrito no pseudo-código simplificado, dados como entrada um grafo G e um valor k, o algoritmo verifica se existe uma k-coloração; se k não é fornecido, então uma coloração ótima é buscada.

Ao invés de testarem-se os valores no intervalo $\{1, \ldots, k\}$, verifica-se primeiramente se o próprio grafo é um conjunto independente ou um grafo bipartido. Caso contrário, então o grafo é no mínimo 2-colorível. Para apertar ainda mais este limitante inferior, determina-se o valor mínimo de k como o máximo entre o tamanho da maior clique $\omega(G)$ e $|V(G)|/\alpha(G)$, tal que estes valores são encontrados por rotinas do próprio SageMath.

Em seguida a este passo, o algoritmo remove do grafo de entrada os vértices de grau menor a k, pois de acordo com o Teorema 18, isto não irá aumentar o número cromático de G.

Teorema 18. *Seja uma k-coloração de G e seja*

$$\{X \subseteq V(G) \text{ tal que para todo } v \in X,\ d_G(v) < k\}$$

Então $\chi(G - X) \leq k$.

Demonstração. Seja N_v o conjunto de vizinhos de v. Assim, se $|N_G(v)|$ tem $k - 1$ vértices, então $\chi(G[\{v\} \cup N_G(v)]) = k$, pois há uma cor em $G - N_G(v)$ não utilizada por nenhum $y \in N_G(v)$ que pode ser atribuída a v. □

Capítulo 6

Conclusão

Neste trabalho fez-se um estudo do problema de coloração de grafos, onde foram apresentados os principais algoritmos exatos mais recentes que o solucionam.

Inicialmente, apresentamos de maneira breve a origem histórica do problema e apresentamos limitantes e demonstrações conhecidos na literatura. Apresentamos também a coloração de vértices em grafos cordais, que é polinomial tanto nesta classe de grafos como em sua superclasse, que constitui-se dos grafos perfeitos.

Apresentamos alguns algoritmos exatos conhecidos na literatura, dentre eles, os baseados em Programação Dinâmica, *Branch-and-Bound* e Programação Linear Inteira. Os algoritmos do primeiro grupo possuem análise de pior caso. Por outro lado, os algoritmos baseados em *Branch-and-Bound* e Programação Linear Inteira possuem apenas resultados experimentais, pois a análise de pior caso nestes algoritmos é mais complexa. Assim, buscamos explicar seu funcionamento, acrescentando exemplos para facilitar a compreensão do leitor e intuições a partir dos resultados obtidos pelos autores. Introduzimos de maneira breve as informações necessárias para compreensão do funcionamento de algoritmos que solucionam instâncias de coloração de grafos como programas lineares inteiros. Observamos também que até o momento os algoritmos baseados em *Branch-and-Price* e *Branch-and-Bound* DSATUR são os mais promissores para resolução do problema de coloração de grafos, destacando-se o trabalho de Mehrotra e Trick (1995) no primeiro grupo de algoritmos e o trabalho de Brélaz (1979) no segundo grupo, ambos precursores dos trabalhos mais recentes conhecidos até o momento.

Observamos que apesar das abordagens exatas para o problema de coloração de grafos diferirem entre si, elas apresentam relações que nem sempre estão claras ao pesquisador. Por exemplo, a heurística DSATUR tanto pode encontrar uma clique maximal como pode ser adaptada em um algoritmo exato *Branch-and-Bound* que apresenta bons resultados para grafos aleatórios. Além disso, vimos também que apesar de algoritmos baseados na recorrência proposta por Zykov (1962) serem muito menos praticáveis e eficientes em relação as outras abordagens, é interessante o uso desta recorrência na etapa de ramificação de algoritmos *Branch-and-Price*.

Referências Bibliográficas

Andrade, C. E. d., Miyazawa, F. K. e Xavier, E. C. (2006). Um algoritmo exato para o Problema de Empacotamento Bidimensional em Faixas. Em *Anais do XXXVIII Simpósio Brasileiro de Pesquisa Operacional*, páginas 1701–1712.

