ALGORITMOS EXATOS PARA PROBLEMAS DE COLORAÇÃO EM GRAFOS

Alex Marin

DISSERTAÇÃO SUBMETIDA AO CORPO DOCENTE DA COORDENAÇÃO DOS PROGRAMAS DE PÓS-GRADUAÇÃO DE ENGENHARIA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Aprovada por:

Prof. Jayme Luiz Szwarcfiter, Ph.D.

Prof. Luerbio Faria, D.Sc.

Prof. Antonio Luz Furtado, Ph.D.

Prof. André Luiz Pires Guedes, D.Sc.

Profª. Celina Miraglia Herrera de Figueiredo, D.Sc.

RIO DE JANEIRO, RJ - BRASIL

DEZEMBRO DE 2005

MARIN, ALEX

Algoritmos Exatos para Problemas de Coloração em Grafos [Rio de Janeiro] 2005

XI, 124 p. 29,7 cm (COPPE/UFRJ, M.Sc., Engenharia de Sistemas e Computação, 2005)

Dissertação – Universidade Federal do Rio de Janeiro, COPPE

1 - Coloração de grafos

2 - Algoritmos exatos

3 - Número cromático

I. COPPE/UFRJ II. Título (série)

Aos meus amados pais, Noimar e Maria.

Agradecimentos

A meus pais, Noimar e Maria, por terem sido meus maiores incentivadores. Obrigado pelo amor, carinho, compreensão e esforço para tornar este sonho em realidade. Vocês são tudo na minha vida.

A minha família, em especial minhas irmãs Alessandra e Aline, minhas queridas avó Almery e tia-madrinha Neuza, sempre dispostas e prestativas.

Aos Profs. Jayme Szwarcfiter e Luerbio Faria, meus orientadores, por terem acreditado em mim, por suas valiosas contribuições, pela dedicação e incentivos permanentes.

Aos Profs. Antonio Furtado, André Guedes e Celina Figueiredo, por aceitarem fazer parte desta banca, agregando valor inestimável a este trabalho.

Ao meu grande amigo, Natanael Maia, pelo seu companheirismo, pela força nos momentos difíceis e de solidão, por aturar minhas brincadeiras e gozações. Espero que nossa amizade cresça cada vez mais.

A Thayse, uma pessoa iluminada que surgiu em minha vida e que nem mesmo a distância geográfica que nos separa é capaz de diminuir o senti-

mento de carinho e respeito que temos um pelo outro. Te adoro muitão.

A família Martins, em especial ao seu Boris, dona Ionice e a Marlize, pela acolhida, por me tratarem como sendo um membro da família. Não tenho palavras para expressar minha gratidão. Jamais esquecerei de vocês.

Aos colegas, amigos e demais professores da linha de Algoritmos e Combinatória pela força que sempre me deram.

Resumo da Dissertação apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc.)

ALGORITMOS EXATOS PARA PROBLEMAS DE COLORAÇÃO EM GRAFOS

Alex Marin

Dezembro/2005

Orientadores: Jayme Luiz Szwarcfiter
Luerbio Faria

Programa: Engenharia de Sistemas e Computação

Colorir um grafo significa pintar cada um dos seus vértices com uma cor, de forma tal que, se dois vértices quaisquer são adjacentes, então eles recebem cores distintas. Neste trabalho, realizamos um estudo sob forma de levantamento entre os diversos métodos que solucionam, de maneira exata, o problema de encontrar o número mínimo de cores, chamado número cromático, necessárias para colorir um grafo qualquer. Além deste, analisamos os algoritmos exatos para outros dois problemas de coloração em grafos: a 3-coloração e a 4-coloração. Propomos ainda, um novo algoritmo para verificar se um grafo pode ser colorido com três cores.

Abstract of Dissertation presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

EXACT ALGORITHMS FOR GRAPH COLORING PROBLEMS

Alex Marin

December/2005

Advisors: Jayme Luiz Szwarcfiter
Luerbio Faria

Department: Systems Engineering and Computer Science

To color a graph means painting each of its vertices with some color, in such a way that whenever two vertices are adjacent they receive different colors. In the present work, we survey some of the different exact methods described in the literature, for coloring a graph using the minimum possible number of colors - the chromatic number of the graph. In addition, we also study the existing exact methods for the problems of 3-coloring and 4-coloring. Finally, we propose a new algorithm for solving the 3-coloring problem.

Sumário

Lista de Figuras

Capítulo 1

Introdução

Imagine uma situação como esta: em uma fábrica (de substâncias químicas, por exemplo), devem-se armazenar vários produtos. E, por questões de segurança, alguns destes não podem ser estocados juntamente com outros. O problema consiste em alocar os diversos produtos em um menor número de compartimentos (salas), de tal forma que não ocorram conflitos entre as substâncias.

Podemos modelar este problema utilizando a estrutura de um grafo, onde os vértices irão representar as substâncias e as arestas corresponderão aos conflitos entre as mesmas. Assim sendo, o problema agora é colorir os vértices do grafo, com o menor número de cores possível, tal que vértices adjacentes recebam cores distintas.

KARP [27] mostra que encontrar uma atribuição ótima de k cores ($k \geq 3$) aos vértices de um grafo é um problema NP-completo. Admitindo que P $\neq$ NP, podemos afirmar que não existirá um algoritmo de tempo polinomial para a solução exata desse problema. Entretanto, essa dificuldade inerente ao problema não isenta-nos da necessidade de resolvê-lo o mais eficientemente

possível, na verdade, o fato deste problema ser difícil faz com que a busca por soluções eficientes seja de suma importância.

Neste trabalho, faremos uma inspeção entre os algoritmos exatos existentes na literatura para problemas de coloração em grafos. Mais precisamente, estudaremos os procedimentos que determinam o valor mínimo ótimo de cores que um grafo pode receber, valor esse denominado de número cromático. Analisaremos também os métodos que verificam se um grafo pode ser colorido com três (problema da 3-coloração) ou quatro (4-coloração) cores.

Além dessa análise, propomos um novo método algorítmico que determina se um grafo é ou não 3-colorível. Para elaborarmos este procedimento, utilizamos os conceitos propostos por ZYKOV [46] e que, até então, eram usados somente na determinação do número cromático.

A seguir, descreveremos como a dissertação está organizada, ressaltando a contribuição de cada capítulo.

No Capítulo 1, serão apresentados alguns conceitos inerentes aos objetos de estudo desta dissertação. Primeiramente, faremos uma breve descrição do modo como o problema da coloração em grafos se originou. Em um segundo momento, apresentamos algumas definições relevantes em grafos, as quais serão utilizadas ao longo do trabalho. A seguir, falaremos sobre a importância da complexidade computacional e definiremos a notação que medirá a eficiência dos algoritmos. Finalizando o capítulo, apresentamos as definições de algumas técnicas usadas em algoritmos exatos, as quais são a programação dinâmica e a *branch-and-reduce*.

No Capítulo 2, discorreremos sobre conjuntos independentes de um grafo G qualquer, mostrando que para obter o número cromático de G não é necessário que consideremos todos os conjuntos independentes, mas somente os maximais. Ainda, apresentaremos limites superiores para o número de

conjuntos independentes maximais e alguns algoritmos para a geração destes conjuntos.

No capítulo seguinte, iniciaremos o estudo dos algoritmos que determinam o número cromático de um grafo G. Nele, apresentaremos os métodos exatos que utilizam, de alguma maneira, o conceito de conjuntos independentes maximais para obter uma coloração ótima de G.

No Capítulo 4, encerramos a parte sobre algoritmos para o cálculo do número cromático, com o método proposto por ZYKOV [46], o qual, conforme dito anteriormente, servirá de base para a elaboração de um novo método que verifica a 3-colorabilidade de um grafo.

Algoritmos exatos para o problema da 3-coloração de vértices em um grafo são o assunto do Capítulo 5. Mostraremos os métodos propostos por LAWLER [32] e por BEIGEL e EPPSTEIN [2, 3]; fechamos o capítulo apresentando um novo algoritmo, o qual utiliza técnicas conhecidas na literatura, mas que ainda não tinham sido empregadas para a solução deste problema.

No Capítulo 6, abordaremos o problema da 4-coloração, a qual, será verificada através de três diferentes métodos.

No Capítulo 7, serão expostas as conclusões desta dissertação, bem como uma sugestão de alguns problemas e trabalhos futuros.

1.1 Origem histórica do problema de coloração

A história do problema de coloração começou em 1852, quando Francis Guthrie tentava colorir os vários distritos do mapa da Inglaterra de modo tal que, dois distritos vizinhos não tivessem a mesma cor. Depois de ter refletido sobre o problema, conjecturou que qualquer mapa poderia ser colo-

rido com apenas quatro cores. Francis Guthrie, que foi advogado, botânico e matemático, tinha um irmão mais novo, Frederick Guthrie, o qual era aluno de Augustus De Morgan. Durante uma aula, Frederick apresentou a conjectura do seu irmão mais velho ao professor De Morgan. Este ficou muito entusiasmado e, no mesmo dia, escreveu uma carta a Sir William Rowan Hamilton na qual explicava o problema. Esta carta foi conservada e encontra-se hoje nos arquivos do Trinity College em Dublin.

Contrastando com a animação de De Morgan, Hamilton não achou o problema interessante. Respondeu somente quatro dias mais tarde, dizendo que tão cedo não tencionava debruçar-se sobre a questão.

Nos tempos que se seguiram, foi, sobretudo, através de De Morgan que a comunidade científica tomou conhecimento da conjectura de Guthrie, ou Conjectura das Quatro Cores como é mais conhecida. De Morgan escreveu algumas cartas para outros matemáticos conhecidos, o problema foi discutido e teve alguns desenvolvimentos.

Somente em 1976, ou seja, 124 anos após o estabelecimento da Conjectura das Quatro Cores, que dois matemáticos, Kenneth Appel e Wolfgang Haken [1], apresentaram uma demonstração do Teorema das Quatro Cores, a qual incluía mais de mil horas de processamento do computador mais veloz da época.

1.2 Grafos – conceitos básicos

Muitas situações reais podem ser descritas através de um diagrama formado por um conjunto de pontos, juntamente com linhas unindo certos pares destes pontos. Por exemplo, os pontos podem representar cidades e as linhas representam as estradas que ligam as cidades uma a uma.

Um *grafo* G é formado por um par de conjuntos $G = (V, E)$, onde V é um conjunto finito de elementos denominados *vértices*, juntamente com um conjunto E de pares não ordenados de elementos de V denominados *arestas*.

Exemplo:

grafo G

$V = \{1, 2, 3, 4, 5, 6, 7\}$

$E = \{(1,2), (1,3), (1,4), (2,3), (2,6), (3,5), (3,6), (4,5), (5,7)\}$

Uma aresta e será denotada pelos vértices que a formam: $e = (v, w)$. Nesse caso, os vértices v e w são ditos *adjacentes* ou *vizinhos*.

Igualmente, se duas arestas são formadas por pares de vértices com um vértice em comum elas também são chamadas *adjacentes*. Exemplo: (1,2) e (1,4).

Ao longo deste trabalho estaremos interessados somente em grafos *simples*, ou seja, grafos que não possuem *laços* (aresta que liga um vértice v a ele mesmo) e nem *arestas paralelas* (mais de uma aresta ligando os vértices v e w).

Sempre que necessário, utilizaremos as variáveis $n = |V|$ para indicar o número de vértices e $m = |E|$ para o número de arestas do grafo G.

O *grau* $d(v)$ de um vértice v em G é o número de arestas de G incidentes em v. Denotaremos, respectivamente, por $\delta(G)$ e $\Delta(G)$ o *grau mínimo* e *máximo* dos vértices de G.

A *vizinhança* de um vértice v é o conjunto $N(v) = \{w \in V | (v, w) \in E\}$ e, por conveniência notacional, seja $N[v] = \{v\} \cup N(v)$.

O *grafo complemento* $\bar{G}$ de um grafo simples G é um grafo simples, com conjunto de vértices V e, dois vértices são adjacentes em $\bar{G}$ se e somente se, eles não são adjacentes em G.

Um grafo H é um *subgrafo* de G se $V(H) \subseteq V(G)$ e $E(H) \subseteq E(G)$ e dizemos que H é um *subgrafo próprio* de G se $H \neq G$. Se o subgrafo H satisfizer a seguinte propriedade: $v, w \in V(H)$ e $(v, w) \in E(G) \Rightarrow (v, w) \in E(H)$, então H é chamado de *subgrafo induzido* de G, cuja notação é $G[H]$.

Grafos são assim nomeados, possivelmente porque podem ser representados graficamente, e sua representação gráfica nos ajuda a entender muitas das suas propriedades. Cada vértice é indicado por um ponto e cada aresta é representada por uma linha que une dois pontos (vértices). A *representação gráfica* para o grafo G do exemplo acima pode ser vista na Figura 1.1.

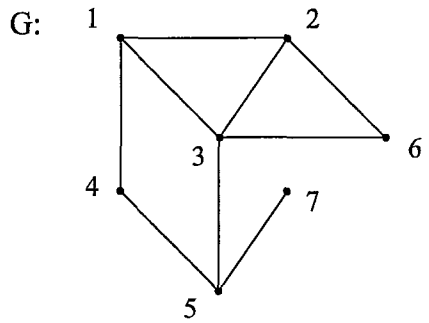

Figura 1.1: Representação gráfica do grafo G.

Uma representação geométrica (gráfica) é *plana* se ela puder ser desenhada em um plano sem cruzamento de linhas. Um grafo é *planar* quando este admitir uma representação plana.

Um grafo G é *completo* quando quaisquer dois vértices em G forem adjacentes.

O grafo $G = (V, E)$ é *vazio*, se e somente se o conjunto de arestas E for um conjunto vazio.

Um grafo *bipartido* é aquele no qual o seu conjunto de vértices V pode ser particionado em dois subconjuntos X e Y de tal modo que toda aresta de G une um vértice de X a um de Y.

Um *passeio* é uma seqüência de vértices onde cada qual é adjacente a seus vizinhos na seqüência. Um exemplo de passeio para o grafo da Figura 1.1 pode ser representado por: 4,1,3,1,2,1,3,2.

Um *caminho* é um passeio sem repetição de vértices. Exemplo: 4,1,3,2.

Um *ciclo* é um passeio fechado (o último vértice é igual ao primeiro) $v_1, \ldots, v_k, v_{k+1}$ onde $v_1, \ldots, v_k$ é um caminho e $v_1 = v_{k+1}$ para $k \geq 3$. Exemplo: 1,3,2,1.

Sejam v e w dois vértices de G, tal que existe um caminho entre eles em G. Então v e w pertencem a uma mesma *componente conexa.* Um grafo no qual quaisquer dois vértices pertencem a mesma componente conexa é chamado *conexo*, caso contrário, *desconexo.*

Uma *árvore* é um grafo conexo e sem ciclos. Uma *floresta* é uma coleção de árvores.

Chamamos de *conjunto independente* de vértices todo subconjunto $S \subseteq V(G)$ formado por vértices dois a dois não adjacentes em G.

Seja um grafo $G = (V, E)$. Uma *k-coloração própria* de G é uma atribuição de k cores $\{1, 2, \ldots, k\}$ aos vértices de G tal que vértices adjacentes não recebam a mesma cor.

Uma *classe de cor* de G é um conjunto de vértices com a mesma cor, em uma coloração de G. Assim, uma classe de cor é um *subconjunto independente* de G.

Uma k-coloração particiona o conjunto de vértices V de um grafo G em k conjuntos independentes (possivelmente vazios) $V_1, V_2, \ldots, V_k$ onde $V_i = \{v \in V | cor(v) = i\}$ para $1 \leq i \leq k$.

O *número cromático* de G, denotado por $\chi(G)$ é o menor k para o qual G admite uma k-coloração própria. Observamos que o número cromático de um grafo completo é igual a n, uma vez que cada vértice é adjacente com todos os demais.

Dizemos que uma k-coloração de G é uma *coloração ótima* se $k = \chi(G)$. Ainda, se $k = \chi(G)$ então G é dito um *grafo k-cromático.*

Um grafo G é *crítico* se para todo subgrafo próprio H de G temos que $\chi(H) < \chi(G)$. Dizemos que G é *k-crítico* se ele for crítico e $\chi(G) = k$.

O *problema de coloração* em grafos consiste em determinar o número cromático de um grafo G qualquer.

1.3 Complexidade computacional

Complexidade computacional é a área da ciência da computação que contempla as razões pelas quais alguns problemas são difíceis (custosos) de serem resolvidos pelos computadores. Há 40 anos atrás, esta área praticamente não existia, atualmente, ela figura como uma das maiores áreas da atividade de pesquisa na ciência da computação teórica.

Para obter uma solução de um problema são usados métodos, chamados *algoritmos*, que consistem em um conjunto pré-determinado e bem definido de regras e processos, com um número finito de passos .

A complexidade computacional de um algoritmo diz respeito aos recursos computacionais – espaço de memória e tempo de processamento – requeridos para solucionar um problema. Geralmente existe mais de um algoritmo para resolver um mesmo problema. A análise de complexidade computacional é, portanto, fundamental no processo de definição/escolha de algoritmos.

A análise real dos requisitos de um algoritmo é complexa e nem sempre tem significado prático (pode-se mudar o computador, o sistema operacional, a linguagem, o compilador, etc.). Prefere-se, ao invés disso ter uma medida aproximada, comparável e matematicamente tratável. Uma característica importante é que a eficiência de um algoritmo é sempre em relação ao seu tempo de execução. Para chegar a esta medida, utiliza-se a contagem das operações elementares básicas (operadores aritméticos, laços de condição, comparações, etc.) onde cada uma dessas operações requer uma unidade de tempo. Isso é feito de modo a tornar a análise independente da velocidade de determinado computador que porventura seja utilizado para executar o algoritmo. Portanto, calcula-se passos ao invés de ciclos de máquina e, para chegarmos a um determinado valor, utiliza-se uma notação.

A mais conhecida e também mais utilizada é a notação-O (notação *big-O*). Com esta notação, um algoritmo cujo número de passos em relação ao tamanho dos dados de entrada é limitado superiormente por uma constante positiva vezes uma função $f(n)$, é dito ser $O(f(n))$.

Uma vez escolhida uma medida de desempenho de um algoritmo, o que resta estabelecer é uma relação entre essa medida e a questão de determinar se um algoritmo é bom ou não. Para respondermos a esta questão, devemos analisar a natureza do problema, ou seja, o quão difícil (custoso) é resolvê-lo.

Existem duas principais classes de problemas de decisão, a classe **P** que engloba todos os problemas que podem ser *resolvidos* em uma máquina seqüêncial determinística em um tempo que é polinomial em relação ao tamanho da entrada, ou seja, $O(p(n))$ onde p é um polinômio e n o tamanho da entrada; e a classe **NP** a qual é formada pelos problemas onde as possíveis soluções podem ser *certificadas* em tempo polinomial fornecendo a informação correta, ou equivalentemente, a solução pode ser encontrada

em um tempo polinomial em uma máquina não-determinística. Um exemplo para problemas desta classe são os limitados por funções não polinomiais do tipo $O(2^n)$.

Resumindo, os problemas da classe P, também conhecidos como *tratáveis*, são aqueles que possuem algoritmo de resolução cuja complexidade é polinomial. Para a resolução de todos os problemas da classe NP, não se conhece um algoritmo de tempo polinomial.

Dentro da classe NP existe uma sub-classe chamada *NP-completa.* Os problemas NP-completos são os mais difíceis problemas em NP. Um problema pertence a esta sub-classe se ele pertence a NP e todo outro problema em NP é polinomialmente redutível a ele. Logo, se encontrarmos uma maneira de resolver um problema NP-completo eficientemente, poderemos usar este algoritmo para resolver todos os problemas em NP eficientemente.

O desenvolvimento da teoria da complexidade computacional têm levado a novos estudos sobre a dificuldade inerente de resolver um número cada vez maior de problemas. E tem contribuído, fornecendo métodos rigorosos de avaliação de algoritmos e classificação de problemas.

No presente trabalho estaremos interessados em algoritmos que resolvam até a otimalidade problemas da classe NP, especificamente o problema de coloração de vértices em grafos, o qual é NP-completo [27].

Desta forma, todos os algoritmos abordados nesta dissertação possuem, no pior caso, complexidade de tempo exponencial. Assim sendo, fatores polinomiais em relação ao tamanho da entrada podem ser ignorados.

Utilizaremos uma notação big-O modificada, chamada de $O^*(c^n)$, a qual irá representar um conjunto de funções contidas em um fator polinomial de uma função em $O(c^n)$. Em outras palavras, escreveremos $O^*(c^n)$ para uma

complexidade da forma $O(c^n \cdot poli(n))$. Esta modificação se justifica pois c^n tem um crescimento exponencial. Ainda, se efetuarmos um arredondamento (para cima) em c, pode-se omitir a estrela, pois tempos polinomiais em funções exponenciais crescem mais lentamente do que qualquer função exponencial cujo expoente seja suficientemente grande.

Teorema 1.1. $O((c+\epsilon)^n) = O^*(c^n)$, $c > 1$, $n \in \mathbb{N}$ *e* $\epsilon > 0$.

Demonstração. É suficiente provar que existe $\alpha \in \mathbb{R}^+$ tal que $\alpha(c+\epsilon)^n > c^n \cdot n^k$, $k \in \mathbb{N}^*$.

Observe que $log_c(c+\epsilon)^n = n \cdot log_c(c+\epsilon)$ que é menor ou igual a $n \cdot (1+y)$, $y > 0$, pois $c + \epsilon > c$. Note também que $log_c(c^n \cdot n^k) = log_c c^n + log_c n^k = n + k \cdot log_c n$. Portanto, podemos escolher $\alpha = 1$, pois existe n_0 de n tal que se $n > n_0$ acontece que: $n > k/y \cdot log_c n = log_c n^{k/y}$. □

1.4 Técnicas usadas em algoritmos exatos

Nesta seção, apresentaremos duas das principais técnicas para construção de algoritmos exatos eficientes. São elas: a programação dinâmica e a *branch-and-reduce* (também denominada análise de casos ou poda na árvore de busca). Para maiores informações sobre estas ou outras técnicas, sugerimos ao leitor o *survey* de WOEGINGER [45].

1.4.1 Programação dinâmica

A programação dinâmica talvez seja a técnica mais utilizada para a construção de algoritmos exatos. Um dos fatores para essa grande utilização é a facilidade na sua compreensão e aplicação. Seu uso pode reduzir maciçamente o tempo de execução. Porém, ela pode armazenar uma quantidade exponencial de informação.

A idéia desta técnica consiste em armazenar resultados intermediários, os quais serão utilizados futuramente, mais de uma vez. Assim, evita-se que tais resultados sejam recalculados.

Neste trabalho, a programação dinâmica pode ser encontrada em algoritmos para conjuntos independentes maximais e para o número cromático.

1.4.2 Branch-and-reduce

Esta técnica é assim nomeada pois, os algoritmos que a utilizam, realizam duas operações: *branch* (do inglês: ramificar, dividir) e *reduce* (reduzir, diminuir).

A operação *branch* permite que o algoritmo resolva o problema fazendo chamadas recursivas em instâncias menores, ou seja, subpartes do problema original. Esta operação é também denominada *análise de casos*, pois, geralmente, os algoritmos fazem uma diferenciação das instâncias em casos e, para cada caso, diferentes operações recursivas podem ser efetuadas.

A operação *reduce* é quase auto-explicativa, significa que o algoritmo consegue simplificar os dados de entrada. Por exemplo, no caso onde alguma parte da entrada é redundante ou no caso onde alguma parte da solução pode ser obtida sem a necessidade da realização da operação *branch*. Assim sendo, essa operação pode resolver algumas instâncias mais fáceis do problema em tempo polinomial. A operação *reduce* pode ser considerada também, um caso especial da operação *branch*, na qual somente uma chamada recursiva existe.

Para fazermos a análise de complexidade de um algoritmo *branch-and-reduce* podemos proceder da seguinte forma: sabemos que, na operação *branch*, para cada caso, o algoritmo efetua várias ramificações (chamadas recursivas). Sem perda de generalidade, assumimos que o parâmetro de

complexidade é em função da entrada n e que, para um determinado caso, k ramificações foram feitas, dividindo n por $t_1, t_2, \ldots, t_k$. Assim, o tempo de execução pode ser representado por uma recursão da forma $T(n) = T(n - t_1) + T(n - t_2) + \cdots + T(n - t_k)$ já ignorando os fatores polinomiais. Denominamos $t = (t_1, t_2, \ldots, t_k)$ como sendo o *vetor de ramificação* (*branching vector*) para este caso. A solução da recursão é $T(n) = \alpha_t^n$, onde α_t é a raiz positiva de $1 - 1/x^{t_1} - 1/x^{t_2} - \cdots - 1/x^{t_k}$.

A complexidade de tempo de um algoritmo *branch-and-reduce* é então calculada em função da análise de todos os casos possíveis e, o pior deles irá dominar o tempo de execução. Uma vez que a operação *reduce* pode ser implementada em tempo polinomial, temos que a complexidade de pior caso representará algum caso da operação *branch*. Assim, ela é da ordem de $O^*(\alpha^n)$, onde α é o maior entre todos os α_t. Provas destes resultados e maiores informações podem ser encontradas em [30].

Capítulo 2

Colorações por Conjuntos Independentes

O assunto abordado neste capítulo trata de conjuntos independentes de um grafo. Mostraremos como utilizar este conceito para encontrar uma coloração ótima para o grafo G. Na seção 2.1, apresentaremos os limites superiores para o número de conjuntos independentes maximais e, alguns algoritmos para a obtenção de tais conjuntos.

Ao longo do tempo, descobriram-se várias maneiras diferentes de se obter o número cromático de um grafo G qualquer. O objetivo, na maioria delas, deriva-se da própria definição de número cromático e de classes de cor. Consiste em se encontrar o menor número k de conjuntos independentes $V_1, V_2, \ldots, V_k$ de G, de forma que sua união contenha todos os vértices de V, ou seja, $\cup_{i=1,k}\ V_i = V$. Nesse caso, dizemos que $V_1, V_2, \ldots, V_k$ constituem uma *cobertura* de G por conjuntos, ou simplesmente que *cobrem* G. O menor valor de k, com o qual isso acontece é o número cromático de G.

Sendo assim, o seguinte procedimento básico pode ser utilizado para encontrar uma coloração ótima:

Procedimento:

(i) Determina-se a coleção de todos os conjuntos independentes de G, que serão aqui denotados por CI;

(ii) Identifica-se C uma subcoleção mínima de CI's que cubra G. Cada C_i de C assim obtido é uma classe de cor e o número cromático de G é o número de elementos de C.

Um conjunto independente $S \subseteq V$ de um grafo $G = (V, E)$ é *maximal* quando cada vértice em $G[V \backslash S]$ está ligado por uma aresta a algum vértice em S, ou seja, um conjunto independente é maximal quando ele não é subconjunto próprio de outro conjunto independente. Portanto temos que, dado um grafo G, qualquer conjunto independente de G está contido em um conjunto independente maximal de G, que denotaremos por CIM. Adotaremos $S(G)$ como o conjunto de todos os CIM's em G e $S^{=L}(G)$ o subconjunto de CIM's cujos tamanhos sejam iguais a L. Similarmente, temos $S^{\leq L}(G)$ e $S^{\geq L}(G)$. Vamos mostrar que é suficiente considerar CIM's no lugar de CI's no procedimento prévio.

O seguinte teorema, apresentado por HAMMER e RUDEANU [24], garante que para a obtenção do número cromático de G é suficiente considerar apenas os conjuntos independentes maximais.

Teorema 2.1 (Hammer e Rudeanu, 1968). *O número cromático χ de um grafo G qualquer é igual ao menor número de conjuntos independentes maximais que cobrem G.*

Demonstração. Considere que $\chi(G) = k$, logo deve haver um conjunto $C = \{C_1, C_2, \ldots, C_k\}$ de classes de cor, formado pelas k cores possíveis de G. Cada classe de cor C_j, $1 \leq j \leq k$, corresponde a um conjunto independente, o qual sempre está contido em um conjunto independente maximal. Sem perda de generalidade, teremos então um conjunto $S = \{S_1, S_2, \ldots, S_k\}$ de conjuntos independentes maximais cobrindo os vértices de G. E, o conjunto C de classes de cor pode ser obtido a partir de S através das seguintes relações:

$$C_1 = S_1$$

$$C_2 = S_2 - S_1$$

$$\vdots$$

$$C_k = S_k - \bigcup_{j=1}^{k-1} S_j$$

□

Corolário 2.2. *Se G for um grafo k-cromático, então existirá, pelo menos, uma coloração ótima de G em que uma das classes de cor (designada por C_1 na prova do Teorema 2.1) é um CIM de G.*

Chamamos de *coloração independente* de G uma k-coloração de G tal que a primeira classe de cor, C_1, é um CIM de G, a segunda classe de cor, C_2, é um CIM de $G_2 = G[V \backslash C_1]$, e a i-ésima classe de cor, C_i, é um CIM de $G_i = G[V \backslash \{C_1 \cup C_2 \cup \ldots \cup C_{i-1}\}]$, para $1 \leq i \leq k$.

Teorema 2.3 (Christofides, 1971). *Se $\chi(G) = k$, então existe uma k-coloração independente de G.*

Demonstração. Seja G um grafo onde $k = \chi(G)$, logo, G é um grafo k-cromático, então existe (Teorema 2.1), pelo menos uma coloração ótima de

G em que uma classe de cor, C_1, é um CIM de G. Façamos $G_2 = [V \backslash C_1]$, por construção, G_2 é um grafo $(k-1)$-cromático, então existe uma $(k-1)$-coloração ótima em que uma das classes de cor, C_2, é um CIM de G_2. Aplicando o mesmo procedimento para $G_i = [V \backslash \{C_1 \cup C_2 \cup \ldots \cup C_{i-1}\}]$, para $1 \leq i \leq k$, obteremos uma k-coloração independente de G cujas classes de cor estarão representadas por $C_1, C_2, \ldots, C_k$. □

Logo, o número cromático de G pode ser obtido através da construção das colorações independentes de G. Para efetuarmos essa construção, utilizam-se operações que envolvem conjuntos independentes maximais.

Evidentemente, qualquer algoritmo que utilize esta idéia, deverá ser capaz de gerar/determinar todos os CIM's de um grafo (subgrafo) em um menor tempo possível. Na próxima seção serão apresentados alguns métodos que realizam tal tarefa.

2.1 Obtenção dos Conjuntos Independentes Maximais

Enumerar todos os conjuntos independentes maximais de um grafo é um problema bastante conhecido e com muitas aplicações tanto no âmbito prático, quanto no teórico. Para alguns exemplos, consulte [33]. Nesta seção buscaremos estabelecer limites superiores para este problema e apresentar alguns algoritmos para a geração de todos os conjuntos independentes maximais.

Uma *clique* de G é um subconjunto $S \subseteq V(G)$ formado por vértices dois a dois adjacentes, ou seja, é qualquer subgrafo completo de G.

A relação que podemos fazer entre conjuntos independentes e cliques é: $S \subseteq V(G)$ é uma clique em G se e somente se S for um conjunto independente em $\bar{G}$ (grafo complemento de G).

O problema de encontrar todos os conjuntos independentes maximais de um grafo é, portanto, equivalente ao de encontrar todas as cliques maximais do grafo, pois, como visto anteriormente, cada conjunto independente maximal de um grafo G corresponde a uma clique maximal do grafo complementar $\bar{G}$ de G.

2.1.1 Limites Superiores

Devido à essa complementaridade entre cliques e conjuntos independentes, podemos afirmar que: o número de CIM's é um limite superior, enquanto que a cardinalidade da maior clique é um limite inferior para o número cromático. Entretanto, existem casos onde esses limites encontram-se muito distantes do valor ótimo. Naturalmente, o limite superior acima só possui algum interesse para o caso de grafos cuja quantidade de CIM's é menor do que n, visto que n é um limite superior para o número cromático.

Inicialmente, forneceremos o limite exato para o número de conjuntos independentes maximais de um grafo. Em seguida, apresentaremos limites superiores para este mesmo número, porém, iremos restringir o tamanho L de cada conjunto independente maximal.

Em 1965, MOON e MOSER [35] apresentaram um artigo que fornece o número máximo de cliques em um grafo. E, devida à equivalência entre cliques e conjuntos independentes, podemos, estender o resultado de Moon e Moser para conjuntos independentes.

Lema 2.4. *Se G é formado pela união disjunta de subgrafos completos de tamanho x, então o número máximo de CIM's é atingido quando $x = 3$.*

Demonstração. Para encontrar o número máximo de CIM's de G, devemos encontrar o valor máximo da seguinte função: $f(x) = x^{n/x}$

$$\begin{aligned} f(x) &= x^{n/x} = e^{lnx^{n/x}} \\ f'(x) &= e^{lnx^{n/x}} \cdot (n/x \cdot lnx)' \\ &= x^{n/x}\left[\frac{-1}{x^2} \cdot n \cdot lnx + \frac{n}{x} \cdot \frac{1}{x}\right] \\ &= \frac{n \cdot x^{n/x}}{x^2}\left[-1 \cdot lnx + 1\right], \end{aligned}$$

$lnx = 1$ quando $x = e$. Logo, o valor máximo será quando $x = e$, como $e \approx 2.38$ devemos testar para $x = 2$ e $x = 3$, substituindo na função temos: $f(2) = 2^{n/2} \approx 1.41^n$ e $f(3) = 3^{n/3} \approx 1.44^n$. Desta forma, o número máximo de CIM's é obtido quando $x = 3$. □

Assim, o valor exato para o número de conjuntos independentes maximais em um grafo é dado pelo seguinte teorema:

Teorema 2.5 (Moon e Moser, 1965). *O número máximo de conjuntos independentes maximais em um grafo qualquer é:*

$$\begin{cases} 3^{n/3} & \text{se } n \equiv 0 \text{ (mod 3)}, \\ 4 \cdot 3^{(n-4)/3} & \text{se } n \equiv 1 \text{ (mod 3)}, \\ 2 \cdot 3^{(n-2)/3} & \text{se } n \equiv 2 \text{ (mod 3)}. \end{cases} \tag{2.1}$$

Estes limites podem ser obtidos através dos grafos extremais (Figura 2.1) formados pela união disjunta de:

- $n/3$ K_3;

- um K_4 ou dois K_2, com $(n-4)/3$ K_3;
- um K_2 com $(n-2)/3$ K_3.

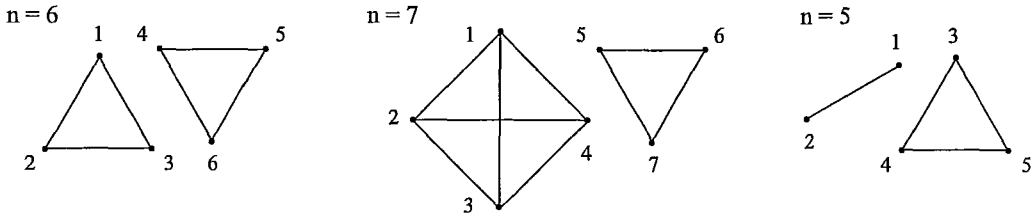

Figura 2.1: Grafos com número máximo de conjuntos independentes maximais.

Se compararmos os três valores propostos, facilmente verificamos que:

$$4 \cdot 3^{(n-4)/3} < 2 \cdot 3^{(n-2)/3} < 3^{n/3}$$

portanto, um grafo pode ter, no máximo, $3^{n/3}$ conjuntos independentes maximais. Esse limite é bastante conhecido na literatura e pode ser referenciado como *limite de Moon e Moser*.

A seguir, apresentaremos os limites superiores para o número de conjuntos independentes maximais em função do tamanho L que cada conjunto pode ter. Esses limites foram apresentados por BYSKOV [10] e, assim como o de Moon e Moser, são obtidos através de argumentos combinatoriais. As provas dos teoremas podem ser obtidas em [10].

Teorema 2.6 (Byskov, 2004). *O número máximo de conjuntos independentes maximais de tamanho exatamente L em qualquer grafo é*

$$\lfloor n/L \rfloor^{(\lfloor n/L \rfloor + 1)L - n} (\lfloor n/L \rfloor + 1)^{n - \lfloor n/L \rfloor L} \tag{2.2}$$

O valor mostrado em 2.2 foi proposto inicialmente por CROITORU [16] para grafos cujos conjuntos independentes maximais têm tamanho no máximo L.

Teorema 2.7 (Byskov, 2004). *O número máximo de conjuntos independentes maximais de tamanho no máximo L em qualquer grafo é*

(a) Para $L \leq n/3$:

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

(b) Para $L \geq n/3$:

$$\begin{cases} 3^{n/3} & \text{se } n \equiv 0 \text{ (mod 3)}, \\ 4 \cdot 3^{(n-4)/3} & \text{se } n \equiv 1 \text{ (mod 3)}, \\ 2 \cdot 3^{(n-2)/3} & \text{se } n \equiv 2 \text{ (mod 3)}. \end{cases}$$

Teorema 2.8 (Byskov, 2004). *O número máximo de conjuntos independentes maximais de tamanho pelo menos L em qualquer grafo é*

(a) Para $L \leq n/3$:

$$\begin{cases} 3^{n/3} & \text{se } n \equiv 0 \text{ (mod 3)}, \\ 4 \cdot 3^{(n-4)/3} & \text{se } n \equiv 1 \text{ (mod 3)}, \\ 2 \cdot 3^{(n-2)/3} & \text{se } n \equiv 2 \text{ (mod 3)}. \end{cases}$$

(b) Para $L \geq n/3$:

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

Deste modo, o autor mostra que o limite de Croitoru é um limite superior para o número de conjuntos independentes maximais de tamanho L para qualquer valor de L. Ainda, para conjuntos independentes maximais de tamanho *no máximo* L o limite de Croitoru é um limite superior justo para $L \leq n/3$ e, para $L \geq n/3$ o limite de Moon e Moser é um limite superior justo. Para conjuntos independentes maximais de tamanho *pelo menos* L, o limite de Croitoru é um limite superior justo para $L \geq n/3$ e o limite de Moon e Moser é justo para $L \leq n/3$.

2.1.2 Algoritmos para geração dos CIM's

Vimos no início deste capítulo que a maioria dos métodos para a obtenção do número cromático em um grafo utilizam o conceito de conjuntos independentes. Para atingir um melhor desempenho, estes métodos devem, portanto, obter tais conjuntos em um menor tempo possível. Os algoritmos que geram/enumeram todos os conjuntos independentes maximais de um grafo G qualquer são o assunto abordado nessa seção.

2.1.2.1 O procedimento de Tsukiyama et al.

Nesta subseção apresentaremos o algoritmo desenvolvido por TSUKIYAMA et al. [43] para encontrar todos os conjuntos independentes maximais de um grafo. Este processo foi baseado nas idéias de PAULL e UNGER [38].

Antes porém, iremos mostrar que todos os CIM's de um grafo G qualquer podem ser obtidos através do seguinte procedimento:

Considere S_i o conjunto de todos os conjuntos independentes maximais do subgrafo induzido H de G, tal que $H = \{1, 2, \ldots, i\}$, para $1 \leq i \leq n$. Por definição, $S_0 = \emptyset$. O objetivo é construir S_i a partir de S_{i-1} de forma a

obter-se, sucessivamente, ao final do processo, S_n, a família de todos os K conjuntos independentes maximais para G.

Suponha que $S \in S_{i-1}$ e considerando que o conjunto $N(i)$ representa os vértices adjacentes a i, temos que $\begin{cases} \text{se } S \cap N(i) \neq \emptyset, & \text{então } S \in S_i. \\ \text{se } S \cap N(i) = \emptyset, & \text{então } S \cup \{i\} \in S_i, \end{cases}$

E, se $S' \in S_i$ temos que: $\begin{cases} \text{se } \{i\} \notin S', & \text{então } S' \in S_{i-1}, \\ \text{se } \{i\} \in S', & \text{então existe um } S \in S_{i-1} \\ & \text{tal que } S' - \{i\} \subseteq S. \end{cases}$

Analisando as relações acima, percebemos que um conjunto independente maximal S' qualquer em S_i ou já pertencia a S_{i-1}, ou pode ser derivado de algum conjunto independente maximal S em S_{i-1} através da fórmula: $S' = (S - N(i)) \cup \{i\}$.

Dessa forma, se unirmos todos os conjuntos S', obtidos pela fórmula acima, com todos os conjuntos S pertencentes a S_{i-1} teremos uma família F que conterá todos os conjuntos de S_i.

$$F = \{S' | S' = (S - N(i)) \cup \{i\}, S \in S_{i-1}\} \cup S_{i-1}$$

Assim, resta-nos extrair de F os elementos de S_i. A forma mais intuitiva seria a de comparar todos os elementos de F, dois a dois, selecionando apenas os maximais e evitando as repetições. Essa maneira não é muito eficaz, pois, para cada iteração, deveríamos armazenar toda a família F.

Para incluir em S_i somente os conjuntos independentes que sejam maximais, Tsukiyama et al. apresentam o seguinte procedimento:

Para cada $S \in S_{i-1}$ faça:

(i) Se $S \cap N(i) \neq \emptyset$, então $S \in S_i$;

(ii) Calcule $S' = (S - N(i)) \cup \{i\}$. S' será um CIM de S_i se e somente se para todo vértice j, tal que $j < i$, $j \notin S$, tivermos $N(j) \cap S' \neq \emptyset$.

Outro ponto a ser considerado é a não inclusão em S_i de um conjunto maximal já pertencente a S_i. Podemos perceber facilmente que as duplicatas (quando estas existirem) são provenientes do passo (ii) no procedimento anterior. Senão, vejamos: um dado $S' \in S_i$ pode ser obtido através de diferentes conjuntos $S^1, S^2 \in S_{i-1}$ tal que $S' = (S^1 - N(i)) \cup \{i\} = (S^2 - N(i)) \cup \{i\}$.

A fim de evitar as duplicatas, utiliza-se a seguinte regra: dado um conjunto $S \in S_{i-1}$, incluiremos $S' = (S - N(i)) \cup \{i\}$ em S_i somente se, $S \cap N(i)$ for o *lexicograficamente* menor conjunto dentre todos os $S \in S_{i-1}$ para os quais $S - N(i)$ é idêntico.

O que o algoritmo faz, na realidade, é executar os passos (i) e (ii) do procedimento prévio e, para cada S' gerado, tal que este seja maximal, testar se ele é o lexicograficamente menor. Isso ocorrerá, se e somente se, não existir um vértice $j < i$, $j \notin S$, tal que:

a) $N(j) \cap (S - N(i)) = \emptyset$ e

b) $N(j) \cap (S \cap N(i)) \cap \{1, 2, \ldots, j-1\} = \emptyset$

De modo a facilitar o entendimento, apresentaremos a seguir um exemplo prático do funcionamento do algoritmo, para tal, utilizaremos o grafo G da Figura 1.1.

$i = 1$ $\quad S = \emptyset \Rightarrow S \cap N(i) \neq \emptyset \rightarrow \emptyset \neq \emptyset \rightarrow (F)$, S não é CIM de S_1

$\Rightarrow S' = (S - N(i)) \cup \{i\} = \{1\} \rightarrow \nexists j < i, j \notin S$

$\rightarrow S'$ é CIM de S_1

$S_1 = \{1\}$

$i = 2$ $\quad S = \{1\} \Rightarrow S \cap N(i) \neq \emptyset \rightarrow S$ é CIM de S_2

$\Rightarrow S' = (S - N(i)) \cup \{i\} = \{2\} \rightarrow S'$ é CIM de S_2

$S_2 = \{1\}, \{2\}$

$i = 3$ $\quad S = \{1\} \Rightarrow$ é um CIM de S_3

$\Rightarrow S' = \{3\}$ é CIM, fazer teste lexicog.: $\nexists\, j < i, j \notin S$

$j = 2 \quad N(j) \cap (S - N(i)) = \emptyset \rightarrow \{1,3,6\} \cap \emptyset \rightarrow (V)$

$N(j) \cap (S \cap N(i)) \cap \{1, 2, \ldots, j-1\} = \emptyset$

$\{1,3,6\} \cap \{1\} \cap \{1\} \rightarrow \{1\} = \emptyset \rightarrow (F)$

é lexicograficamente menor $\rightarrow S' \in S_3$.

$S = \{2\} \Rightarrow$ é um CIM de S_3

$\Rightarrow S' = \{3\}$ é CIM, fazer teste lexicog.: $\nexists\, j < i, j \notin S$

$j = 1 \quad \{2,3,4\} \cap \emptyset \rightarrow \emptyset = \emptyset \rightarrow (V)$

$\{2,3,4\} \cap \{2\} \cap \emptyset \rightarrow \emptyset = \emptyset \rightarrow (V)$

não é lexicog. menor $\rightarrow S' \notin S_3$.

$S_3 = \{1\}, \{3\}, \{2\}$

$\vdots$

$i = 7$ $\quad S = \{1,5,6\} \Rightarrow$ é um CIM de S_7

$\Rightarrow S' = \{1,6,7\}$ é CIM e é o lexicog. menor $\rightarrow S' \in S_7$.

$S = \{3,4\} \Rightarrow$ não é um CIM de S_7

$\Rightarrow S' = \{3,4,7\}$ é CIM e é o lexicog. menor $\rightarrow S' \in S_7$.

$S = \{2,4\} \Rightarrow$ não é um CIM de S_7

$\Rightarrow S' = \{2,4,7\}$ é CIM e é o lexicog. menor $\rightarrow S' \in S_7$.

$S = \{4,6\} \Rightarrow$ não é um CIM de S_7

$\Rightarrow S' = \{4,6,7\}$ é CIM e é o lexicog. menor $\rightarrow S' \in S_7$.

$S = \{2,5\} \Rightarrow$ é um CIM de S_7

$$\Rightarrow S' = \{2,7\} \text{ não é um CIM} \rightarrow S' \notin S_7.$$

$$S_7 = \{1,5,6\},\{1,6,7\},\{3,4,7\},\{2,4,7\},\{4,6,7\},\{2,5\}$$

A obtenção de S_n a partir de S_1, através do processo acima, consiste na construção de uma árvore binária onde os conjuntos independentes maximais de S_i correspondem aos nós do nível i, sendo que a raiz está no nível 1.