Appel, K. e Haken, W. (1977). Every planar map is four colorable. part i: Discharging. *Illinois J. Math.*, 21(3):429–490.

Appel, K., Haken, W. e Koch, J. (1977). Every planar map is four colorable. part ii: Reducibility. *Illinois J. Math.*, 21(3):491–567.

Araújo Neto, A. S. e Gomes, M. J. N. (2014). Problema e algoritmos de coloração em grafos - exatos e heurísticos. *Revista de Sistemas e Computação*, 4(2):201–115.

Barnhart, C., Johnson, E. L., Nemhauser, G. L., Savelsbergh, M. W. P. e Vance, P. H. (1998). Branch-and-price: Column generation for solving huge integer programs. *Oper. Res.*, 46(3):316–329.

Beigel, R. e Eppstein, D. (2005). 3-coloring in time $O(1.3289^n)$. *Journal of Algorithms*, 54(2):168 – 204.

Berge, C. (1961). Färbung von graphen, deren sämtliche bzw. deren ungerade kreise starr sind. *Wiss. Z. Martin-Luther-Univ. Halle-Wittenberg Math.-Natur. Reihe*, 10(114):88.

Björklund, A., Husfeldt, T. e Koivisto, M. (2009). Set partitioning via inclusion-exclusion. *SIAM Journal on Computing*, 39(2):546–563.

Bodlaender, H. L. e Kratsch, D. (2006). An exact algorithm for graph coloring with polynomial memory. *UU-CS*, 2006.

Bondy, J. e Murty, U. (2008). *Graph Theory*. Springer Publishing Company, Incorporated.

Brélaz, D. (1979). New methods to color the vertices of a graph. *Commun. ACM*, 22(4):251–256.

Brown, J. R. (1972). Chromatic scheduling and the chromatic number problem. *Management Science*, 19(4-part-1):456–463.

Byskov, J. (2004). Enumerating maximal independent sets with applications to graph colouring. *Operations Research Letters*, 32(6):547–556.

Byskov, J. M. (2002). Chromatic number in time $O(2.4023^n)$ using maximal independent sets. *BRICS Report Series*, 9(45).

Chudnovsky, M., Lagoutte, A., Seymour, P. e Spirkl, S. (2017). Colouring perfect graphs with bounded clique number. *Journal of Combinatorial Theory, Series B*, 122:757 – 775.

Chudnovsky, M., Robertson, N., Seymour, P. e Thomas, R. (2006). The strong perfect graph theorem. *Annals of mathematics*, 164:51–229.

Cormen, T. H., Leiserson, C. E., Rivest, R. L. e Stein, C. (2009). *Introduction to Algorithms*. MIT Press, 3 edition.

Corneil, D. G. e Graham, B. (1973). An algorithm for determining the chromatic number of a graph. *SIAM Journal on Computing*, 2(4):311–318.

Cornuejols, G., Liu, X. e Vuskovic, K. (2003). A polynomial algorithm for recognizing perfect graphs. Em *44th Annual IEEE Symposium on Foundations of Computer Science, 2003. Proceedings.*, páginas 20–27.

Croitoru, C. (1979). On stables in graphs. *Proc. 3rd Coll. Operations Research*, páginas 55–60.

Dantzig, G. (1963). *Linear programming and extensions*. Rand Corporation Research Study. Princeton Univ. Press, Princeton, NJ.

Dong, F., Koh, K. e Teo, K. (2005). *Chromatic Polynomials and Chromaticity of Graphs*. World Scientific Pub.

Eppstein, D. (2001). Small maximal independent sets and faster exact graph coloring. Em *Proceedings of the 7th International Workshop on Algorithms and Data Structures*, WADS '01, páginas 462–470, London, UK, UK. Springer-Verlag.

Furini, F., Gabrel, V. e Ternier, I.-C. (2017). An improved dsatur-based branch-and-bound algorithm for the vertex coloring problem. *Networks*, 69(1):124–141.