Dado um grafo G qualquer, com conjunto de vértices $\{1,2,\ldots,n\}$, a árvore binária Q correspondente a G, obtida através do processo anterior é assim definida:

(1) $\{1\}$ é a raiz da árvore;

(2) Seja S um nó qualquer de Q situado no nível i. Se $i = n$, então S é uma folha. Logo, S será um conjunto independente maximal para G. Para $1 \leq i < n$, S será um nó *pai* e terá, no máximo, dois filhos, temos, portanto, dois subcasos:

(2.a) Se $S \cap N(i+1) = \emptyset$ (isto é, se S não possui vértices adjacentes ao vértice $i+1$), então o nó pai S terá somente um filho, à esquerda, formado por $S \cup \{i+1\}$.

(2.b) Se existe algum vértice pertencente a $N(i+1)$ e também a S, então S terá, potencialmente, 2 filhos. O primeiro, à direita, é uma cópia de S e está sempre presente. O potencial filho da esquerda será o nó $S' = (S - N(i+1)) \cup \{i+1\}$ se S' for um subconjunto independente maximal dos primeiros $i+1$ vértices. Note que, neste caso, S' será, potencialmente o filho de alguns conjuntos que estão no mesmo nível da árvore, mais precisamente, S' poderá ser filho de qualquer subconjunto independente maximal dos primeiros i

vértices que contém $S' - N(i+1)$. De todos esses conjuntos, S' será o filho daquele que for o lexicograficamente menor.

Percebe-se ainda, que existe uma correspondência de um para um entre os conjuntos independentes maximais de G e as folhas da árvore. Com isso, se construirmos a árvore em pré-ordem, em cada folha visitada teremos um novo conjunto independente maximal para o grafo G. Chamaremos a árvore Q de árvore dos quatro japoneses em homenagem aos autores que desenvolveram tal processo. Na Figura 2.2 podemos observar a árvore do tipo Q correspondente ao exemplo visto logo acima, note que a parte tracejada não chega a ser construída pelo algoritmo.

A análise de complexidade será realizada da seguinte forma: seja q o número de arestas e K o número de CIM's para o subgrafo induzido H. Iremos admitir que as operações de intersecção, união e diferença entre conjuntos são efetuadas em um tempo constante c.

No passo (i) executamos K operações entre conjuntos, o que produz um tempo da ordem de $O(K \cdot c)$. Já o passo (ii), em cada um dos K elementos pertencentes a S_{i-1} efetuaremos, no máximo, q operações entre conjuntos, pois a diferença entre o conjunto de vértices $\{1, 2, \ldots, i-1\}$ e a quantidade de vértices pertencentes a qualquer $S \in S_{i-1}$ sempre será maior que o valor q. Logo, o tempo gasto nesse passo é de $O(K \cdot q \cdot c)$.

A análise do último passo (teste lexicográfico) é idêntica à realizada no passo (ii), porém neste, para cada q, duas operações envolvendo conjuntos são realizadas. Produzindo então um tempo de $O(K \cdot q \cdot 2c)$.

Como a execução desses três passos ocorre, no pior caso, n vezes (uma para cada vértice de G), temos uma complexidade de tempo total de $O(n(K \cdot$

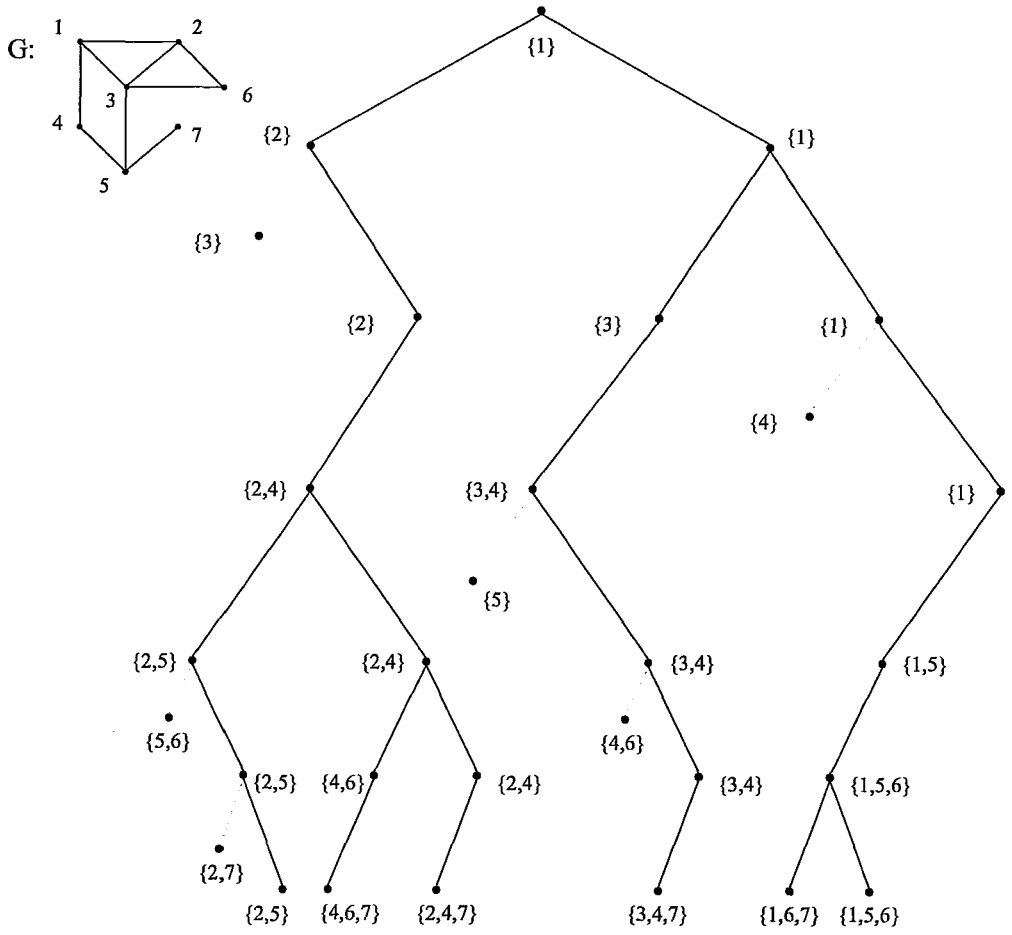

Figura 2.2: A árvore dos quatro japoneses para o grafo G.

$c + K \cdot q \cdot c + K \cdot q \cdot 2c)$) que resulta em $O(nqK)$, sabendo que $m > q$ temos: $O(nmK)$.

Considerando que este procedimento pode ser representado através da construção de uma árvore binária, cuja altura máxima é n. E, efetuando a computação dos nós através de um percurso em profundidade, podemos obter uma complexidade de espaço da ordem de $O(n + m)$.

2.1.2.2 O procedimento de Eppstein

Este algoritmo foi proposto por Eppstein [19] em 2003 em um artigo sobre o número cromático de um grafo. Tal procedimento pode ser classificado como sendo do tipo *branch-and-reduce*, pois seu funcionamento baseia-se na geração recursiva dos conjuntos independentes (não necessariamente maximais) de acordo com a análise de diversas situações em que um vértice v pode ocorrer em um grafo. Assim, o autor apresenta o seguinte limite superior para o número de conjuntos independentes maximais:

Teorema 2.9 (Eppstein, 2001). *Seja G um grafo com n vértices, e L um número não negativo. Então o número de conjuntos independentes maximais $S \subset V(G)$ para os quais $|S| \leq L$ é, no máximo, $3^{4L-n}4^{n-3L}$.*

Demonstração. Precisamos provar que o número de CIM's de cardinalidade no máximo L é de no máximo $3^{4L-n}4^{n-3L}$. Usaremos indução em n; no caso base $n = 0$ e existe apenas um (vazio) conjunto independente maximal, logo para qualquer $L \geq 0$ temos: $1 \leq 3^{4L}4^{-3L} = (81/64)^L$. Para $n > 0$, dividiremos a análise em casos de acordo com os graus dos vértices em G:

- Se G contém um vértice v de grau maior ou igual a três, então cada CIM S irá conter ou não v:

  a) $v \in S$, se $S\backslash\{v\}$ for um CIM de $G[V\backslash N[v]]$, logo o subgrafo $G[V\backslash N[v]]$ terá, pelo menos, quatro vértices a menos;

  b) $v \notin S$, se S for um CIM de $G[V\backslash\{v\}]$, e o subgrafo $G[V\backslash\{v\}]$ terá um vértice a menos;

Assim, temos:

$$\begin{aligned}
\text{n}^\text{o}\ \text{CIM}(G) &\leq \text{n}^\text{o}\ \text{CIM}(n-4, L-1) + \text{n}^\text{o}\ \text{CIM}(n-1, L) \\
&\leq 3^{4(L-1)-(n-4)}4^{(n-4)-3(L-1)} + 3^{4L-(n-1)}4^{(n-1)-3L} \\
&\leq 1/4 \cdot (3^{4L-n}4^{n-3L}) + 3/4 \cdot (3^{4L-n}4^{n-3L}) \\
&\leq 3^{4L-n}4^{n-3L} = 3^{4L-n}4^{n-3L}
\end{aligned}$$

- Se G contém um vértice v de grau igual a um. Seja u o seu vizinho, então cada CIM S conterá u ou v. Então:

$$\begin{aligned}
\text{n}^\text{o}\ \text{CIM}(G) &\leq \text{n}^\text{o}\ \text{CIM}(n-2, L-1) + \text{n}^\text{o}\ \text{CIM}(n-2, L-1) \\
&\leq 3^{4(L-1)-(n-2)}4^{(n-2)-3(L-1)} + 3^{4(L-1)-(n-2)}4^{(n-2)-3(L-1)} \\
&\leq 4/9 \cdot (3^{4L-n}4^{n-3L}) + 4/9 \cdot (3^{4L-n}4^{n-3L}) \\
&\leq 8/9 \cdot 3^{4L-n}4^{n-3L} \leq 3^{4L-n}4^{n-3L}
\end{aligned}$$

- Se G contém um vértice v de grau igual a zero, então todo CIM S conterá v. Logo:

$$\begin{aligned}
\text{n}^\text{o}\ \text{CIM}(G) &\leq \text{n}^\text{o}\ \text{CIM}(n-1, L-1) \\
&\leq 3^{4(L-1)-(n-1)}4^{(n-1)-3(L-1)} \\
&\leq 1/27 \cdot (3^{4L-n})\ 16 \cdot (4^{n-3L}) \\
&\leq 16/27 \cdot 3^{4L-n}4^{n-3L} \leq 3^{4L-n}4^{n-3L}
\end{aligned}$$

- Se G contém uma cadeia $u - v - w - x$ de vértices de grau dois, então cada CIM S conterá u, ou conterá v, ou não contém u e contém w:

$$\begin{aligned}
\text{n}^\text{o}\ \text{CIM}(G) &\leq 2 \cdot\ \text{n}^\text{o}\ \text{CIM}(n-3, L-1) + \text{n}^\text{o}\ \text{CIM}(n-4, L-1) \\
&\leq 2 \cdot 3^{4(L-1)-(n-3)}4^{(n-3)-3(L-1)} + 3^{4(L-1)-(n-4)}4^{(n-4)-3(L-1)} \\
&\leq 2/3 \cdot (3^{4L-n}4^{n-3L}) + 1/4 \cdot (3^{4L-n}4^{n-3L}) \\
&\leq 11/12 \cdot 3^{4L-n}4^{n-3L} \leq 3^{4L-n}4^{n-3L}
\end{aligned}$$

- No caso restante, G é formado pela união disjunta de triângulos, todos os conjuntos independentes maximais têm exatamente $n/3$ vértices, e existem exatamente $3^{n/3}$ CIM's [35]. Se $L \geq n/3$ então $3^{n/3} \leq 3^{4L-n}4^{n-3L}$. Se $L < n/3$, não existe CIM de cardinalidade no máximo L.

Assim sendo, em todos os casos, o número de conjuntos independentes maximais encontra-se dentro do limite proposto. □

A análise descrita acima, pode ser facilmente transformada em um algoritmo recursivo (Algoritmo 2.1) que obtém todos os conjuntos independentes maximais de G com tamanhos menores ou iguais a um limite L.

O Algoritmo 2.1 gera todos os CIM's de um grafo (subgrafo) menores do que um determinado limite. A variável H representa um conjunto de vértices que forma um subgrafo induzido em G. O conjunto de vértices S irá representar o CIM (inicialmente vazio) e, L limita o número de vértices de H que podem ser incluídos em S.

Na linha 3, temos a saída do algoritmo, a qual fornece os conjuntos independentes gerados. Alguns destes não são maximais, portanto, se somente CIM's são desejados, faz-se necessário a inclusão de um teste de maximalidade após a esta linha.

Para medir a complexidade deste algoritmo, notamos que ele é a transcrição em pseudocódigo da análise feita para os possíveis casos em que um vértice pode ocorrer. Vimos também que esta análise forma a prova do Teorema 2.9. Assim, o algoritmo executará, no máximo, $3^{4L-n}4^{n-3L}$ chamadas recursivas. Cada chamada recursiva pode ser implementada em tempo polinomial em relação ao tamanho do grafo H passado como parâmetro. Logo,

Algoritmo 2.1: CIM's de tamanho máximo L (Eppstein, 2003).

```
1  geraCIM (conjunto H, conjunto S, inteiro L):
2  se (H = ∅ ou L = 0) então
3      retorne S;
4  senão se (existe v ∈ H com grau(v, H) ≥ 3) então
5      geraCIM (H\{v}, S, L);
6      geraCIM (H\N[v], S ∪ {v}, L − 1);
7  fim
8  senão se (existe v ∈ H com grau(v, H) = 1) então
9      seja u o vizinho de v;
10     geraCIM (H\N[u], S ∪ {u}, L − 1);
11     geraCIM (H\N[v], S ∪ {v}, L − 1);
12 fim
13 senão se (existe v ∈ H com grau(v, H) = 0) então
14     geraCIM (H\{v}, S ∪ {v}, L − 1);
15 senão se (algum ciclo em H não é um triângulo ou L ≥ |H|/3) então
16     seja u, v e w vértices adjacentes de grau 2,
17     tal que (se possível) u e w são não adjacentes;
18     geraCIM (H\N[u], S ∪ {u}, L − 1);
19     geraCIM (H\N[v], S ∪ {v}, L − 1);
20     geraCIM (H\({u} ∪ N[w]), S ∪ {w}, L − 1);
21 fim
```

o tempo total é da ordem de $O^*(3^{4L-n}4^{n-3L})$.

2.1.2.3 O procedimento de Byskov

Apresentaremos nesta seção dois algoritmos propostos por Byskov [10] para a geração dos CIM's de um grafo G qualquer. O primeiro enumera todos os CIM's independentemente do tamanho L que cada CIM pode ter, já o segundo gera todos os CIM's de tamanho exatamente L. O autor propõe ainda outros dois algoritmos, um que produz todos os conjuntos de tamanho no máximo L e outro para os conjuntos de tamanho pelo menos L. Estes dois últimos não serão abordados neste trabalho pois apresentam como única modificação em relação aos demais o respectivo teste no tamanho dos CIM's a serem gerados, para maiores informações, consulte [10].

Assim como o algoritmo de Eppstein para geração de CIM's de tamanho no máximo L visto anteriormente, os algoritmos de Byskov que enumeram todos os CIM's de G, também são do tipo *branch-and-reduce* e, utilizam a idéia de, após escolhido um determinado vértice v, decidir incluí-lo ou não em um CIM.

Se S é um CIM de G que contém v então $S\backslash\{v\}$ é um CIM de $G[V\backslash N[v]]$. Caso contrário, ou seja, S não contém v, então S é um CIM de $G[V\backslash\{v\}]$ e em S existe um vértice u que é vizinho de v.

Para enumerar todos os CIM's do grafo (subgrafo) $G\backslash N[v]$ ou $G\backslash\{v\}$, o procedimento (Algoritmo 2.2) utiliza, linhas 6 e 7, chamadas recursivas sobre os grafos ou tenta incluir os vizinhos de v um por vez (linha 13).

Na variável S armazenaremos o conjunto independente, o qual é construído através da inclusão dos vértices de G. No passo inicial do processo,

$H = G$ e $S = \emptyset$, na linha 3 temos a saída do algoritmo, a qual retorna o conjunto independente S que nem sempre é maximal.

Algoritmo 2.2: CIM's independente do tamanho (Byskov, 2004).

```
1  geraCIM (conjunto H, conjunto S):
2  se (H = ∅) então
3      retorne S;
4  senão se (o maior grau em H é ≥ 3) então
5      seja v o vértice que tenha o maior grau em H;
6      geraCIM (H\N[v], S ∪ {v});
7      geraCIM (H\{v}, S);
8  fim
9  senão
10     seja v o vértice que tenha o menor grau em H;
11     geraCIM (H\N[v], S ∪ {v});
12     para todo vértice w ∈ N(v) faça
13         geraCIM (H\N[w], S ∪ {w});
14     fim
15 fim
```

Como dito inicialmente, este algoritmo também é do tipo *branch-and-reduce* e, para medirmos sua complexidade, consideraremos os seguintes casos:

(a) se o vértice v possui grau maior ou igual a três, então o algoritmo executará duas chamadas recursivas (fará duas ramificações), uma com pelo menos quatro vértices a menos e a outra com um vértice a menos, temos, segundo a notação vista na Seção 1.4, que: $T(n) = T(n-4) + T(n-1)$;

$$t = (4, 1)$$

$$\alpha_t = 1 - x^{-4} - x^{-1} \Rightarrow \alpha_t \approx 1,3802$$

(b) para todo v que tiver grau zero, simplesmente inclui-se v no conjunto independente, produzindo um tempo de $T(n) = T(n-1)$;

$$t = (1)$$

$$\alpha_t = 1 - x^{-1} \Rightarrow \alpha_t = 1$$

(c) caso v tenha grau igual um, e seja u o seu vértice vizinho, incluiremos v ou u no conjunto independente, e o tempo é de $T(n) = T(n-2) + T(n-2)$;

$$t = (2, 2)$$

$$\alpha_t = 1 - x^{-2} - x^{-2} \Rightarrow \alpha_t = \sqrt{2} \approx 1,4142$$

(d) se o algoritmo falhar nos casos anteriores, então todos os vértices restantes possuirão grau dois. Assim, para cada conjunto de três vértices, três ramificações são possíveis, e em cada uma delas, três vértices são removidos. Logo, temos: $T(n) = T(n-3) + T(n-3) + T(n-3)$.

$$t = (3, 3, 3)$$

$$\alpha_t = 1 - x^{-3} - x^{-3} - x^{-3} \Rightarrow \alpha_t = \sqrt[3]{3} \approx 1,4423$$

Tendo como base os conceitos da Seção 1.4, percebe-se que o pior caso (maior valor de α_t) encontra-se no item (d). Portanto, o algoritmo de Byskov para a geração de todos os CIM's (independente do tamanho) de um grafo G qualquer, é da ordem de $O^*(3^{n/3}) = O(1.4423^n)$.

O próximo algoritmo, também proposto por Byskov, gera todos os CIM's, de tamanho exatamente L, de um grafo G. O tempo necessário para a

obtenção de tais conjuntos é obtido através do teorema a seguir e encontra-se dentro de um fator polinomial do limite superior calculado em (2.2).

Teorema 2.10 (Byskov, 2004). *Qualquer grafo contém, no máximo,*

$$d^{(d+1)L-n}(d+1)^{n-dL} \tag{2.3}$$

conjuntos independentes maximais de tamanho exatamente L, para qualquer $d \in \mathbb{N}^*$, e o Algoritmo 2.3 enumera todos eles, dentro de um fator polinomial deste limitante.

Algoritmo 2.3: CIM's de tamanho exatamente L (Byskov, 2004).

```
1  geraCIM (conjunto H, conjunto S, inteiro L):
2  se (|H| = L) então
3      retorne S ∪ H;
4  senão se (|H| > L > 0) então
5      se (o vértice de maior grau em H é ≥ d, seja v este vértice) então
6          geraCIM (H\N[v], S ∪ {v}, L − 1);
7          geraCIM (H\{v}, S, L);
8      fim
9      senão
10         seja v o vértice que tenha o menor grau em H;
11         geraCIM (H\N[v], S ∪ {v}, L − 1);
12         para todo vértice w ∈ N(v) faça
13             geraCIM (H\N[w], S ∪ {w}, L − 1);
14         fim
15     fim
16 fim
```

Fazendo $d = \lfloor n/L \rfloor$ temos (2.2). E, o limite superior em (2.3) é justo somente para $n/(d+1) \leq L \leq n/d$.

Demonstração. Quando aplicamos o Algoritmo 2.3 em um grafo com n vértices, temos que o número de folhas na árvore de recursão será um limite superior tanto para o número de CIM's de tamanho L quanto para o tempo de execução do algoritmo.

A análise de casos deste algoritmo é semelhante a realizada no anterior, a única diferença ocorre na inclusão do parâmetro L. Assim, seja $T(n, L)$ a menor função que satisfaz a seguinte relação de recorrência:

$$T(n,L) = \max \begin{cases} T(n-(d+1), L-1) + T(n-1, L) \\ 1 \cdot T(n-1, L-1) \\ 2 \cdot T(n-2, L-1) \\ \vdots \\ d \cdot T(n-d, L-1) \end{cases}$$

onde $T(n, 0) = T(L, L) = 1$. Então $T(n, L)$ é um limite superior no número de folhas na árvore de recursão.

Devemos agora mostrar que (2.3) é um limite superior para $T(n, L)$:

- Se $n = L$, teremos, no máximo, um CIM, neste caso o grafo G é totalmente desconexo:

$$\begin{aligned} d^{(d+1)L-n}\ (d+1)^{n-dL} &= d^{(d+1)L-L}\ (d+1)^{L-dL} \\ &= d^{Ld}\ (d+1)^{-L(-1+d)} \\ &= (d^d)^L\ ((d+1)^{(-1+d)})^{-L} \\ &= (d^d/(d+1)^{d-1})^L \\ &> 1 \quad \text{para } d \geq 1. \end{aligned}$$

Logo, (2.3) é um limite superior.

- Se $L = 0$, também teremos no máximo um CIM, pois neste caso o grafo G é o grafo nulo:

$$\begin{aligned} d^{(d+1)L-n}\ (d+1)^{n-dL} &= d^{(d+1)\cdot 0-n}\ (d+1)^{n-d\cdot 0} \\ &= d^{-n}\ (d+1)^n \\ &= ((d+1)/d)^n \\ &> 1 \quad \text{para } d \geq 1. \end{aligned}$$

Logo, (2.3) é um limite superior.

Então, por indução, mostraremos para os demais casos:

$$\begin{aligned} &T(n-(d+1), L-1) + T(n-1, L) \\ &\quad \leq d^{(d+1)\cdot(L-1)-(n-d-1)}\ (d+1)^{(n-d-1)-d(L-1)} + \\ &\qquad d^{(d+1)L-(n-1)}\ (d+1)^{(n-1)-dL} \\ &\quad \leq (1+d)\cdot\left(d^{(d+1)L-n}\ (d+1)^{(n-dL-1)}\right) \\ &\quad \leq (1+d)\cdot d^{(d+1)L-n}\ (d+1)^{n-dL}\cdot(d+1)^{-1} \\ &\quad \leq d^{(d+1)L-n}\ (d+1)^{n-dL} \end{aligned}$$

(2.3) é um limite superior.

$$
\begin{aligned}
d \cdot T(n-d, L-1) &\leq d \cdot \left(d^{(d+1)\cdot(L-1)-(n-d)} \ (d+1)^{(n-d)-d(L-1)}\right) \\
&\leq d \cdot d^{(d+1)L-n-1} \ (d+1)^{n-dL} \\
&\leq d \cdot d^{(d+1)L-n} \ d^{-1} \cdot (d+1)^{n-dL} \\
&\leq d^{(d+1)L-n} \ (d+1)^{n-dL}
\end{aligned}
$$

(2.3) é um limite superior.

Restam os casos: $a \cdot T(n-a, L-1)$ onde $0 < a < d$

$$
\begin{aligned}
a \cdot T(n-a, L-1) &\leq a \cdot \left(d^{(d+1)\cdot(L-1)-(n-a)} \ (d+1)^{(n-a)-d(L-1)}\right) \\
&\leq a \cdot d^{(d+1)L-n} \cdot d^{-d-1+a} \ (d+1)^{n-dL} \cdot (d+1)^{d-a} \\
&\leq \frac{a \cdot (d+1)^{d-a}}{d^{d+1-a}} \cdot d^{(d+1)L-n} \ (d+1)^{n-dL} \\
&< d^{(d+1)L-n} \ (d+1)^{n-dL}
\end{aligned}
$$

(2.3) é um limite superior.

Sendo assim, (2.3) é um limite superior para $T(n, L)$. □

Percebe-se que o Algoritmo 2.3 utiliza estruturas bastantes parecidas com as do procedimento anterior, a única exceção é a inclusão do parâmetro L, que representará o número de vértices a serem incluídos no conjunto independente maximal (variável S). Notamos ainda que, na linha 3 encontra-se a saída do algoritmo, a qual poderá produzir conjuntos que não sejam maximais.

Capítulo 3

Algoritmos baseados em Colorações Independentes

Vimos no capítulo anterior, que o número cromático de um grafo G pode ser obtido através da construção de suas colorações independentes. No presente capítulo, apresentaremos os algoritmos exatos que, de alguma maneira, utilizam esse conceito para a determinação de $\chi(G)$.

Iniciamos com o algoritmo de CHRISTOFIDES [13], o qual foi o precursor na utilização de CIM's para a obtenção de uma coloração ótima de G. Em seguida, temos o método proposto por ROSCHKE e FURTADO [39], cuja idéia permite a construção de uma árvore de subgrafos mais compacta. O próximo algoritmo é o apresentado por LAWLER [32], que, através de técnicas de programação dinâmica, aplicadas aos subgrafos de G, consegue encontrar uma coloração independente ótima.

Na seção 3.4, descrevemos o algoritmo de busca em profundidade, proposto por SZWARCFITER [42], o qual, utilizando o percurso em profundidade na árvore de Christofides, obtém uma menor complexidade de espaço para o problema. O algoritmo seguinte, cuja autoria deve-se à EPPSTEIN [19],

foi o primeiro método a reduzir a complexidade de tempo defendida por Lawler. Para alcançar tal resultado, o autor limita o tamanho dos conjuntos independentes maximais a serem testados. Finalizando o capítulo, apresentamos o método de BYSKOV [8, 11], o qual, atualmente, detém a menor complexidade de tempo.

3.1 O algoritmo de Christofides

O primeiro pesquisador a utilizar a idéia de encontrar uma coloração ótima de G entre as suas colorações independentes foi CHRISTOFIDES [13]. Apresentaremos nesta seção a forma através da qual ele obtém tal resultado. Antes porém, faz-se necessária a introdução de algumas definições:

Um *subgrafo p-colorível maximal* de um grafo G é o subgrafo induzido H_p o qual é p-cromático (pode ser colorido com p cores, tal que vértices adjacentes recebam cores distintas) e que não exista um $H'_p \supset H_p$ que também seja p-cromático. Desta forma, um subgrafo 1-colorível maximal H_1 é todo e qualquer CIM de G. O número cromático de G será dado pelo menor valor de p tal que $H_p = G$.

Todo subgrafo p-colorível maximal pode ser obtido através de um subgrafo $(p-1)$-colorível maximal de acordo com a seguinte relação:

$$H_p = H_{p-1} \cup H_1[V \backslash H_{p-1}] \tag{3.1}$$

onde:

H_{p-1} é qualquer subgrafo da família de subgrafos $(p-1)$-colorível maximais de G.

$H_1[V \backslash H_{p-1}]$ é qualquer CIM do subgrafo formado pelos vértices de G excluindo os pertences a H_{p-1}.

Sabendo que, para cada grafo (subgrafo) existe uma correspondente quantidade de subgrafos 1-colorível maximais (CIM's), temos que a família de subgrafos p-coloríveis maximais pode ser calculada através da união de cada subgrafo $(p-1)$-colorível maximal H_{p-1} com todos os subgrafos 1-colorível maximais $H_1[V \backslash H_{p-1}]$ correspondentes.

Além disso, é evidente que se um conjunto $H_p \supseteq H'_p$, então H'_p deve ser removido da família de subgrafos p-coloríveis maximais.

A seguir, apresentaremos o algoritmo de Christofides, o qual obtém o número cromático de G através da construção sucessiva das famílias de subgrafos p-coloríveis maximais. Tal construção é realizada através da equação (3.1) a qual produz somente colorações independentes. Para facilitar a compreensão do processo, considere F_k como sendo a família de subgrafos k-coloríveis maximais de G. A variável k indicará a colorabilidade de cada família e, ao final do algoritmo, conterá o valor do número cromático.

(1) Encontre todos os subgrafos 1-colorível maximais de G. Tais conjuntos formarão a família F_1. Faça $k = 1$.

(2) Para cada $H_k \in F_k$ vá para o passo (2.a). Se todos os $H_k \in F_k$ já foram examinados, faça $k = k + 1$ e repita esse passo.

(2.a) Encontre os CIM's do grafo $G' = G \backslash H_k$. Se um existe e ainda não foi testado, vá para o passo (2.b). Se todos já foram encontrados, volte para (2).

(2.b) Seja S o CIM encontrado no passo (2.a), calcule $H_{k+1} = H_k \cup S$. Se $H_{k+1} = G$, pare. O valor $k+1$ é o número cromático de G. Se $H_{k+1} \neq G$ vá para o passo (2.c).

(2.c) Se $H_{k+1} \subseteq H'_{k+1}$ para algum $H'_{k+1} \in F_{k+1}$, faça $F_{k+1} = F_{k+1}$. Senão, se $H_{k+1} \supseteq H'_{k+1}$ então $F_{k+1} = \{F_{k+1} - H'_{k+1}\} \cup H_{k+1}$. Caso contrário, $F_{k+1} = F_{k+1} \cup H_{k+1}$. Volte para o (2.a).

Mostraremos o funcionamento do algoritmo, aplicando-o ao grafo da Figura 1.1 que temos usado como exemplo:

Passo (1): $F_1 = \{\{1,5,6\},\{1,6,7\},\{3,4,7\},\{2,4,7\},\{4,6,7\},\{2,5\}\}$

Passo (2): $H_k = \{1,5,6\}$

Passo (2.a): $G' = G \backslash H_k = \{2,3,4,7\}$ $\quad S(G') = \{\{2,4,7\},\{3,4,7\}\}$

Passo (2.b): $S = \{2,4,7\}$ $\quad H_2 = H_k \cup S = \{1,5,6 \uparrow 2,4,7\}$

Passo (2.c): $F_2 = F_2 \cup H_2 = \{1,5,6 \uparrow 2,4,7\}$

Passo (2.b): $S = \{3,4,7\}$ $\quad H_2 = \{1,5,6 \uparrow 3,4,7\}$

Passo (2.c): $F_2 = \{\{1,5,6 \uparrow 2,4,7\},\{1,5,6 \uparrow 3,4,7\}\}$

Passo (2): $H_k = \{1,6,7\}$

Passo (2.a): $G' = \{2,3,4,5\}$ $\quad S(G') = \{\{2,5\},\{2,4\},\{3,4\}\}$

Passo (2.b): $S = \{2,5\}$ $\quad H_2 = \{1,6,7 \uparrow 2,5\}$

Passo (2.c): $F_2 = F_2 = \{\{1,5,6 \uparrow 2,4,7\},\{1,5,6 \uparrow 3,4,7\}\}$

$\vdots$

Passo (2): $H_k = \{2,5\}$

Passo (2.a): $G' = \{1,3,4,6,7\}$ $\quad S(G') = \{\{3,4,7\},\{4,6,7\},\{1,6,7\}\}$

Passo (2.b): $S = \{3,4,7\}$ $\quad H_2 = \{2,5 \uparrow 3,4,7\}$

$\vdots$

Passo (2.c): $F_2 = \{\{1,5,6 \uparrow 2,4,7\},\{1,5,6 \uparrow 3,4,7\},\{3,4,7 \uparrow 2,5\}\}$

Passo (2): $k = k+1 \rightarrow k = 2$

Passo (2): $H_k = \{1,5,6 \uparrow 2,4,7\}$

Passo (2.a): $G' = \{3\}$ $\quad S(G') = \{3\}$

Passo (2.b): $S = \{3\}$ $\quad H_3 = \{1,5,6 \uparrow 2,4,7 \uparrow 3\} = V$

Assim, $\chi(G) = 3$ e a correspondente coloração é: $\{1,5,6\}$, $\{2,4,7\}$, $\{3\}$. Outras colorações como: $\{1,5,6\}$, $\{3,4,7\}$, $\{2\}$ e $\{3,4,7\}$, $\{2,5\}$, $\{1,6\}$ também podem ser obtidas. Entretanto, colorações do tipo: $\{1,5,6\}$, $\{3,4\}$, $\{2,7\}$ são válidas, porém não são independentes, logo não são produzidas.

3.1.1 A árvore de Christofides

Apresentaremos agora uma outra maneira de descrever o algoritmo de Christofides visto anteriormente. Esta nova abordagem se justifica pois, além de auxiliar no entendimento, facilita a análise de complexidade do processo.

Um *galho* de uma árvore é o caminho simples que liga dois nós da mesma. Dizemos que um nó p é *pai* de um nó f (*filho*) se p encontra-se no nível t, f no nível $t+1$ e existe um galho que liga p a f. Um nó que não possui filhos é chamado de *nó folha*. Uma árvore composta por nós que possuem, no máximo, dois filhos é chamada de *árvore binária*. Toda árvore possui um nó especial r, denominado *raiz*, o qual, diferente dos demais, não possui pai.

Uma *árvore de subgrafos* ou *árvore de Christofides* de um grafo G é a árvore T, cujos nós são identificados por algum subgrafo de G e cujos galhos são representados por conjuntos de vértices de G, da seguinte maneira:

(i) O grafo G é o nó raiz da árvore T, estando no nível 0;

(ii) Seja H um nó de T, localizado no nível t, com conjunto de vértices V', ou seja, $H = G[V']$ (H é um subgrafo de G induzido por V'). Se V' for vazio, então H é um grafo nulo, sendo, portanto, uma folha de T. Caso contrário, sejam $S_1, S_2, \ldots, S_i$ os CIM's de H, tal que $i \geq 1$. Neste caso, H tem i filhos no nível $t+1$, são eles: $G[V' \backslash S_1], G[V' \backslash S_2], \ldots, G[V' \backslash S_i]$, e i galhos $S_1, S_2, \ldots, S_i$, ligando H a cada um dos seus filhos, respectivamente.

Utilizando o conceito dessa nova estrutura, o que o algoritmo faz é, em outras palavras, construir, sucessivamente, os níveis da árvore de subgrafos de G, até atingir a primeira folha de T. O nível em que se encontra essa folha será o número cromático de G. As classes de cor de uma coloração

independente ótima estarão representadas pela seqüência de galhos (CIM's) que ligam a raiz de T à folha atingida.

A idéia do algoritmo consiste, basicamente, em:

(1) Faça $t = 0$ e coloque G como nó raiz de T, no nível t;

(2) Encontre todos os CIM's de cada nó H pertencente ao nível t;

(3) Para cada CIM S_i de cada H, cria-se um filho H_i de H, pertencente ao nível $t+1$ e cujos elementos são formados pelos vértices do nó pai (H) menos os vértices encontrados em S_i;

(4) Para cada filho H_i de H, testar:

(4.a) Se H_i é um grafo nulo (é uma folha), PARE. Caso contrário, vá para o passo (4.b);

(4.b) Se H_i já foi obtido anteriormente no nível $t+1$, ou seja, H_i é filho de um pai anterior a H, então remova o filho H_i de H;

(5) Faça $t = t + 1$ e volte para o passo (2).

Na Figura 3.1 apresentamos um exemplo de uma árvore do tipo T para o grafo G. Note que, a parte tracejada não chega a ser construída pelo algoritmo.

É fácil perceber que o algoritmo utiliza, para a construção da árvore de subgrafos, o percurso em largura, pois somente passa para o nível $t+1$ após completar o nível t. Ramificações da árvore que chegam a um nó que contém um subgrafo já obtido em outro nó do mesmo nível são descartadas, pois forneceriam colorações equivalentes de G. O processo termina quando a primeira folha é encontrada. O número cromático será o valor de $t+1$ (pois a árvore começa no nível 0).

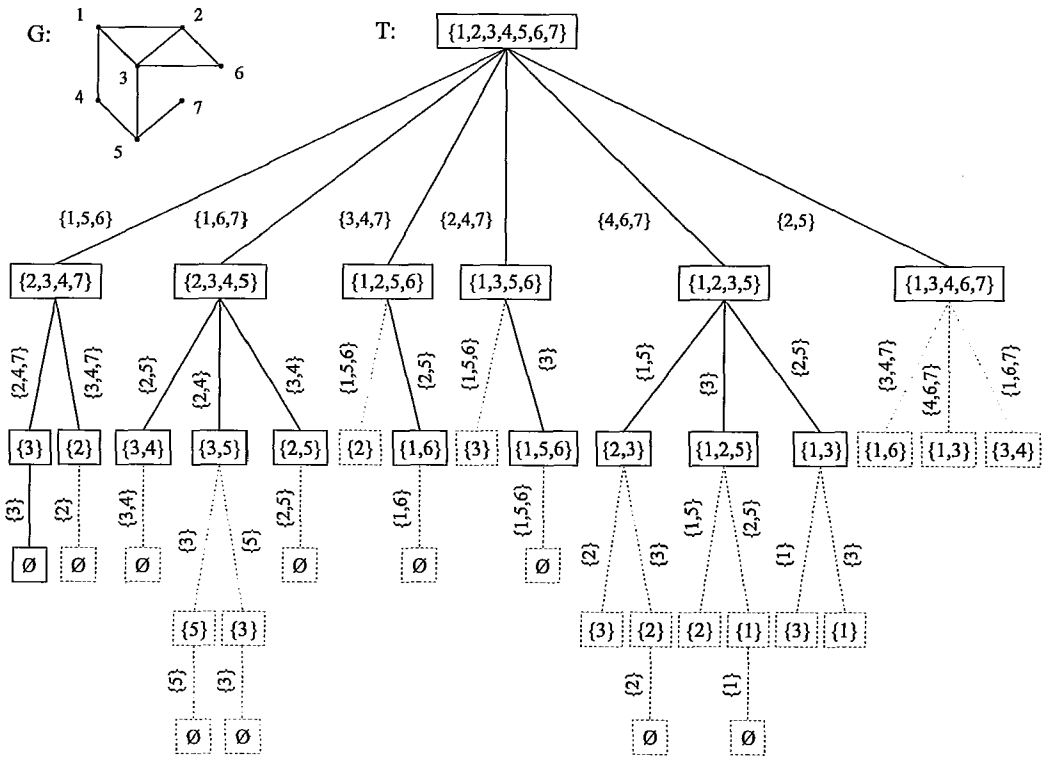

Figura 3.1: A árvore de subgrafos para o grafo G.

Por questões práticas, iremos calcular a complexidade deste algoritmo através da análise da árvore de subgrafos. Assim sendo, podemos afirmar que a complexidade irá depender diretamente do número de níveis existentes (altura) na árvore, que, no pior caso, será da ordem de $O(n)$.

MOON e MOSER [35] demonstraram que um grafo com n vértices possui como limite superior $3^{n/3}$ conjuntos independentes maximais, assim, o número de nós em cada nível t é, no máximo, $O((3^{n/3})^t)$.

TSUKIYAMA et al. [43] mostraram que os CIM's de um grafo com n vértices podem ser identificados em $O(mnK)$, onde K é o número de CIM's encontrados. Obtemos assim, o valor de $O(mn(3^{n/3})^t)$ para o cálculo dos CIM's dos nós do nível t.

Somando o valor de $mn(3^{n/3})^t$ para cada um todos os n níveis da árvore, temos:

$$\sum_{t=1}^{n} mn(3^{n/3})^t < \sum_{t=1}^{n} mn(3^{n/3})^n = mn^2(3^{1/3})(3^{n^2/3})$$

Logo a complexidade é de $O(mn^2(3^{1/3})^{n^2})$. Sendo $3^{1/3} \approx 1.4422$, temos que a complexidade de tempo do algoritmo é de $O^*(1.4422^{n^2})$.

Sabemos que o número de conjuntos independentes maximais de um grafo qualquer é de ordem exponencial [35]. Vimos que o algoritmo calcula e armazena simultaneamente os CIM's de G e de subgrafos de G. No pior caso, todos os subgrafos de G serão armazenados, como um grafo qualquer possui 2^n subgrafos, temos que a complexidade de espaço do algoritmo é de $O(2^n)$.

3.2 O algoritmo de Roschke e Furtado

Baseado na idéia original de Christofides, destacamos nesta seção, o trabalho apresentado por ROSCHKE e FURTADO [39] em 1973.

Neste artigo, eles propõem uma estratégia do tipo *branch-and-bound*, a qual evita que alguns ramos da árvore de Christofides sejam construídos, resultando em uma estrutura com o objetivo de reduzir o número de nós.

Para possibilitar a aplicação desta estratégia, consideraremos que cada nó H da árvore de subgrafos de G possui uma lista de CIM's candidatos, os quais serão utilizados em uma futura seleção.

A construção da árvore é feita da seguinte maneira: a partir de um nó H qualquer, escolhe-se um vértice $v \in H$ (ainda não colorido), tal que v esteja presente no menor número possível de CIM's candidatos de H. Seja t esse número, então tem-se o conjunto $S_1, S_2, \ldots, S_t$ de CIM's candidatos (organizados em uma ordem decrescente de cardinalidade) que contém v.

Vimos, através do Teorema 2.1, que se um grafo G é k-cromático, então existe uma coloração ótima $C = \{C_1, C_2, \ldots, C_k\}$ que pode ser obtida utilizando-se um conjunto $S = \{S_1, S_2, \ldots, S_k\}$ de CIM's, tal que $\cup_{i=1}^{k} S_i = V$.

Desta forma, existe uma coloração ótima de G que irá conter, pelo menos, um S_i com $1 \leq i \leq t$ (pois o vértice v deve ser coberto). Portanto, é suficiente ramificar o nó H em somente t filhos: $H \backslash S_1, H \backslash S_2, \ldots, H \backslash S_t$.

Naturalmente, o conjunto de CIM's candidatos de um nó H corresponderá aos CIM's do subgrafo $G[H]$. No entanto, ao invés de executar novamente um algoritmo para a geração dos CIM's (como sugere o método de Christofides), os autores mostram que é possível obter a lista $S^{(H)}$ de CIM's candidatos de um nó H (exceto a raiz) através da lista $S^{(H_p)}$ de CIM's candidatos do nó pai H_p de H. Para tal, considere as seguintes regras:

Seja S_i o CIM pertencente a $S^{(H_p)}$ utilizado para gerar H. Para cada CIM $S \in S^{(H_p)}$, verificar:

a) se $S \cap S_i = \emptyset$, então, S será um CIM candidato de H;

b) se $S \cap S_i \neq \emptyset$, armazene $S' = S - S_i$ em um conjunto temporário.

A lista $S^{(H)}$ será formada pela união dos CIM's produzidos pela primeira regra com todos os S', pertencentes ao conjunto temporário, que não sejam subconjunto próprio de algum CIM em $S^{(H)}$. Desta forma, os autores mostram que somente é necessário aplicar o algoritmo para geração de CIM's uma única vez, e que esta é feita no nó raiz da árvore, o qual corresponde ao grafo G inicial.

Entretanto, se fizermos uma estimativa do número de passos utilizados pelo algoritmo, notaremos que este fato não reduz necessariamente a complexidade do processo, uma vez que para construir a lista de candidatos $S^{(H)}$

de um nó H devemos examinar toda a lista $S^{(H_p)}$ do nó pai H_p de H e que o tamanho desta lista pode ser, no pior caso, $3^{n/3}$.

Ainda, supondo que a lista de candidatos de um nó H tenha tamanho $3^{n/3}$, teremos que o número de filhos de H é de $3^{(n/3)-1}$, e que este valor corresponde a quantidade de vezes que um vértice ocorre para esse número de CIM's.

Assim sendo, podemos admitir que a complexidade de tempo é da mesma ordem de grandeza que a do método de Christofides, a qual, como visto anteriormente, é de $O^*(1.4422^{n^2})$.

Como a lista de CIM's candidatos de um nó H pode ser obtida através da lista de seu nó pai H_p, consegue-se construir a árvore de subgrafos utilizando um percurso em profundidade e que a altura dessa árvore será, no pior caso, da ordem de $O(n)$, e, admitindo que a lista de cada nó H tenha tamanho de $3^{n/3}$, obteremos uma complexidade de espaço da ordem de $O(n(3^{n/3}))$.

Para efeitos de comparação, consideraremos G como sendo o mesmo grafo exemplo utilizado no algoritmo de Christofides. Apresentamos, na Figura 3.2, a árvore de subgrafos reduzida construída através das técnicas vistas nesta seção.

O leitor pode comparar a árvore de subgrafos da Figura 3.2 do presente algoritmo com a árvore obtida pela aplicação do algoritmo de Christofides (Figura 3.1). A presente árvore é substancialmente menor para o mesmo exemplo. Acreditamos que o algoritmo de Roschke e Furtado apresente uma complexidade de caso médio tal que o situaria entre os melhores algoritmos existentes para o problema de coloração.

Antes de encerrarmos esta seção, gostariamos de citar o trabalho proposto, independentemente, por WANG [44] em 1974, o qual utiliza o mesmo

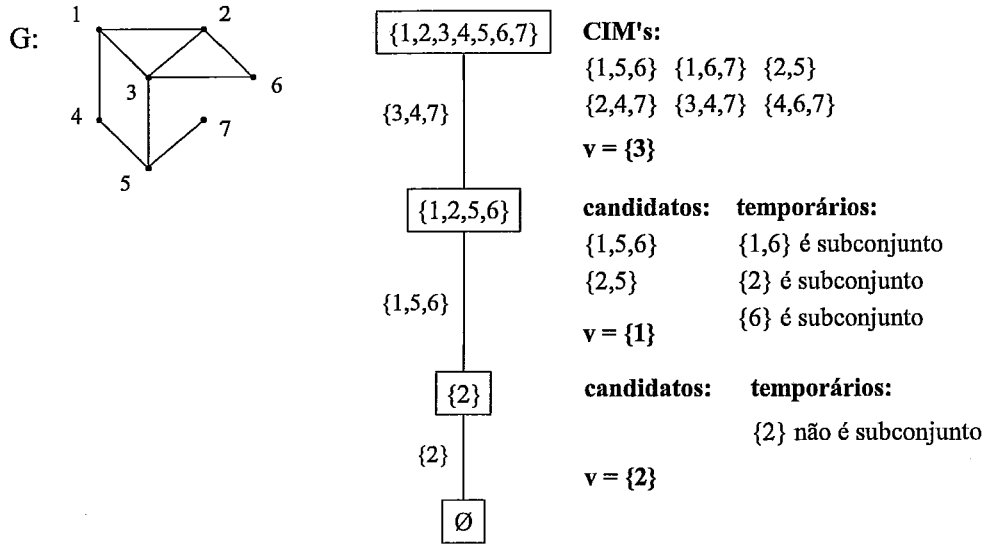

Figura 3.2: A árvore de subgrafos reduzida para o grafo G.

critério *branch-and-bound* visto anteriormente, porém, para cada subgrafo H da árvore, aplica-se (como no método de Christofides) um algoritmo para geração dos CIM's.