Goldbarg, E. e Goldbarg, M. (2012). *GRAFOS: conceitos, algoritmos e aplicações*. Elsevier.

Grötschel, M., Lovász, L. e Schrijver, A. (1984). Polynomial algorithms for perfect graphs. Em Berge, C. e Chvátal, V., editores, *Topics on Perfect Graphs*, volume 88 de *North-Holland Mathematics Studies*, páginas 325 – 356. North-Holland.

Gualandi, S. e Malucelli, F. (2012). Exact solution of graph coloring problems via constraint programming and column generation. *INFORMS Journal on Computing*, 24(1):81–100.

Held, S., Cook, W. e Sewell, E. C. (2011). *Safe Lower Bounds for Graph Coloring*, páginas 261–273. Springer Berlin Heidelberg, Berlin, Heidelberg.

Karp, R. (1972). Reducibility among combinatorial problems. *Complexity of computer computations: proceedings*, páginas 85–103.

Kreher, D. L. e Stinson, D. R. (1999). *Combinatorial algorithms: generation, enumeration, and search*. CRC Press LTC.

Lawler, E. (1976). A note on the complexity of the chromatic number problem. *Information Processing Letters*, 5(3):66 – 67.

Lewis, R. (2016). *A Guide to Graph Colouring: Algorithms and Applications*. Springer, Berlin.

Lund, C. e Yannakakis, M. (1994). On the hardness of approximating minimization problems. *J. ACM*, 41(5):960–981.

Madsen, B. A., Byskov, J. M. e Skjernaa, B. (2002). On the number of maximal bipartite subgraphs of a graph. *BRICS Report Series*, 9(17).

Malaguti, E., Monaci, M. e Toth, P. (2011). An exact approach for the vertex coloring problem. *Discrete Optimization*, 8(2):174–190.

Marin, A. (2005). Algoritmos exatos para problema de coloração em grafos. Dissertação de Mestrado, Universidade Federal do Rio de Janeiro.

Matoušek, J. e Gärtner, B. (2007). *Understanding and using linear programming*. Springer.

McDiarmid, C. (1979). Determining the chromatic number of a graph. *SIAM Journal on Computing*, 8(1):1–14.

Mehrotra, A. e Trick, M. A. (1995). A column generation approach for graph coloring. *INFORMS Journal on Computing*, 8:344–354.

Méndez-Díaz, I. e Zabala, P. (2008). A cutting plane algorithm for graph coloring. *Discrete Applied Mathematics*, 156(2):159 – 179. Computational Methods for Graph Coloring and it's Generalizations.

Moon, J. e Moser, L. (1965). On cliques in graphs. *Israel Journal of Mathematics*, 3(1):23–28.

Papadimitriou, C. H. e Steiglitz, K. (1998). *Combinatorial Optimization: Algorithms and Complexity*. Dover Pub.

Rose, D. J., Tarjan, R. E. e Lueker, G. S. (1976). Algorithmic aspects of vertex elimination on graphs. *SIAM Journal on Computing*, 5(2):266–283.

San Segundo, P. (2012). A new dsatur-based algorithm for exact vertex coloring. *Comput. Oper. Res.*, 39(7):1724–1733.

Schindl, D. (2004). Some combinatorial optimization problems in graphs with applications in telecommunications and tomography.

Sewell, E. C. (1996). An improved algorithm for exact graph coloring. *DIMACS series in discrete mathematics and theoretical computer science*, 26:359–373.

Tsukiyama, S., Ide, M., Ariyoshi, H. e Shirakawa, I. (1977). A new algorithm for generating all the maximal independent sets. *SIAM Journal on Computing*, 6(3):505–517.

Turner, J. S. (1988). Almost all k-colorable graphs are easy to color. *Journal of Algorithms*, 9(1):63 – 82.

Wang, C. C. (1974). An algorithm for the chromatic number of a graph. *J. ACM*, 21(3):385–391.

Zykov, A. (1962). On some properties of linear complexes. *Algebraic Topology. v17*, páginas 418–419.