3.3 O algoritmo de Lawler

Em um artigo publicado em 1976 sobre complexidade de algoritmos para grafos, LAWLER [32] descreve como obter uma coloração independente ótima para um grafo qualquer, usando a técnica de programação dinâmica. Podemos perceber que nesta publicação, o principal enfoque do autor não foi o de apresentar um novo procedimento, nem tampouco melhorar os já existentes, mas sim propor uma nova formulação para o processo de modo a facilitar o cálculo da complexidade para esse tipo de algoritmo.

Utilizando o conhecimento do Teorema 2.1, Lawler percebeu que se um grafo G é k-colorível, então existe uma partição do conjunto de vértices V em

k conjuntos independentes, onde pelo menos um deles é um CIM. Ou seja, uma das k classes de cor de uma k-coloração ótima de G poderia ser atríbuida a este CIM. Assim, $\chi(G) = 1 + \chi(G \backslash S)$, para algum conjunto independente maximal S de G.

Estendendo esse resultado para um subgrafo induzido não vazio H' de G, temos que $\chi(H') = 1 + \chi(H' \backslash S)$, para algum conjunto independente maximal S de H'.

Sabemos que existe um número finito de CIM's para o subgrafo induzido H', precisamos encontrar S, um CIM de H' tal que o valor $\chi(H' \backslash S)$ seja o menor possível. Para isso, Lawler testa todos os CIM's de H', utilizando técnicas de programação dinâmica, representadas pelas seguintes equações:

$$\chi(H') = \begin{cases} 1 + \min_{S \subseteq H'} \{\chi(H' \backslash S)\} & \text{se } H' \neq \emptyset, \\ 0 & \text{se } H' = \emptyset. \end{cases} \tag{3.2}$$

Podemos notar que, estas equações, representam, de um modo algébrico, as idéias apresentadas anteriormente por CHRISTOFIDES [13].

O algoritmo funciona da seguinte maneira: seja um subconjunto $H \subseteq V$ de vértices, denotamos por $H' = G[H]$, o subgrafo de G o qual é induzido pelos vértices em H e, seja, $\chi(H')$ o número cromático de H'. Se H é vazio, então $\chi(H') = 0$. Para $H \neq \emptyset$ temos:

$$\chi(H') = 1 + \min\{\chi(H' \backslash S) : S \text{ é um CIM em } H'\}$$

Aplicando este procedimento, recursivamente, através dos conjuntos H em uma ordem de cardinalidade crescente, de modo que quando verificarmos H, todos os seus subconjuntos já foram examinados, teremos ao final da recursão calculado $\chi(H)$, onde $H = G$.

Algoritmo 3.1: Número Cromático (Lawler, 1976).

```
1 seja X um vetor de inteiros indexado de 0 até 2^n − 1;
2 X[∅] = 0;
3 para (H = 1) até (2^n − 1) faça
4     X[H] = ∞;
5     para todo CIM S do subgrafo induzido H de G faça
6         X[H] = min(X[H], X[H\S] + 1);
7     fim
8 fim
9 retorne X[V];
```

Vários algoritmos presentes neste trabalho utilizam técnicas de programação dinâmica. Nestes casos, usaremos um vetor X com uma entrada para cada um dos 2^n subgrafos induzidos de G e em cada entrada armazenaremos o número cromático do subgrafo correspondente. Iremos indexar X com subconjuntos de vértices, e por razões práticas, assumiremos que todos os subconjuntos de um subconjunto H são armazenados antes de H em X.

Assim, no algoritmo acima, percorremos X de 0 até G, calculando o número cromático de cada subgrafo através de (3.2), de modo que, ao final do processo obtemos o número cromático de G.

De modo a facilitar o entendimento do algoritmo, mostraremos seu funcionamento através de um exemplo. Para tal, considere G como sendo o grafo da Figura 1.1.

$$H = \{1\} \qquad X[H] = \infty \qquad S_1 = \{1\} \qquad X[H] = min(\infty, X[\emptyset] + 1) = 1$$

$$H = \{2\} \qquad X[H] = \infty \qquad S_1 = \{2\} \qquad X[H] = min(\infty, X[\emptyset] + 1) = 1$$

$$\vdots$$

$$H = \{7\} \qquad X[H] = \infty \qquad S_1 = \{7\} \qquad X[H] = min(\infty, X[\emptyset] + 1) = 1$$

$$\vdots$$

$$\begin{aligned} H = \{1,2\} \quad X[H] = \infty \quad & S_1 = \{1\} \quad X[H] = min(\infty, X[\{2\}] + 1) = 2 \\ & S_2 = \{2\} \quad X[H] = min(2, X[\{1\}] + 1) = 2 \end{aligned}$$

$$\begin{aligned} H = \{1,3\} \quad X[H] = \infty \quad & S_1 = \{1\} \quad X[H] = min(\infty, X[\{3\}] + 1) = 2 \\ & S_2 = \{3\} \quad X[H] = min(2, X[\{1\}] + 1) = 2 \end{aligned}$$

$$\vdots$$

$$H = \{6,7\} \quad X[H] = \infty \quad S_1 = \{6,7\} \quad X[H] = min(\infty, X[\emptyset] + 1) = 1$$

$$\begin{aligned} H = \{1,2,3\} \quad & S_1 = \{1\} \quad X[H] = min(\infty, X[\{2,3\}] + 1) = 3 \\ & S_2 = \{2\} \quad X[H] = min(3, X[\{1,3\}] + 1) = 3 \\ & S_3 = \{3\} \quad X[H] = min(3, X[\{1,2\}] + 1) = 3 \end{aligned}$$

$$\vdots$$

$$\begin{aligned} H = \{5,6,7\} \quad & S_1 = \{5,6\} \quad X[H] = min(\infty, X[\{7\}] + 1) = 2 \\ & S_2 = \{6,7\} \quad X[H] = min(2, X[\{5\}] + 1) = 2 \end{aligned}$$

$\vdots$

$$
\begin{array}{lll}
H = \{2,3,...,7\} & S_1 = \{2,4,7\} & X[H] = min(\infty, X[\{3,5,6\}] + 1) = 3 \\
& S_2 = \{2,5\} & X[H] = min(3, X[\{3,4,6,7\}] + 1) = 3 \\
& S_3 = \{3,4,7\} & X[H] = min(3, X[\{2,5,6\}] + 1) = 3 \\
& S_4 = \{4,6,7\} & X[H] = min(3, X[\{2,3,5\}] + 1) = 3 \\
& S_5 = \{5,6\} & X[H] = min(3, X[\{2,3,4,7\}] + 1) = 3
\end{array}
$$

$$
\begin{array}{lll}
H = \{1,2,...,7\} & S_1 = \{1,6,7\} & X[H] = min(\infty, X[\{2,3,4,5\}] + 1) = 3 \\
& S_2 = \{1,5,6\} & X[H] = min(3, X[\{2,3,4,7\}] + 1) = 3 \\
& S_3 = \{2,4,7\} & X[H] = min(3, X[\{1,3,5,6\}] + 1) = 3 \\
& S_4 = \{2,5\} & X[H] = min(3, X[\{1,3,4,6,7\}] + 1) = 3 \\
& S_5 = \{3,4,7\} & X[H] = min(3, X[\{1,2,5,6\}] + 1) = 3 \\
& S_6 = \{4,6,7\} & X[H] = min(3, X[\{1,2,3,5\}] + 1) = 3
\end{array}
$$

Para estimar a complexidade de tempo exigida pelo algoritmo, consideraremos o tempo requerido para computar as equações em (3.2).

Para todo conjunto de vértices $H' \subseteq V$. Suponha, para um fixo H', que já foi encontrado o $\chi(H'')$, para todos subconjuntos próprios $H'' \subseteq H'$. O tempo requerido para calcular $\chi(H')$ é então proporcional ao número de CIM's do subgrafo induzido em H' mais o tempo gasto para gerar esses CIM's.

Seja $|H'| = p$. Lembramos que o número de conjuntos independentes maximais para um grafo com p vértices é, no máximo, $3^{p/3}$ [35], e que todos os conjuntos independentes maximais de um grafo com p vértices podem ser gerados em tempo na ordem de $O(mpK)$, onde K é o número de subconjuntos independentes maximais [43].

Assim, temos que o tempo requerido para computar $\chi(H')$ é limitado por uma função da ordem de $O(mp3^{p/3})$. Somando o valor $mp3^{p/3}$ para todos

$H' \subseteq H$, temos:

$$\sum_{p=0}^{n} \binom{n}{p} mp3^{p/3} < mn \sum_{p=0}^{n} \binom{n}{p} 3^{p/3}$$

e sabendo que o binômio de Newton pode ser calculado através de:

$$(a+b)^n = \sum_{p=0}^{n} \binom{n}{p} a^{n-p} \cdot b^p$$

$$(1+x)^n = \sum_{p=0}^{n} \binom{n}{p} 1^{n-p} \cdot x^p = \sum_{p=0}^{n} \binom{n}{p} x^p$$

Admitindo que $x = 3^{1/3}$, temos uma complexidade de $O(mn(1+3^{1/3})^n)$. Sendo $1+3^{1/3} \approx 2.4422$, e ignorando os fatores polinomiais, a complexidade de tempo produzida pelo algoritmo é da ordem de $O^*(2.4422^n)$.

Este algoritmo, assim como o de CHRISTOFIDES [13], possui complexidade de espaço da ordem de $O(2^n)$, pois, no pior caso, calcula e armazena os CIM's de todos subgrafos de G.

3.4 O algoritmo de busca em profundidade

Pensando em melhorar a complexidade de espaço, SZWARCFITER [42] (c.f. SANTOS [40]) propôs em 1978, um algoritmo que constrói a árvore de Christofides utilizando o percurso em profundidade, ou seja, para cada subgrafo, representado na árvore como um nó, é gerado apenas um único CIM (caso exista), tal que garantidamente este já não tenha sido fornecido anteriormente, assim, consegue-se tratar seqüencialmente os conjuntos independentes maximais de um grafo, sem que haja necessidade de colocá-los ao mesmo tempo na memória, ou seja, ao invés de guardar para cada nível todos os nós filhos de um certo nó pai, armazena-se somente um nó de cada nível, à medida que segue-se em direção às folhas.

Através do algoritmo desenvolvido por TSUKIYAMA et al. [43] já se obtinha os CIM's de um grafo, um de cada vez. Faltava encontrar uma maneira na qual, conforme a produção de cada CIM, este pudesse ser imediatamente utilizado no processo de construção da árvore de Christofides.

Lembremos que a árvore de Christofides de um grafo G é uma árvore onde, cada nó representa algum subgrafo de G, encontrado durante o processo de obtenção de colorações independentes para G. Cada galho, que liga um nó *pai* a um nó *filho*, corresponde a um CIM do primeiro (nó pai). As classes de cor de uma coloração independente de G são os galhos que ligam a raiz a uma folha qualquer. A folha que se encontrar no menor nível possível, digamos nível k, determinará uma k-coloração exata para G.

Por simplicidade e para um melhor entendimento, apresentaremos novamente, na Figura 3.3, a árvore de Christofides para um grafo G, o qual servirá como exemplo ao longo desta seção.

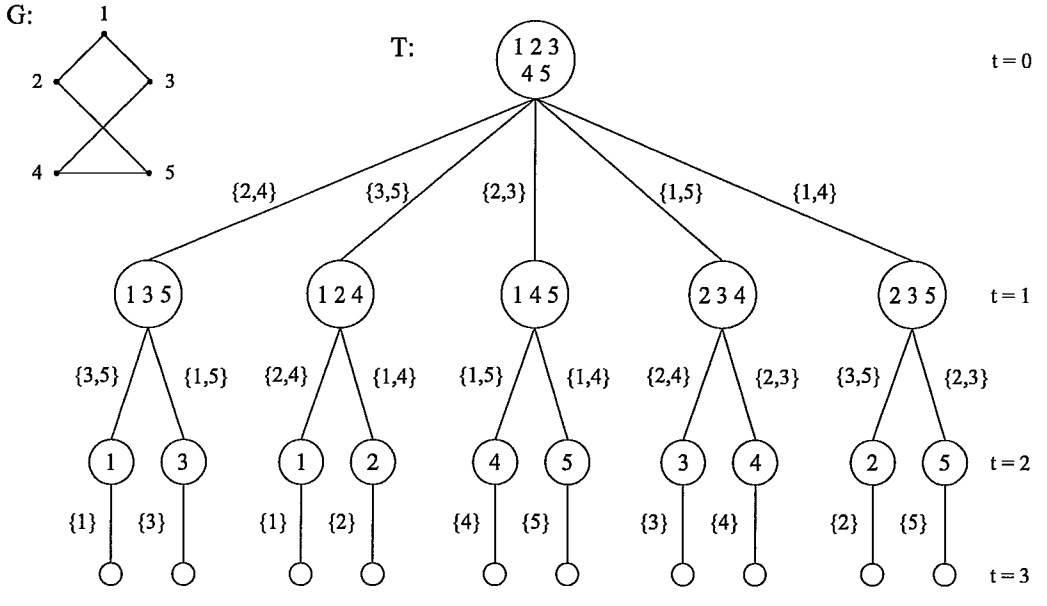

Figura 3.3: A árvore de Christofides para o grafo G.

Para realizar o correto entrosamento entre os algoritmos de TSUKIYAMA et al. [43] e o de CHRISTOFIDES [13], Szwarcfiter constrói uma nova árvore para o grafo G, a qual consiste no entrelaçamento da árvore de Christofides, T, para G, com várias árvores dos quatro japoneses, uma para cada nó não-folha de T, as quais substituem os galhos de T.

Dado um grafo G qualquer, seja W uma árvore para G, contendo nós do tipo T, correspondendo a subgrafos de G e nós do tipo Q, correspondendo a subconjuntos de vértices de G.

Definimos a árvore W da seguinte forma:

(i) G é a raiz de W, sendo, portanto do tipo T;

(ii) Seja H um nó de W do tipo T. Se H for um grafo vazio, então H não possui filhos, caso contrário, H possui um único filho H', do tipo Q, tal que H' é o nó raiz da árvore dos quatro japoneses, correspondente ao subgrafo H;

(iii) Seja H um nó de W do tipo Q. Se H for uma folha na árvore dos quatro japoneses, então H tem, exatamente, um filho H' na árvore W, sendo H', do tipo T, o qual consiste no subgrafo induzido pelos vértices que restaram da subtração algébrica do nó do tipo T antecedente a H' pelos vértices que compõem o nó H. Caso contrário, isto é, H não é uma folha na árvore dos quatro japoneses, então H tem um ou dois filhos em W, que são os mesmos filhos de H na árvore dos quatro japoneses.

Na Figura 3.4, apresentamos a árvore W, para o grafo G da figura anterior.

A idéia do algoritmo de busca em profundidade consiste, basicamente, em:

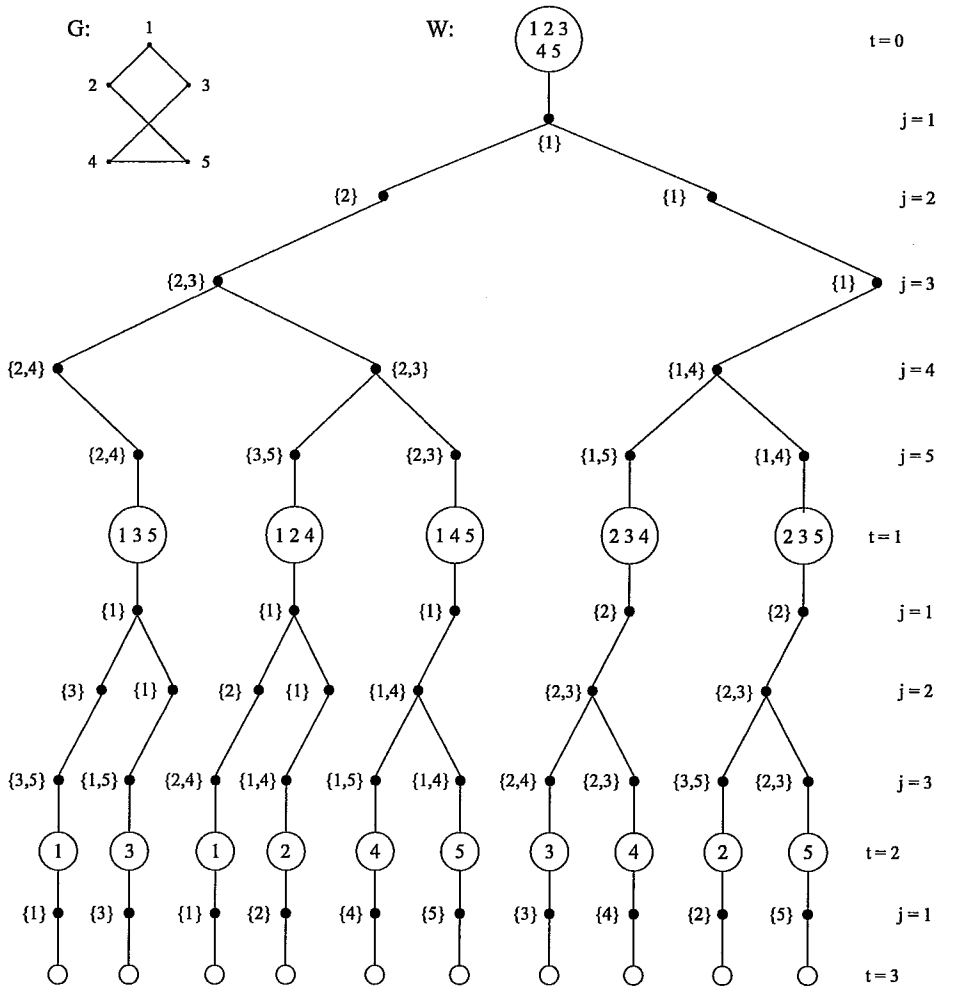

Figura 3.4: A árvore de busca em profundidade para o grafo G.

(1) Inicialize $k = n$, onde k representa um limite superior para $\chi(G)$ e faça $t = 1$, o qual representará os níveis dos nós do tipo T;

(2) Coloque o grafo G como nó raiz da árvore W;

(3) Supondo que os vértices estão classificados por valor crescente dos seus rótulos, faça $H = G$;

(4) Seja H um nó do tipo T, se H é vazio, vá para o passo (10). Senão, coloque o vértice v de menor índice pertencente a H como sendo o filho único de H, chamaremos este filho de S, marque v como selecionado;

(5) Escolha o vértice $j \in H$ de menor rótulo ainda não selecionado e, calcule $R = (S - N_j) \cup \{j\}$. Caso contrário vá para o passo (6);

(5.a) Se R é maximal e R é lexicograficamente menor, então o conjunto R será um nó do tipo Q, filho esquerdo de S. Se $S \cap N_j \neq \emptyset$, empilhe o estado geral do sistema (S, H e j). Faça $S = R$, marque j como selecionado e volte para o passo (5);

(5.b) Se $S \cap N_j \neq \emptyset$, então o nó S terá um filho à direita S', o qual é uma cópia de S. Faça $S = S'$, marque j como selecionado e volte para o passo (5);

(6) Se $t < k$, ou seja, se o nível t não ultrapassou o limite superior k para $\chi(G)$, vá para o passo (7), caso contrário vá para o passo (10).

(7) Se $|S| = n$ então atualize o limite superior de $\chi(G)$, fazendo $k = t$ e vá para o passo (10), senão vá para o passo (8);

(8) Faça $t = t + 1$, $n = n - |S|$ e $H = H - S$;

(9) S terá um único filho do tipo T, o qual será o subgrafo H, volte para o passo (4).

(10) Se a pilha estiver vazia, PARE. Senão, desempilhar o estado geral de S, H e de j contido na pilha e retornar ao passo (5.b) com estes valores.

Para fazer a análise de complexidade deste algoritmo lembremos que ele consiste no entrelaçamento de uma árvore de subgrafos com várias árvores

dos quatro japoneses. Sabemos que, no pior caso, a altura da árvore de subgrafos será da ordem de $O(n)$. Ainda, o número de CIM's de um grafo qualquer não ultrapassa $3^{n/3}$ [35]. Temos assim que o número de nós no nível t é menor que $O((3^{n/3})^t)$.

O algoritmo proposto por TSUKIYAMA et al. [43], constrói a árvore dos quatro japoneses gerando todos os conjuntos independentes maximais de um grafo com n vértices com tempo na ordem de $O(mnK)$, o qual produz um valor de $O(mn(3^{n/3})^t)$ para o cálculo dos CIM's dos nós do nível t.

Somando este valor para cada um todos os n níveis da árvore, temos:

$$\sum_{t=1}^{n} mn(3^{n/3})^t \leq \sum_{t=1}^{n} mn(3^{n/3})^n = mn(3^{1/3})^{n^2}$$

Sendo $3^{1/3} \approx 1.4422$, temos que a complexidade de tempo do algoritmo é de $O^*(1.4422^{n^2})$, a qual, como não poderia deixar de ser, é idêntica a do algoritmo de Christofides.

Para facilitar o entendimento sobre a medida da complexidade de espaço, utilizaremos o pseudocódigo do Algoritmo 3.2.

Antes de analisarmos tal complexidade, torna-se conveniente a apresentação de algumas observações à respeito do pseudocódigo. Este procedimento constrói, recursivamente, a árvore do tipo W para um grafo G qualquer, sendo t o nível da árvore de subgrafos e j o nível da árvore dos quatro japoneses em que se está a cada instante. O inteiro k representa a melhor aproximação já obtida para o número cromático até o momento, sendo igual a $\chi(G)$, ao término da execução.

As chamadas recursivas das linhas 6 e 8 representam, respectivamente, a inclusão em W dos filhos esquerdo e direito em um nó do tipo Q. Enquanto que a chamada da linha 17, corresponde a inclusão de um nó do tipo T.

Algoritmo 3.2: Número Cromático (Szwarcfiter, 1978).

```
1  COLORE (S, j)        /* S é um CIM de {v_1, v_2, ..., v_j}        */
2  j = j + 1;
3  se (j ≤ n) então
4      R = (S − N(v_j)) ∪ (v_j);
5      se (S ∩ N(v_j) ≠ ∅) então
6          COLORE (S, j);        /* gera filho da direita        */
7      se (R é maximal) e (R é lexicograficamente menor) então
8          COLORE (R, j);        /* gera filho da esquerda       */
9      fim
10 fim
11 se (t < k) então
12     se (|S| = n) então  k = t;
13     senão
14         t = t + 1;
15         n = n − |S|;
16         G = G − S;
17         COLORE ({v_1}, 1);
18         G = G + S;
19         n = n + |S|;
20         t = t − 1;
21     fim
22 fim
23 classifique cada lista de adjacência de G por valor crescente dos rótulos
24 dos vértices;
25 t = 1;      k = n;
26 COLORE ({v_1}, 1);
```

A operação $G - S$ indicada na linha 16 consiste em se obter o subgrafo de G induzido pelo conjunto de vértices $V \backslash S$. Para esse subgrafo uma nova rotulação de vértices $v_1, v_2, \ldots, v_p$ é designada, tal que $v_1 < v_2 < \ldots < v_p$.

A operação $G + S$ indicada na linha 18 consiste em se recompor o grafo existente antes da execução da linha 16, preservando a sua rotulação inicial.

Após efetuarmos tais observações, podemos calcular a complexidade de espaço utilizada pelo algoritmo. Lembremos que um limite superior para a altura da árvore W é dado por $n + (n-1) + (n-2) + \ldots + 1 = \frac{n(n+1)}{2}$, ou seja, $O(n^2)$ que corresponde a soma das alturas de todas as árvores dos quatro japoneses que ligam uma folha à raiz de W. Como a profundidade máxima da recursão efetuada pelo algoritmo é igual a altura da árvore W, temos que a primeira estimativa de espaço necessário é de $O((n+m)n^2)$, o que corresponde a armazenar, para cada nível da recursão, todo o grafo G considerado.

Nota-se, no entanto, que G pode ser armazenado uma única vez. Para tal, observamos que o algoritmo calcula $G - S$ (linha 16) e, posteriormente, realiza a recuperação do grafo original, através da operação $G + S$ (linha 18). Em outras palavras, ignoram-se (inserindo uma marca) todos os vértices $v \in S$, bem como todas as arestas que possuem v como um de seus vértices formadores, obtendo-se o subgrafo $G \backslash S$. Quando o processo voltar da recursão, recuperamos o grafo original, removendo os marcadores dos vértices $v \in S$ e das correspondentes arestas.

Ainda, percebe-se que para cada nível da recursão, utiliza-se um espaço de $O(n)$ para os conjuntos de vértices S e R, uma vez que cada um deles tem tamanho inferior a n.

Assim necessitamos armazenar, no pior caso, somente $n^2 \cdot 2n$, ou seja, para cada um dos n^2 níveis da árvore W, gasta-se um espaço de $2n$ (conjuntos S e R). Logo, uma nova aproximação de $O(n^3)$ para a complexidade de espaço é obtida.

Com algumas modificações estruturais, as quais serão vistas a seguir, o algoritmo exato de coloração pode ser implementado, de tal modo que, em cada nível da recursão o espaço utilizado seja tão somente $O(c)$, onde c é uma constante.

A variável local R pode ser transformada em global, uma vez que, entre sua geração e utilização (linhas 4 e 5, respectivamente) nenhuma chamada recursiva é efetuada.

Da mesma forma, transformaremos a variável local S em global. Para tal, deveremos recalcular S, sempre que necessário, sem provocar um aumento na complexidade de tempo.

Assim, no início da função *COLORE*, antes da operação $j = j+1$, calcula-se S da seguinte maneira: se $j = 1$, S é simplesmente $\{v_1\}$, caso contrário, precisamos saber se a função *COLORE* foi invocada por *COLORE* (R, j), linha 6, ou por *COLORE* (S, j), linha 8. Este problema é resolvido através da adição de uma variável booleana global. Se *COLORE* foi invocada pela linha 8, S permanecerá inalterado, caso *COLORE* for invocada pela linha 6, S é obtido através de $S = (S - N(v_j)) \cup \{v_j\}$.

Outro ponto onde S deve ser recalculado é o ponto de retorno imediatamente após *COLORE* (R, j), na linha 6, S é recuperado pela equação $S = (S - \{v_j\}) \cup ((S - \{v_j\}) \cap N(v_j))$ que é a operação inversa à efetuada na linha 4. É evidente que, no ponto de retorno após *COLORE* (S, j), na linha 8, não há necessidade de recalcularmos S, uma vez que S não foi alterado por essa chamada.

Finalmente, consideremos a chamada $COLORE\,(\{v_1\},1)$ da linha 17. Armazenaremos S em uma pilha imediatamente antes de $COLORE\,(\{v_1\},1)$ e o recuperaremos imediatamente depois de $COLORE\,(\{v_1\},1)$. Considerando que os CIM's que compõem essa pilha, em um instante qualquer do processo, são necessariamente disjuntos, temos que o tamanho máximo de tal pilha será de $O(n)$.

Nota-se que agora apenas a profundidade da recursão é considerada. Portanto, obtemos uma terceira aproximação para a complexidade de espaço da ordem de $O(n^2)$.

Em SANTOS [40] é descrito ainda como implementar o algoritmo com complexidade de espaço da ordem de $O(n+m)$ utilizando como idéia principal a eliminação da recursão.

3.5 O algoritmo de Eppstein

Em 2003, EPPSTEIN [19] apresentou um algoritmo exato para o problema de coloração em grafos com complexidades de tempo de $O^*(2.4150^n)$ e de espaço na ordem de $O(2^n)$, sendo assim, o primeiro pesquisador a melhorar a complexidade de tempo proposta por LAWLER [32] para o problema do número cromático.

A idéia de Lawler, vista na seção 3.3 e representada logo abaixo, consiste em, através de operações recursivas, calcular o número cromático de um subgrafo induzido H de G através da computação prévia de todos os seus subconjuntos. Vale ressaltar que, todos os subconjuntos H de V estão organizados em uma ordem numérica (ou qualquer outra ordem que garanta que todos os subconjuntos próprios de cada conjunto H já foram visitados antes de H):

para todo H, um subconjunto de vértices de G

$\chi(G[H]) = n$

para cada CIM S de $G[H]$ faça

$\chi(G[H]) = min(\chi(G[H]), \chi(G[H \backslash S]) + 1)$

A primeira mudança proposta por Eppstein, diz respeito à maneira de como o número cromático de um subgrafo $G[H]$ é obtido: ao invés de remover um conjunto independente maximal de cada subgrafo induzido H e computar o número cromático de $\chi(G[H])$ através deste subconjunto resultante, como faz o algoritmo de Lawler, adiciona-se um conjunto independente maximal S de $G[V \backslash H]$ e calcula-se o número cromático do superconjunto $\{H \cup S\}$ resultante, ou seja, usa-se o valor de $\chi(G[H])$ para atualizar todos os superconjuntos de H.

para todo H, um subconjunto de vértices de G

para cada CIM S de $G[V \backslash H]$ faça

$\chi(G[H \cup S]) = min(\chi(G[H \cup S]), \chi(G[H]) + 1)$

A verificação da corretude deste processo é simples: inicialmente, admite-se que $\chi(G[H])$ já foi calculado anteriormente. Sabemos que se um grafo é k-colorível, ele pode ser dividido em k classes de cor (conjuntos independentes) e pelo menos uma delas será um conjunto independente maximal (Corolário 2.2). Assim, suponha, sem perda de generalidade, que para um subgrafo induzido k-colorível H', uma de suas classes de cor será o conjunto independente maximal S e que existirá um subconjunto H de H', tal que $H' = \{H \cup S\}$, como $\chi(G[H])$ já foi obtido, teremos, para algum H, que $\chi(G[H']) = \chi(G[H \cup S]) \leq \chi(G[H]) + 1$.

Utilizando esta nova abordagem para a obtenção do número cromático, Eppstein mostra, através do teorema a seguir, que é suficiente calcular corretamente o número cromático de um subgrafo induzido H de G, somente quando este for um subgrafo j-cromático maximal.

Teorema 3.1 (Eppstein, 2003). *Se $G[H']$ é um subgrafo k-cromático maximal de G, existe um subgrafo $(k-1)$-cromático maximal $G[H]$ de G tal que $G[H' \backslash H]$ é um conjunto independente maximal de $G[V \backslash H]$.*

Demonstração. Para cada uma das k-colorações ótimas de $G[H']$, divida o conjunto de vértices de H' em $C_1, C_2, \ldots, C_k$ classes de cor tal que $|C_1| \geq |C_2| \geq \cdots \geq |C_k|$. Dentre todas as k-colorações, escolhemos aquela que possuir a classe de cor C_k de menor tamanho. Removendo esta classe de cor do conjunto H', obtemos um subconjunto H de H' que é $(k-1)$-cromático. Por construção, $G[H]$ é maximal em $G[H']$. Precisamos mostrar que $G[H' \backslash H]$ é um conjunto independente maximal de $G[V \backslash H]$. Se qualquer vértice $v \in G[V \backslash H']$ puder ser adicionado em $G[H]$ ou $G[H' \backslash H]$, então $G[H' \cup \{v\}]$ será k-cromático, o que contradiz a maximalidade de $G[H']$. Logo, $G[H]$ é maximal em G e $G[H' \backslash H]$ é um conjunto independente maximal de $G[V \backslash H]$. □

Assim, analisando o resultado obtido no teorema acima, concluímos que, existe uma técnica que determina o número cromático de um grafo G, calculando, recursivamente, o número cromático de todos os seus subgrafos $(k-1)$-cromático maximais, e que somente os CIM's S de $G[V \backslash H]$ serão analisados.

A segunda mudança defendida pelo autor é a de que podemos limitar o tamanho dos conjuntos independentes maximais S a serem gerados.

Suponha, sem perda de generalidade, que $G[H']$ é um subgrafo k-cromático maximal, logo $G[H']$ terá várias (talvez somente uma) k-colorações ótimas.

Em uma destas, existirá uma classe de cor C_k que será a menor possível (terá o menor número de vértices). Fazendo $H = \{H' - C_k\}$, temos que C_k corresponderá a algum CIM S de $G[V \backslash H]$ (Teorema 3.1) e que $G[H]$ é um subgrafo $(k-1)$-cromático maximal. Este por sua vez, também pode ser dividido em várias $(k-1)$-colorações ótimas, cuja menor classe de cor possível C_{k-1}, será menor ou igual a $\frac{|H|}{k-1} = \frac{|H|}{\chi(G[H])}$, pois se todas as classes de cor de $G[H]$ tivessem o mesmo tamanho, o valor de C_{k-1} seria exatamente igual a $\frac{|H|}{\chi(G[H])}$.

É fácil perceber que a menor classe de cor de $G[H]$ sempre será maior ou igual a menor classe de cor de $G[H']$, pois $\frac{|H|}{\chi(G[H])} \geq \frac{|H'|}{\chi(G[H'])}$. E, sabendo que algum CIM S de $G[V \backslash H]$ irá corresponder à menor classe de cor de $G[H']$, podemos restringir o tamanho dos CIM's em $|S| \leq \frac{|H|}{\chi(G[H])}$.

para todo H, um subconjunto de vértices de G
  $limite = |H|/\chi(G[H])$
  para cada CIM S de $G[V \backslash H]$, tal que $|S| \leq limite$
    $\chi(G[H \cup S]) = min(\chi(G[H \cup S]), \chi(G[H]) + 1)$

Uma vez que, podemos restringir o tamanho dos conjuntos independentes maximais de $G[V \backslash H]$. Eppstein, através do Teorema 2.9, mostra que nenhum grafo contém mais do que $3^{4L-n}4^{n-3L}$ conjuntos independentes maximais de tamanho no máximo L e que, existe um procedimento recursivo (Algoritmo 2.1) que consegue enumerar tais conjuntos em um tempo da ordem de $O^*(3^{4L-n}4^{n-3L})$.

Suponha que $L = n/3$, então o limite de Eppstein para o número de CIM's em um grafo é igual ao de Moon e Moser (Teorema 2.5). Entretanto, para $L < n/3$, o limite é menor.

Vimos que o tamanho L dos CIM's a serem gerados depende da relação $L \leq \frac{|H|}{\chi(G[H])}$. Assim sendo, para obter melhores resultados, temos: $n/3 \geq \frac{|H|}{\chi(G[H])}$ que resulta em $\chi(G[H]) \geq 3$. Em outras palavras, se $\chi(G[H]) \leq 2$ iremos admitir CIM's de tamanhos maiores que $n/3$ e, portanto, obteremos uma maior complexidade.

Desta forma, Eppstein trata, de maneira diferenciada, todos os subgrafos induzidos H cujo número cromático seja inferior a três. Para estes casos, utiliza-se um algoritmo que testa a 3-colorabilidade dos subgrafos H. Este procedimento (vide seção 5.2), também desenvolvido por Eppstein, possui complexidade de tempo da ordem de $O^*(1.3289^n)$.

Após apresentarmos as mudanças defendidas pelo autor e suas respectivas justificativas de utilização, explicaremos em maiores detalhes o funcionamento do procedimento (Algoritmo 3.3), bem como das estruturas por ele empregadas.

O algoritmo utiliza um vetor X, de tamanho 2^n, que será indexado pelos 2^n subconjuntos induzidos de G. O número cromático de cada subconjunto será armazenado em sua correspondente posição no vetor. A inicialização deste vetor ocorre da seguinte forma: para cada subconjunto induzido H, testamos, utilizando o algoritmo para 3-coloração de Eppstein (seção 5.2), se $\chi(G[H]) \leq 3$, caso seja, faz-se $X[H] = \chi(G[H])$, caso contrário, $X[H] = \infty$.

A seguir, analisa-se novamente todos os subconjuntos H em uma ordem numérica (exatamente da mesma forma como faz o algoritmo de Lawler). Quando visita-se H, verificamos se $X[H] < 3$. Em caso afirmativo, o processo continua sem precisar fazer qualquer alteração no vetor X e um novo subconjunto H é escolhido. Mas se $X[H] \geq 3$, faz-se uma busca pelos con-

juntos independentes maximais de $G[V \backslash H]$, limitando o tamanho de cada conjunto em $|H|/X[H]$. Para cada conjunto independente maximal S encontrado, calcula-se $X[H \cup S]$ como sendo o valor mínimo entre o valor existente em $X[H \cup S]$ e $X[H] + 1$.

Finalmente, após percorrer todos os subconjuntos, obtemos o valor de $X[V(G)]$ o qual é o número cromático de G.

Algoritmo 3.3: Número Cromático (Eppstein, 2003).

```
1  seja X um vetor indexado de 0 até 2^n − 1;
2  para (H = 0) até (2^n − 1) faça
3      se (χ(G[H]) ≤ 3) então
4          X[H] = χ(G[H]);
5      senão  X[H] = ∞;
6  fim
7  para (H = 0) até (2^n − 1) faça
8      se (3 ≤ X[H] < ∞) então
9          para todo CIM S de G[V\H] com |S| ≤ |H|/X[H] faça
10             X[H ∪ S] = min(X[H ∪ S], X[H] + 1);
11         fim
12     fim
13 fim
14 retorne X[V];
```

A complexidade de espaço é da ordem de $O(2^n)$, uma vez que está limitada pelo tamanho do vetor X, o qual possui 2^n posições.

A análise de complexidade de tempo é feita da seguinte forma: inicialmente, consideremos o tempo gasto para inicializar X. Para cada subconjunto de G, devemos testar se ele é 3-colorível. Como visto anteriormente,

EPPSTEIN [18] apresenta um algoritmo para o problema de 3-coloração com complexidade de tempo de $O(1.3289^n)$, gasta-se, então, na inicialização:

$$\sum_{H \subset V(G)} O\left(1.3289^{|H|}\right) = O\left(\sum_{i=0}^{n} \binom{n}{i} 1.3289^i\right)$$

Como visto na análise de complexidade do algoritmo de Lawler, temos que:

$$\sum_{i=0}^{n} \binom{n}{i} x^i = \sum_{i=0}^{n} \binom{n}{i} 1^{n-i} \cdot x^i = (1+x)^n$$

Considerando que $x = 1.3289$, temos que a complexidade produzida pela inicialização é de $(1 + 1.3289)^n = O(2.3289^n)$.

A segunda parte do algoritmo será executada somente para os subgrafos H onde $X[H] \geq 3$. Para cada H, precisamos apenas enumerar todos os conjuntos independentes maximais de $G[V \backslash H]$ de tamanho $L \leq \frac{|H|}{X[H]}$. Como $X[H] \geq 3$, temos que o maior tamanho possível para um conjunto independente maximal será quando $X[H] = 3$. Então, para efeitos de análise, somente conjuntos independentes de tamanho no máximo $|H|/3$ serão considerados. Para cada conjunto gerado, gasta-se um tempo constante para atualizar o valor de $X[H \cup I]$, enquanto que, segundo o Teorema 2.9, para gerarmos todos os conjuntos independentes maximais de tamanho no máximo L para um subgrafo H temos um tempo da ordem de $O^*(3^{4L-n}4^{n-3L})$. Assim,

o tempo da segunda parte pode ser limitado por:

$$\begin{aligned}
\sum_{H \subset V(G)} O^*\left(3^{4L-n}4^{n-3L}\right) &= \sum_{H \subset V(G)} O^*\left(3^{4\frac{|H|}{3}-|V\setminus H|}4^{|V\setminus H|-3\frac{|H|}{3}}\right) \\
&= O^*\left(\sum_{i=0}^{n}\binom{n}{i}3^{4\frac{|H|}{3}-|V\setminus H|}4^{|V\setminus H|-3\frac{|H|}{3}}\right) \\
&= O^*\left(\sum_{i=0}^{n}\binom{n}{i}3^{4\frac{i}{3}-(n-i)}4^{(n-i)-3\frac{i}{3}}\right) \\
&= O^*\left(\sum_{i=0}^{n}\binom{n}{i}\left(\frac{4}{3}\right)^{n-i}\left(\frac{3^{4/3}}{4}\right)^{i}\right) \\
&= O^*\left(\frac{4}{3}+\frac{3^{4/3}}{4}\right)^{n} \\
&= O^*(2.4150^n)
\end{aligned}$$

Assim, se compararmos os tempos gastos nas duas fases do algoritmo, percebe-se que o termo final domina o limite geral de tempo. Logo, temos uma complexidade de tempo da ordem de $O^*(2.4150^n)$.

3.6 O algoritmo de Byskov

Dentre todos os algoritmos existentes na literatura para a resolução exata do problema do número cromático em um grafo G qualquer, o de menor complexidade de tempo é o proposto por BYSKOV [8, 11], o qual está fundamentado nas idéias de EPPSTEIN [19]. A melhoria é obtida através de uma sutil modificação no algoritmo original, e da utilização de um melhor limite superior para o número de conjuntos independentes maximais de tamanho L.

Como visto anteriormente, o algoritmo de Eppstein utiliza um vetor $X[H]$ para todos $H \subseteq V$. Na primeira parte do procedimento, verifica-se, para cada

subgrafo induzido H de G, se $G[H]$ é 3-colorível utilizando um algoritmo para 3-coloração também proposto por Eppstein.

Na segunda parte percorre-se novamente todo o vetor de subgrafos X e, toda vez que um conjunto H é alcançado, atualiza-se X para todos os superconjuntos de H para os quais $X[H]$ potencialmente, produzirá melhores valores. Mais precisamente, para cada subconjunto H de V, cujo $X[H] \geq 3$, calcula-se todos os conjuntos independentes maximais S de $G[V \backslash H]$ de tamanho no máximo $|H|/X[H]$ e, para cada S encontrado, atualiza-se o valor de $X[H \cup S]$.

Ainda, Eppstein mostra que existe uma maneira de calcular o número cromático de um grafo G, utilizando os subgrafos $(k-j)$-coloríveis maximais $(0 \leq j < k)$. Assim sendo, Byskov também utiliza o conceito de subgrafos k-coloríveis maximais e, gera tais subgrafos através da seguinte técnica:

Técnica 3.2 (Byskov et al., 2004). *Para enumerar todos os subgrafos k-coloríveis maximais de um grafo $G = (V, E)$, gera-se todos os conjuntos independentes maximais S de G. E para cada S pegue a união de S e todos os subgrafos $(k-1)$-coloríveis maximais de $G[V \backslash S]$.*

A corretude da Técnica 3.2 vêm através da aplicação do Teorema 3.1 para $H' = V$. Esta técnica enumera todos os subgrafos k-coloríveis maximais e, talvez alguns que não sejam maximais. Mais detalhes em [12].

Apresentaremos agora o algoritmo de Byskov, o qual, para um melhor entendimento e posterior análise, será dividido em três partes. O código, em pseudolinguagem, deste procedimento pode ser conferido no Algoritmo 3.4.

A primeira é denominada de *marcando todos os subgrafos 3-coloríveis de um grafo G em um vetor*, e é idêntica a primeira parte do algoritmo de Eppstein, na qual verifica-se a 3-colorabilidade para todos os 2^n subgrafos induzidos de G.

Algoritmo 3.4: Número Cromático (Byskov, 2004).

```
1  seja X um vetor indexado de 0 até 2^n − 1 inicializado em ∞;
2  se (χ(G) ≤ 4) então
3      retorne χ(G);
4  marque todos os subgrafos maximais 4-coloríveis em X;
5  para (H = 0) até (2^n − 1) faça
6      se (4 ≤ X[H] < ∞) então
7          para todo CIM S de G[V\H] com |S| ≤ |H| / X[H] faça
8              X[H ∪ S] = min(X[H ∪ S], X[H] + 1);
9          fim
10     fim
11 fim
12 retorne X[V];
```

Na segunda parte, o autor encontra, através da aplicação da Técnica 3.2, todos os subgrafos 4-coloríveis maximais de G. Senão vejamos: inicialmente, geramos todos os conjuntos independentes maximais S de G. Para cada S encontrado, façamos $H = G[V \backslash S]$. Para cada subgrafo induzido H' de H, testamos se H' é 3-colorível, comparando-o no vetor. Se o sugrafo H' de H for 3-colorível, então $G[H' \cup S]$ será 4-colorível.

Após encontrar todos os subgrafos 4-coloríveis maximais, inicia-se a terceira etapa, a qual corresponde à segunda parte do algoritmo de Eppstein, porém, diferentemente do anterior, o algoritmo de Byskov somente irá exe-

cutar essa etapa para os subgrafos H tais que $X[H] \geq 4$.

Para calcular a complexidade do processo, iremos examinar separadamente cada etapa do algoritmo, uma vez que estas, por questões construtivas, estão bem definidas e são independentes entre si.

A primeira parte, como visto anteriormente, executa as mesmas operações que a etapa de inicialização do algortimo de Eppstein. Portanto, ambas possuem a mesma complexidade, que é da ordem de $O(2.3289^n)$.

A etapa seguinte, que consiste em encontrar todos os subgrafos 4-coloríveis maximais, apresenta como complexidades as definidas no Teorema 3.3 abaixo:

Teorema 3.3 (Byskov, 2004). *Todos os subgrafos 4-coloríveis (maximais) de um grafo pode ser enumerados em um tempo de $O(2.4023^n)$ e com espaço de $O(2^n)$.*

Demonstração. Vimos que para obtermos os subgrafos 4-coloríveis (maximais) de um grafo G, devemos gerar todos os conjuntos independentes maximais S de G e para cada S testar a 3-colorabilidade de todos os subgrafos H' de $G[V \backslash S]$. Tal processo pode ser representado pelo seguinte somatório:

$$\sum_{S \in S(G)} 2^{|V \backslash S|} \leq \sum_{L=1}^{n} \left|S^{=L}(G)\right| \cdot 2^{n-L}$$

O lado direito da expressão se justifica pois, $S^{=L}(G)$ representa o conjunto de CIM's de G com tamanho igual a L.

Podemos dividir o somatório em duas partes, a primeira com CIM's S de tamanho $L < n/5$ e a segunda com tamanho $L > n/5$.

$$\sum_{L=1}^{\lceil \frac{n}{5} \rceil - 1} \left|S^{=L}(G)\right| \cdot 2^{n-L} + \sum_{\lceil \frac{n}{5} \rceil}^{n} \left|S^{=L}(G)\right| \cdot 2^{n-L}$$

Utilizando o limite apresentado no Teorema 2.10 em cada uma das partes temos:

$$\sum_{L=1}^{\lceil \frac{n}{5} \rceil - 1} d^{(d+1)L-n}(d+1)^{n-dL} \cdot 2^{n-L} + \sum_{L=\lceil \frac{n}{5} \rceil}^{n} d^{(d+1)L-n}(d+1)^{n-dL} \cdot 2^{n-L}$$

Ainda neste mesmo teorema, vimos que o limite superior (2.3) para o número de CIM's de tamanho L é justo somente para $n/(d+1) \leq L \leq n/d$.

No somatório da esquerda temos que $L < n/5$, logo:

$$\begin{aligned} n/(d+1) &\leq L \\ n/(d+1) &\leq L < n/5 \\ n/(d+1) &< n/5 \\ d &> 4 \\ d &= 5. \end{aligned}$$

Procedemos da mesma forma para o somatório da direita, com $L > n/5$ e obtemos um valor de $d = 4$. Utilizando esses dois valores para d e substituindo-os nos seus respectivos somatórios temos:

$$\sum_{L=1}^{\lceil \frac{n}{5} \rceil - 1} 5^{6L-n}6^{n-5L} \cdot 2^{n-L} + \sum_{L=\lceil \frac{n}{5} \rceil}^{n} 4^{5L-n}5^{n-4L} \cdot 2^{n-L}$$

Uma maneira de resolver estes somatórios é mover os termos que não dependem de L para fora dos mesmos, obtendo assim duas séries geométricas:

$$6^n \cdot 2^n \cdot 5^{-n} \sum_{L=1}^{\lceil \frac{n}{5} \rceil - 1} 5^{6L} \cdot 6^{-5L} \cdot 2^{-L} + 4^{-n} \cdot 2^n \cdot 5^n \sum_{L=\lceil \frac{n}{5} \rceil}^{n} 4^{5L} \cdot 5^{-4L} \cdot 2^{-L}$$

Para calcular cada série geométrica usaremos a expressão $\dfrac{a_1 \cdot (q^n - 1)}{q - 1}$, onde a_1 corresponde ao primeiro termo, n é o número de termos e q é a razão

da série. Assim, no somatório da esquerda temos:

$$a_1 = \frac{5^6}{6^5 \cdot 2} \qquad q = \frac{5^6}{6^5 \cdot 2} \qquad n = n/5$$

$$\frac{6^n \cdot 2^n}{5^n} \cdot \left(\frac{\frac{5^6}{6^5 \cdot 2} \cdot \left(\frac{5^{6n/5}}{6^{5n/5} \cdot 2^{n/5}} - 1 \right)}{\frac{5^6}{6^5 \cdot 2} - 1} \right)$$

Ignorando as constantes:

$$\frac{6^n \cdot 2^n}{5^n} \cdot \left(\frac{5^{6n/5}}{6^n \cdot 2^{n/5}} - 1 \right)$$

Tratando cada termo de modo independente:

$$\frac{6^n \cdot 2^n}{5^n} \cdot \frac{5^{6n/5}}{6^n \cdot 2^{n/5}} = 5^{n/5} \cdot 2^{4n/5} = 80^{n/5}$$

$$\frac{6^n \cdot 2^n}{5^n} = (12/5)^n < 80^{n/5}$$

Da mesma forma, calcularemos o somatório da direita:

$$a_1 = \frac{4^{5n/5}}{5^{4n/5} \cdot 2^{n/5}} \qquad q = \frac{4^5}{5^4 \cdot 2} \qquad n = 4n/5$$

$$\frac{5^n \cdot 2^n}{4^n} \cdot \left(\frac{\frac{4^n}{5^{4n/5} \cdot 2^{n/5}} \cdot \left(\frac{4^{5 \cdot 4n/5}}{5^{4 \cdot 4n/5} \cdot 2^{4n/5}} - 1 \right)}{\frac{4^5}{5^4 \cdot 2} - 1} \right)$$

Ignorando as constantes:

$$\frac{5^n \cdot 2^n}{4^n} \cdot \left(\frac{4^n}{5^{4n/5} \cdot 2^{n/5}} \cdot \left(\frac{4^{4n}}{5^{16n/5} \cdot 2^{4n/5}} - 1 \right) \right)$$

Calculando os termos separadamente:

$$\frac{5^n \cdot 2^n}{4^n} \cdot \frac{4^n}{5^{4n/5} \cdot 2^{n/5}} \cdot \frac{4^{4n}}{5^{16n/5} \cdot 2^{4n/5}} = \frac{4^{4n}}{5^{3n}} < 80^{n/5}$$

$$\frac{5^n \cdot 2^n}{4^n} \cdot \frac{4^n}{5^{4n/5} \cdot 2^{n/5}} = 2^{4n/5} \cdot 5^{n/5} = 80^{n/5}$$

Podemos notar que o valor $80^{n/5}$ é um limite superior para o somatório. Assim sendo, o tempo gasto para gerar todos os subgrafos 4-coloríveis é da ordem de $O^*(80^{n/5}) = O(2.4023^n)$. □

Após, o algoritmo prossegue da mesma forma que a segunda parte do algoritmo de Eppstein, porém somente precisamos considerar os conjuntos independentes maximais de tamanho no máximo $|H|/4$ e utilizando o limite proposto no Teorema 2.10 temos:

$$\begin{aligned}
\sum_{H \subset V(G)} O^*\left(4^{5L-n}5^{n-4L}\right) &= \sum_{H \subset V(G)} O^*\left(4^{5\frac{|H|}{4}-|V\setminus H|}5^{|V\setminus H|-4\frac{|H|}{4}}\right) \\
&= O^*\left(\sum_{i=0}^{n}\binom{n}{i}\left(4^{\frac{5i}{4}-(n-i)}5^{(n-i)-i}\right)\right) \\
&= O^*\left(\sum_{i=0}^{n}\binom{n}{i}\left(\frac{5}{4}\right)^{n-i}\cdot\left(\frac{4^{5/4}}{5}\right)^{i}\right) \\
&= O^*\left(\left(\frac{5}{4}+\frac{4^{5/4}}{5}\right)^{n}\right),
\end{aligned}$$

que corresponde a $O(2.3814^n)$. Assim, percebe-se que o processo de obtenção dos subgrafos maximais 4-coloríveis domina o limite geral de tempo. Portanto, a complexidade de tempo do algoritmo é da ordem de $O(2.4023^n)$. A complexidade de espaço ainda é de $O(2^n)$, pois o procedimento também utiliza um vetor indexado com 2^n posições.

Capítulo 4

Algoritmos baseados em Redução/Contração

Como dito no início do Capítulo 2, existem diferentes maneiras de calcular o número cromático de um grafo. Naquele momento, estávamos interessados somente nas que fundamentavam-se em conjuntos independentes maximais.

No presente capítulo, abordaremos o problema utilizando um processo de redução de grafos. Os algoritmos pertencentes a esta classe estão baseados em uma propriedade inerente ao número cromático de um grafo qualquer, a qual é descrita abaixo e foi apresentada por ZYKOV [46].

Definição 4.1 (Zykov, 1949). *Seja G um grafo não completo. Dados dois vértices não adjacentes x e y em G, os grafos reduzidos de G são os grafos $G+xy$ e $G/x,y$, assim definidos:*

(i) $G+xy$ é obtido de G acrescentando-se a aresta (x,y);

(ii) $G/x,y$ é obtido de G contraindo-se o vértice y em x, isto é, remove-se y e os vértices adjacentes (vizinhos) $N(x)$ de x tornam-se $N(x) \cup N(y)$.

Um exemplo dos grafos reduzidos $G + xy$ e $G/x, y$ pode ser visto na Figura 4.1 abaixo:

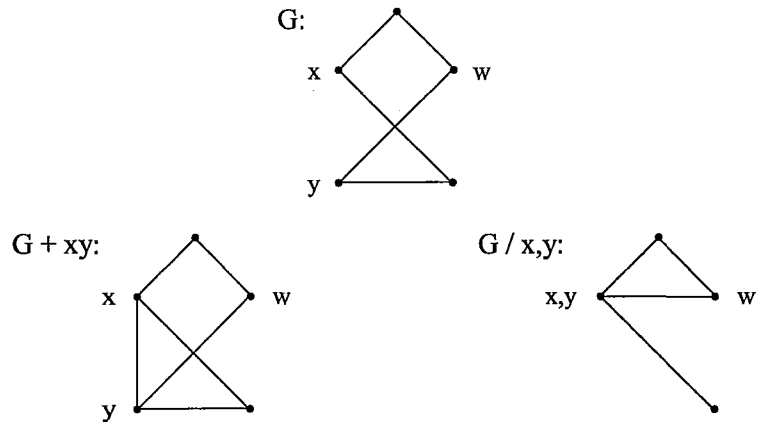

Figura 4.1: Exemplos de grafos reduzidos de G.

Denominamos *árvore de Zykov* para um grafo G, a árvore binária Z, definida da seguinte maneira:

(i) G é a raiz de Z;

(ii) Seja H um nó qualquer de Z. Se H é um grafo completo, então H é uma folha de Z. Caso contrário, seja x e y um par qualquer de vértices não adjacentes de H, H tem por filho esquerdo o grafo reduzido $H + xy$ e por filho direito, o grafo reduzido $H/x, y$.

Para um melhor entendimento, apresentamos na Figura 4.2 uma árvore de Zykov completa.

Note que, sempre que o grafo reduzido $G/x, y$ for gerado, se armazenarmos o rótulo do vértice contraído juntamente com o do vértice resultante, teremos, ao final do processo, que cada folha da árvore representará uma coloração de G, onde cada vértice indicará uma classe de cor.

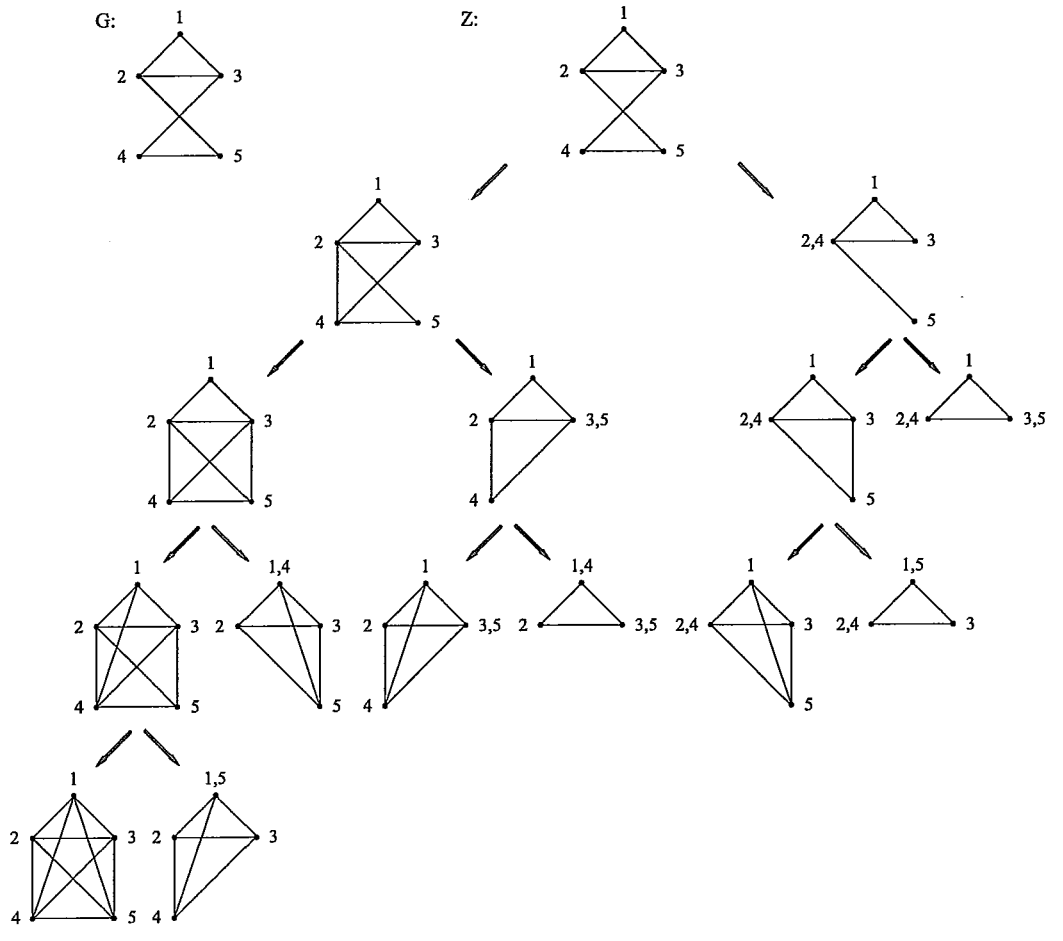

Figura 4.2: A árvore de Zykov para G.

Teorema 4.2 (Zykov, 1949). *Se x e y são dois vértices não adjacentes em G, então $\chi(G) = \min\{\chi(G/x,y), \chi(G+xy)\}$.*

Demonstração. Sabemos que uma coloração (própria) de vértices de um grafo $G = (V, E)$ é um mapeamento $F : V \to \mathbb{N}$, onde vértices adjacentes recebem cores distintas em $\mathbb{N}$; isto é, se $(u, v) \in E$, então $F(u) \neq F(v)$. Seja $C(G)$ o conjunto de colorações (próprias) de G e $|F|$ o número de cores usadas por

$F \in C(G)$, temos:

$$\begin{aligned}\chi(G) &= \min\{|F| : F \in C(G)\} \\ &= \min\{\min\{|F| : F(x) = F(y)\}, \min\{|F| : F(x) \neq F(y)\}\} \\ &= \min\{\chi(G/x,y), \chi(G+xy)\}.\end{aligned}$$

□

Esse teorema pode ser aplicado recursivamente nos nós da árvore de Zykov para um grafo G, obtendo-se $\chi(G) = min\{\chi(H_1), \chi(H_2), \ldots, \chi(H_k)\}$, onde $H_1, H_2, \ldots, H_k$ representam os grafos completos, localizados nas folhas da árvore, cujo número cromático é de obtenção imediata (é igual ao número de vértices do grafo H).

4.1 O algoritmo de Zykov

Denominaremos de algoritmo de Zykov, o pseudocódigo apresentado no Algoritmo 4.1 o qual é uma aplicação imediata do Teorema 4.2.

Ao analisarmos tal procedimento, notamos que este sempre constrói a árvore de Zykov completa, tendo, portanto, que pesquisar os dois filhos de cada nó não-folha. Assim, como mostrado no exemplo da Figura 4.2, percebe-se que a árvore de Zykov não é única (pois depende diretamente do par de vértices não adjacentes escolhido). Entretanto, sua altura será igual ao número de arestas faltantes para que o grafo inicial G torne-se completo, ou seja, $O(\bar{m})$, onde $\bar{m}$ é o número de arestas do grafo complementar de G. Sabendo que a árvore é binária, temos como limite superior para o número total de nós:

$$\sum_{l=0}^{\bar{m}} 2^l = \frac{2^{\bar{m}+1}-1}{2-1} = 2^{\bar{m}+1} - 1$$

Algoritmo 4.1: Número Cromático (Zykov, 1949).

```
1  REDUZA (H,p)      /* grafo H e p seu número de vértices */
2  se (k > p) então  k = p;
3  se (H não é um grafo completo) então
4      sejam x e y dois vértices não adjacentes de H;
5      determine (H + xy) e (H/x,y);
6      REDUZA ((H + xy), p);
7      REDUZA ((H/xy), p - 1);
8  fim
9  k = n;                /* k é um limite superior para χ(G)  */
10 REDUZA (G, n);
```

No pior caso, o grafo complemento de G será o grafo completo, sabemos que o número de arestas de um grafo completo pode ser determindo por: $m = n(n-1)/2$, ou seja, é da ordem de $O(n^2)$. Sendo assim, e levando em consideração que este procedimento gasta um tempo proporcional ao número total de nós da árvore, temos que a complexidade de tempo é da ordem de $O(2^{n^2})$.

Para medirmos a complexidade de espaço, note que o algoritmo constrói a árvore Z utilizando o percurso em profundidade, desta forma basta calcularmos o espaço utilizado pelo ramo mais à esquerda. A altura desse ramo corresponderá a altura máxima da árvore, a qual é da ordem de $O(n^2)$. Em cada nível necessitamos armazenar todo o grafo, logo o espaço exigido pelo método é de $O(n^2(n+m)) = O(n^2m)$.

Antes de finalizarmos este capítulo, gostaríamos de ressaltar que, apesar da simplicidade e elegância do processo, não foram encontrados outros

métodos exatos para a determinação do número cromático de um grafo G qualquer, que utilizassem os conceitos de redução.

Entretanto, diferentemente dos exatos, vários métodos heurísticos foram apresentados [15, 17, 29], os quais não serão abordados pois não fazem parte dos objetivos propostos por esta dissertação.

Capítulo 5

Algoritmos para 3-coloração

A determinação do número cromático em um grafo talvez seja o mais importante problema envolvendo coloração de vértices. Entretanto, muitos esforços nesta área concentram-se também na solução da 3-coloração, isto é, verificar se um grafo pode ser colorido com, no máximo, três cores. Como fruto desses esforços, nota-se que é nesta classe de problemas que se encontram os mais significativos melhoramentos entre os diversos métodos. Algoritmos exatos que resolvem este tipo de problema são o assunto do presente capítulo.

Iremos começar apresentando o algoritmo de LAWLER [32], o qual foi o pioneiro na demonstração de um método não trivial para a resolução da 3-coloração. O primeiro a obter melhores resultados que Lawler foi, segundo os textos pesquisados [2, 3, 11, 18], o alemão Ingo Schiermeyer, que em 1994 desenvolveu um algoritmo com complexidade de tempo $O^*(1.415^n)$. Tal procedimento não será descrito neste trabalho, uma vez que, apesar de todo empenho, não conseguimos ter acesso ao artigo.

Na seção 5.2 temos o algoritmo de BEIGEL e EPPSTEIN [2, 3, 18] cuja característica principal é a não utilização do conceito de conjuntos indepen-

dentes maximais. Vale ressaltar ainda que, este é, dentre os métodos (exatos) existentes na literatura, o que possui o menor limite de tempo.

Por fim, propomos na seção 5.3, um novo método exato para a 3-coloração, este, diferentemente dos demais, emprega as idéias propostas por ZYKOV [46] em seu algoritmo para o número cromático.

5.1 O procedimento de Lawler

O processo mais intuitivo para se testar a 3-colorabilidade em um grafo utilizando o conceito de conjuntos independentes maximais, talvez seja, o proposto por LAWLER [32] em um artigo de 1976, cuja idéia pode ser explicitada da seguinte maneira:

(1) gerar todos os conjuntos independentes maximais do grafo G;

(2) para cada conjunto independente maximal S gerado, testar se o subgrafo induzido $G\backslash S$ é bipartido;

(3) caso algum subgrafo $G\backslash S$ seja bipartido, então G pode ser colorido com três cores.

Segundo Moon e Moser [35] o número de conjuntos independentes maximais de um grafo com n vértices é no máximo $3^{n/3}$. Existe um algoritmo [43] que gera todos os conjuntos independentes maximais de um grafo em tempo da ordem de $O(mnK)$, onde K é o número de conjuntos independentes maximais de G.

E, sabendo que o teste para verificar se um grafo é bipartido pode ser feito em tempo linear de $O(m+n)$, temos que a complexidade de tempo do processo é de $O((m+n)mn3^{n/3})$, que resulta em $O^*(3^{n/3}) \approx O^*(1.4422^n)$.

Já a complexidade de espaço é da ordem de $O(n+m)$, uma vez que em cada iteração é necessário armazenar somente o CIM S e o subgrafo $G \backslash S$.

5.2 O procedimento de Beigel e Eppstein

Beigel e Eppstein (1995, 2005) [2, 3, 18] obtêm uma menor complexidade para o problema da 3-coloração partindo da seguinte idéia: tenta-se resolver o problema parcialmente, limitando cada vértice a somente duas de suas três cores possíveis. Em seguida, verifica-se se a solução parcial pode ser estendida para uma coloração completa.

Para obter melhores resultados, efetua-se uma atribuição *a priori* de cores a um ou mais vértices do grafo. Com isso, consegue-se restringir as possíveis cores dos vértices restantes (se escolhermos uma cor para um determinado vértice, isto irá limitar as cores de vários vizinhos ao mesmo tempo).

Porém, ao realizarmos reduções locais como citado acima, podemos obter uma situação na qual alguns vértices ainda não coloridos, estejam cercados por vizinhos parcialmente coloridos, e portanto, esta solução parcial não conduz a uma solução ótima. Para evitar problemas dessa natureza, os autores generalizam o problema de coloração em um *problema de satisfatibilidade de restrições*, que serão aqui denominados de CSP (*Constraint Satisfaction Problems*) [31, 41].

5.2.1 Problemas de satisfatibilidade de restrições - CSP

Um problema do tipo (a, b)-CSP é assim definido: seja uma coleção de n vértices (variáveis), onde em cada um pode-se atribuir uma das a diferentes cores (estados). Porém, certas combinações de cores não são permitidas, ou seja, temos também, uma coleção de m restrições, onde em cada uma

proíbe-se uma coloração para alguma b-tupla de vértices, logo, uma restrição é formada por b pares do tipo (v, c) onde v representa um vértice e c uma cor. Uma restrição é dita *satisfeita* por uma coloração de vértices se nem todos os vértices da tupla são coloridos da maneira especificada pela restrição. Ou seja, uma coloração de vértices resolve um problema (a, b)-CSP se e somente se para cada restrição existe, pelo menos, um par (v, c) na restrição que não apareça na coloração. Assim, o problema da 3-coloração é um caso do tipo $(3, 2)$-CSP na qual os vértices podem ter apenas três cores e as restrições não permitem que vértices adjacentes tenham a mesma cor.

Qualquer instância do tipo $(d, 2)$-CSP pode ser representada graficamente: cada um dos n vértices será uma espécie de conjunto e conterá d possíveis cores, desenharemos arestas ligando dois conjuntos se os vértices possuírem cores incompatíveis. Note que esta estrutura não é, propriamente, um grafo, pois as arestas conectam as cores dentro dos vértices e não os vértices em si.

Uma representação gráfica do problema da 3-coloração transformado para um do tipo $(3, 2)$-CSP, pode ser obtida da seguinte forma: mantém-se os vértices originais do grafo e suas possíveis cores, após, adicionam-se três restrições por aresta, de modo a garantir que vértices adjacentes terão cores distintas. Um exemplo desta transformação pode ser visto na Figura 5.1.

Devido à natureza do problema, os autores mostram ainda que, através de simplificações, podemos pré-processar uma instância CSP, a fim de, após a execução dessas operações, obter uma instância CSP reduzida (menor número de vértices) equivalente à instância original.

O lema a seguir é um exemplo deste tipo de simplificação, ele provê uma maneira de remover vértices de uma instância $(a, 2)$-CSP, para os quais poucas cores estão disponíveis.

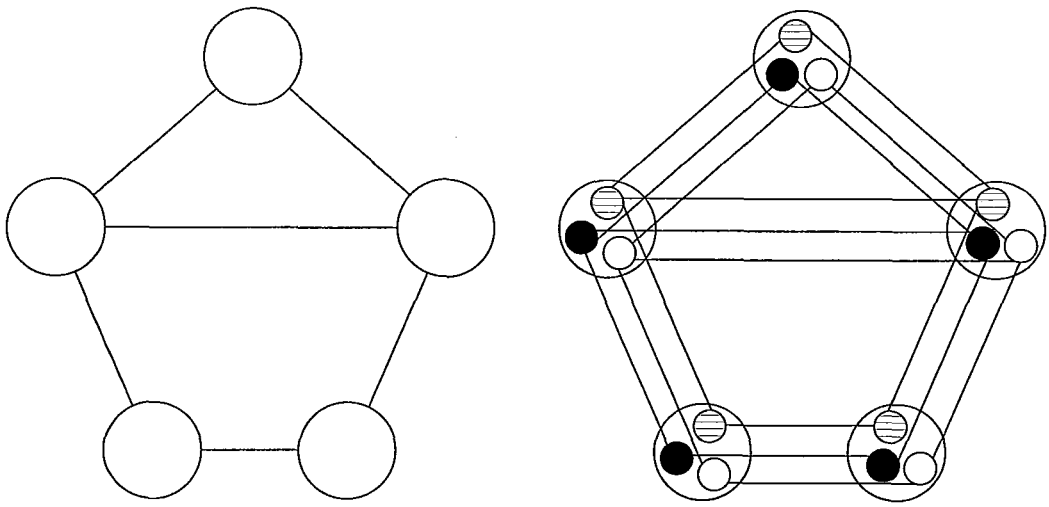

Figura 5.1: Transformação de um problema de 3-coloração para um $(3, 2)$-CSP.

Lema 5.1 (Beigel e Eppstein, 1995). *Seja v um vértice em uma instância $(a, 2)$-CSP, tal que somente duas de suas a cores estejam permitidas para v. Então podemos encontrar uma instância $(a, 2)$-CSP equivalente com um vértice a menos.*

Demonstração. Sejam R e P as duas cores permitidas para um determinado vértice v. Adicionamos restrições do tipo $((u, A), (w, B))$ toda vez que (u, A) aparecer em uma restrição juntamente com (v, R) e, para todo (w, B) que aparecer em uma outra restrição juntamente com (v, P). Estas novas restrições justificam-se pois, se (u, A) e (w, B) estiverem presentes em uma coloração, v não poderá assumir nenhuma de suas duas cores permitidas. Entretanto, se todas as novas restrições forem satisfeitas, uma das duas possíveis cores de v lhe poderá ser atribuída.

Portanto, podemos encontrar um problema equivalente, menor que o original (sem a presença do vértice v), como visto na Figura 5.2. □

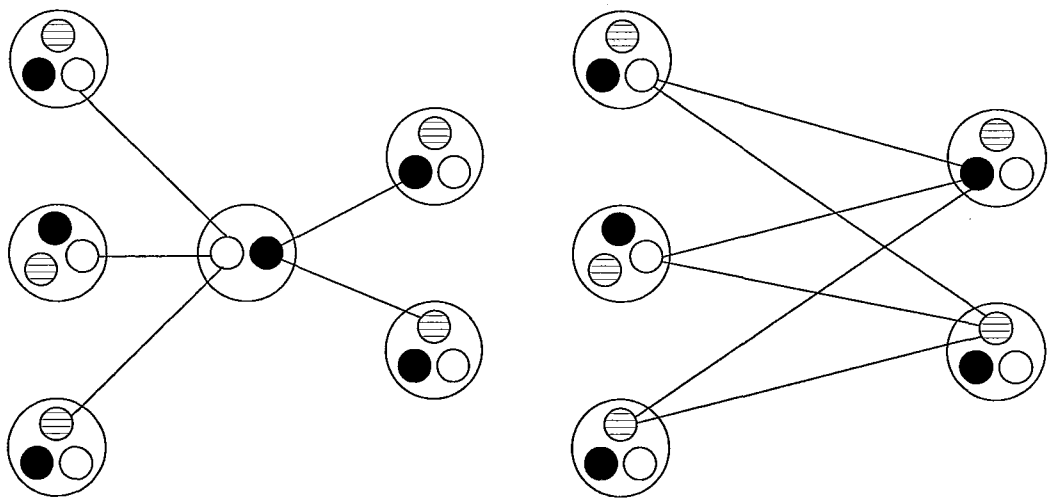

Figura 5.2: Instância $(3,2)$-CSP e instância reduzida após aplicação do Lema 5.1.

Devido a esta habilidade em eliminar vértices parcialmente coloridos, consegue-se resolver o $(3,2)$-CSP sem sair de boas configurações locais, uma vez que evita-se que vértices parcialmente coloridos consigam cercar vizinhos ainda não coloridos.

Além da simplificação vista acima, os autores descrevem outras situações nas quais o número de vértices em uma instância CSP pode ser reduzido sem alterar a equivalência entre as instâncias. Maiores detalhes sobre as outras simplificações podem ser encontrados em [3].

5.2.2 Algoritmo para CSP

Uma vez transformado o problema de 3-coloração de vértices em um do tipo CSP, torna-se relevante, a construção de um procedimento que resolva, eficientemente, essa classe de problemas.

Assim sendo, Beigel e Eppstein desenvolveram um algoritmo *branch-and-reduce* cuja idéia é localizar em uma instância CSP reduzida, um conjunto de

configurações locais, onde para cada configuração, todos os possíveis casos são analisados, em outras palavras, todas as possíveis colorações para os vértices da configuração são testadas. Deste modo, consegue-se ramificar a instância original I em poucas instâncias I_i de menor tamanho, e que, existirá uma solução para I se e somente se, pelo menos uma instância I_i for resolvível.

Em um primeiro artigo, publicado em 1995, eles mostram como encontrar uma solução para o $(3,2)$-CSP em um tempo da ordem de $O^*(1.38028^n)$. Esta medida é obtida através de uma complicada análise de casos e, o pior deles, é descrito a seguir: encontre um par (v, R) que esteja envolvido em restrições com pelo menos três outros vértices. Se atribuirmos para v a cor R, todos os outros vértices envolvidos em restrições com o par (v, R) passarão a ter somente duas cores disponíveis. Aplicando o Lema 5.1 para cada um deles, teremos um problema equivalente com, pelo menos, quatro vértices a menos (v já recebeu uma cor). Por outro lado, se v receber qualquer uma das outras duas cores disponíveis, somente v será removido.

O tempo total para este caso é analisado pela seguinte relação de recorrência: $T(n) \leq T(n-1) + T(n-4)$, a qual pode ser determinada em $O^*(1.38028^n)$.

Em 2001, os autores propuseram um método que possibilita encontrar uma solução para o $(3,2)$-CSP utilizando um tempo de $O^*(1.36443^n)$. Para obter tal melhoria, Beigel e Eppstein estendem a solução para instâncias do tipo $(4,2)$-CSP, ou seja, este novo método consegue resolver instâncias onde cada vértice (variável) possui uma lista de até quatro possíveis cores.

Qualquer instância $(4,2)$-CSP pode ser transformada em uma do tipo $(3,2)$-CSP: divide-se cada variável "4-colorível" em duas variáveis "3-coloríveis", tal que cada uma delas contenha duas das quatros cores originais.

Após, insere-se uma restrição ligando as duas variáveis através da terceira cor (Figura 5.3). Da mesma forma, sempre que uma configuração como esta estiver presente em uma instância $(3,2)$-CSP, podemos transformá-la em uma $(4,2)$-CSP.

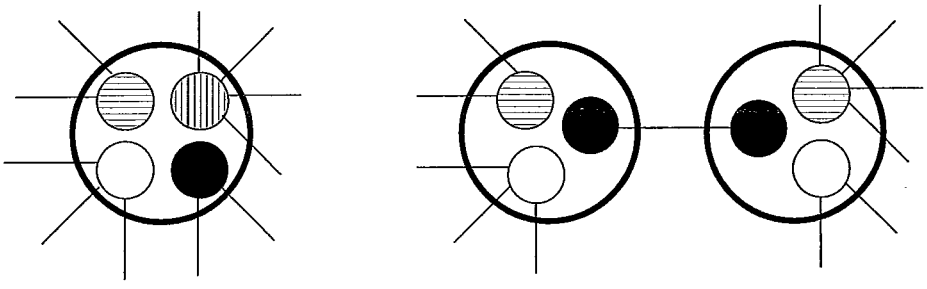

Figura 5.3: Transformação de uma instância $(4,2)$-CSP para uma $(3,2)$-CSP.

Seja n_3 o número de variáveis 3-coloríveis, e n_4 o número de variáveis que podem assumir quatro diferentes cores, então o tamanho n de um problema $(4,2)$-CSP é $n_3 + n_4$. Entretanto, vimos anteriormente que, uma variável 4-colorível pode ser substituída por duas variáveis 3-coloríveis. Logo, podemos admitir que $n = n_3 + 2n_4$.

Levando em consideração que, problemas com muitas variáveis 4-coloríveis são, provavelmente, mais difíceis do que problemas com poucas variáveis, os autores adotam o último como uma maneira natural de medir o tamanho, porém utilizam $n = n_3 + (2 - \epsilon)n_4$, onde $\epsilon \approx 0.095543$. O modo como Beigel e Eppstein determinam o valor da constante ϵ não será abordado neste trabalho, contudo, é suficiente saber que este valor minimiza o tempo em cada caso da análise, uma vez que estes tempos dependem diretamente da escolha de ϵ.

Perceba que o tamanho de uma instância $(3,2)$-CSP permanece igual ao seu número de variáveis, assim, qualquer melhoramento na complexidade para o algoritmo $(4,2)$-CSP, implicará diretamente no $(3,2)$-CSP.

Assim como no algoritmo proposto em 1995, este também irá procurar por um conjunto de configurações locais e para cada configuração encontrada substitui-se a instância atual por outras de menor tamanho. Ainda, também apresenta uma complexa e extensa análise de casos, a qual, é bastante similar à efetuada na resolução do $(3,2)$-CSP. Prosseguiremos sem apresentar tal análise, uma vez que esta não faz parte do assunto abordado no presente trabalho. Para maiores detalhes, consulte [3, 18].

Como dito anteriormente, este método possibilita a utilização de instâncias $(4,2)$-CSP. E, com esta nova abordagem, os autores mostram que podemos interromper a busca pelas configurações locais toda vez que a instância for suficientemente simples para ser resolvida por um algoritmo polinomial, no método antecessor isso não ocorria: a análise dos casos continuava até a instância ser determinada (diretamente) como resolvível ou não.

Teorema 5.2 (Beigel e Eppstein, 2001). *Qualquer instância* $(3,2)$*-CSP pode ser resolvida em tempo da ordem de* $O^*(1.36443^n)$.

Demonstração. Todo algoritmo que utiliza a análise de casos (*branch-and-reduce*) define, implicitamente, uma árvore de busca. No caso atual, cada nó representará uma instância $(3,2)$-CSP.

Assim sendo, iremos explorar a árvore utilizando o percurso em profundidade. Em cada passo, examinaremos a instância corrente I na tentativa de encontrar alguma configuração local que pode ser aplicada e, recursivamente, repetimos o processo para cada "sub-instância" I_i. Se encontrarmos uma que seja suficientemente simples, aplica-se um algoritmo polinomial para verificar se a mesma é resolvível ou não. Caso afirmativo, o processo pára e uma solução é encontrada. Caso contrário, retorna-se para o mais recente ponto da recursão e continua-se a busca com a próxima instância.

Os autores mostram que para o valor ótimo de ϵ, três configurações apresentam a complexidade de pior caso. Estas, serão aqui representadas através de seus vetores de ramificação, são eles: $t = (3-\epsilon, 4-\epsilon, 4-\epsilon) = (1+\epsilon, 4) = (4, 4, 5, 5,)$, os quais podem ser resolvidos em $O^*(1.36443^n)$. □

Vimos que a 3-coloração é um caso especial do $(3, 2)$-CSP. Portanto, se transformarmos o problema de 3-coloração diretamente para um do tipo $(3, 2)$-CSP e aplicarmos o algoritmo para CSP visto acima, teremos uma complexidade de tempo de $O^*(1.36443^n)$.

Como dito no início da seção, para reduzir este tempo ainda mais, Beigel e Eppstein utilizam a seguinte idéia: encontre um pequeno conjunto $N \subset V(G)$ de vértices, o qual possui um grande conjunto M de vizinhos. Escolha uma das $3^{|N|}$ colorações possíveis para os vértices em N. Para cada coloração, transforme o conjunto de vértices restante em uma instância do tipo $(3, 2)$-CSP. Note que os vértices em N já estão coloridos e não precisam ser incluídos na instância $(3, 2)$-CSP. Os vértices em M têm agora um vizinho colorido (vértices em N), logo para cada vértice em M restam, no máximo, duas possíveis cores; por esta razão, estes podem ser removidos da instância $(3, 2)$-CSP através do Lema 5.1.

A instância $(3, 2)$-CSP resultante terá $k = |V(G) - N - M|$ vértices e pode ser resolvida através do Teorema 5.2 em tempo de $O^*(1.36443^k)$. O tempo total é, portanto, $O^*(3^{|N|}1.36443^k)$. Como k depende diretamente de N, se escolhermos N de uma forma adequada, podemos obter um valor menor que $O^*(1.36443^n)$.

Para encontrar tal conjunto, procederemos da seguinte maneira: inicialmente, iremos assumir que todos os vértices em G possuem grau maior ou igual a três, uma vez que, vértices com grau inferior a três podem ser removidos sem alterar a 3-colorabilidade de G.

Em seguida, os autores mostram que se G possui algum ciclo formado apenas por vértices de grau três, então G pode ser substituído por instâncias menores, e que o tempo gasto para efetuar tal transformação pode ser representado pelo vetor de ramificação $t = (5, 6, 7, 8) \approx 1.2433$.

Após aplicada a redução acima, os vértices de grau três restantes (se existirem), irão compor uma (várias) árvore(s). Se uma árvore formada por estes vértices tiver oito ou mais elementos, então uma nova redução pode ser efetuada em G. Devido a esta simplificação, cujo tempo corresponderá a $t = (2, 5, 6) \approx 1.3247$, podemos assumir que os vértices de grau três irão formar árvores de, no máximo, sete elementos, e que o grafo contém muitos vértices de grau alto.

Denominaremos *floresta densa*, a floresta F contida em uma instância do grafo (já sofreu reduções), cujos vértices internos possuem grau quatro ou mais, ou seja, somente as folhas de F poderão ter graus menores que quatro (para um exemplo, considere os três níveis superiores da Figura 5.4).

Uma floresta densa F será maximal quando todos os nós internos forem adjacentes somente a vértices pertencentes à F, nenhuma folha deverá ter três ou mais vizinhos fora de F, e não existir um vértice não pertencente a F que tenha quatro ou mais vizinhos fora de F.

Uma vez definida, Beigel e Eppstein mostram que, uma floresta densa maximal F consegue cobrir uma parcela dos vértices em uma instância do grafo. Tal parcela, cujo valor corresponde a $|G \backslash F| \leq 20r/3$, onde r representa o número de folhas de F, é obtida utilizando, além de outras informações, o fato de que as árvores formadas somente por vértices de grau três terão no máximo sete elementos. Em outras palavras, esta parcela indica que o número de vértices pertencentes a G e não pertencentes a F é de, no máximo, $20r/3$.

O conjunto N será então formado por todos os nós internos de F, além destes, alguns vértices pertencentes ao subgrafo $|G\backslash F|$ restante também poderão ser incluídos em N. A fim de localizar tais vértices, constrói-se uma floresta J, que cobre os vértices de $G\backslash F$, a qual é constituída por árvores de altura dois, com três ramificações e no máximo cinco folhas. A construção de J envolve técnicas de fluxo em rede (*network flow*) que serão omitidas por não fazerem parte do escopo deste trabalho. Maiores informações podem ser obtidas em [3].

A inclusão (em N) de um vértice pertencente a uma árvore de J, irá depender da configuração apresentada por cada árvore. Se a árvore possui exatamente cinco folhas então, a raiz terá três filhos e, dois destes (x e y) possuirão dois filhos (folhas) cada, enquanto que o filho restante da raiz, terá somente um, totalizando as cinco folhas.

Neste caso, x e y serão incluídos em N, ou seja, iremos atribuir cores para estes vértices, os vizinhos de x e y serão eliminados (Lema 5.1) e os vértices restantes farão parte da instância $(3,2)$-CSP.

Por outro lado, se a árvore apresenta menos que cinco folhas, então incluiremos em N somente a raiz da mesma.

Assim sendo, se considerarmos apenas o tempo gasto para colorir uma árvore de J, obteremos, no pior caso, um valor de $(3^1 \cdot 1.36443^3)^{1/7} \approx 1.3366$, o qual ocorre quando o número de folhas é três.

Após vistas estas definições, estamos aptos para descrever o algoritmo de Beigel e Eppstein para a 3-coloração:

a) aplique as duas reduções iniciais em G (eliminar ciclos de vértices com grau três e remover árvores formadas por vértices de grau três com mais de sete elementos);

b) encontre em G uma floresta densa maximal F;

c) encontre uma floresta J, formada por árvores de altura dois, que cubra os vértices de $G \backslash F$;

d) o conjunto N (conjunto de vértices que receberá uma cor) será formado pelos vértices internos de F e, como dito acima, por alguns vértices pertencentes à árvores de J;

e) vértices adjacentes a N estarão limitados a somente duas cores e serão retirados do problema através da aplicação do Lema 5.1;

f) os vértices restantes formarão a instância $(3,2)$-CSP e serão coloridos pelo algoritmo para CSP descrito no início da seção.

Para facilitar o entendimento, considere a Figura 5.4 abaixo: em p temos a quantidade de vértices que são raízes em F; em q o número de vértices internos não-raiz; em r a quantidade de folhas de F; s contém os vértices adjacentes às folhas de F e em t estão representados os vértices restantes, os quais possuirão, no máximo, grau três e farão parte de alguma árvore em J. Portanto, o tempo total do algoritmo é no máximo de $3^p \cdot 2^q \cdot 1.36443^s \cdot (3 \cdot 1.36443^3)^{t/7}$.

Podemos notar que este limite está sujeito às seguintes restrições:

$$p, q, r, s, t \geq 0$$
$$p + q + r + s + t = n$$
$$4p + 2q \leq r \text{ (definição de floresta densa)}$$
$$2r \geq s \text{ (maximalidade de } F)$$
$$20r/3 \geq s + t \text{ (quantidade de vértices cobertos por } F)$$

O pior caso ocorre quando: $s + t = 20r/3$, $s = 2r$, $t = 14r/3$, $4p + 2q = r$, $p = 0$ e $r = 2q$.

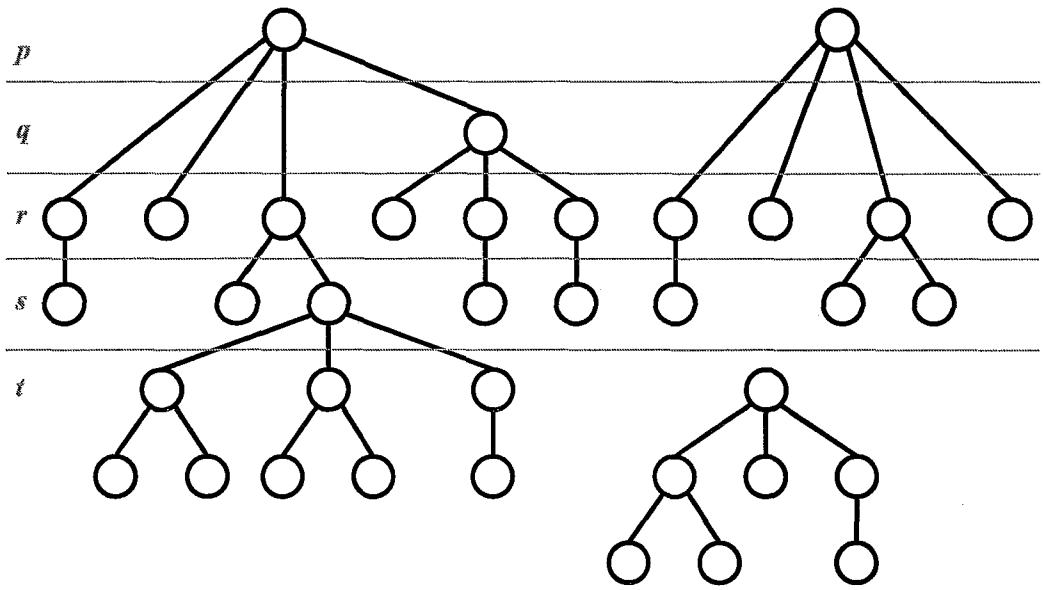

Figura 5.4: Partição dos vértices em grupos.

Colocando a segunda restrição em função de q temos:

$$p + q + r + s + t = n$$
$$0 + q + 2q + 4q + 28q/3 = n$$
$$49q/3 = n \quad \Rightarrow \quad q = 3n/49$$

E, substituindo os valores de pior caso no limite:

$$3^p \cdot 2^q \cdot 1.36443^s \cdot (3 \cdot 1.36443^3)^{t/7}$$
$$3^0 \cdot 2^q \cdot 1.36443^{4q} \cdot (3 \cdot 1.36443^3)^{4q/3}$$
$$2^{3n/49} \cdot 1.36443^{12n/49} \cdot (3 \cdot 1.36443^3)^{12n/147}$$

Que resulta em uma complexidade de tempo da ordem de $O(1.3289^n)$. Esse tempo é, atualmente, o menor valor existente na literatura para a solução do problema de 3-coloração em um grafo.

Para finalizar, merecem destaque, entre tantas outras, duas características: a primeira, diz respeito à forma como os autores abordaram o problema, pro-

pondo uma solução que não utiliza, em nenhum momento, o conceito (tradicional) de conjuntos independentes maximais. A segunda peculiaridade ficou marcada pela utilização de variadas técnicas e idéias, as quais, apesar de aumentarem a dificuldade no entendimento do processo, foram as principais responsáveis pela obtenção do resultado.

5.3 Um novo algoritmo

Apresentaremos a seguir um novo algoritmo exato para a determinação da 3-coloração em um grafo G qualquer. Este procedimento foi desenvolvido por FARIA [20], em 2005, cuja motivação ao construí-lo pode ser atribuída, em parte, à pesquisa que efetuamos para a confecção do presente trabalho.

A diferença entre esse algoritmo e os demais vistos no capítulo atual é que ele utiliza as idéias propostas por ZYKOV [46], as quais podem ser resumidas no Teorema 4.2.

Inicialmente, enunciaremos alguns teoremas básicos, a fim de mostrar que para um grafo G não ser 3-colorível, é necessário, que existam vértices de grau maior ou igual a três.

Teorema 5.3. *: Se um grafo G qualquer é k-crítico, então o grau mínimo (δ) de G é $\delta \geq k-1$.*

Demonstração. Seja G um grafo k-crítico e suponha (por contradição) que $\delta < k-1$. Seja v um vértice de G tal que o grau $d(v)$ de v seja igual a δ, ou seja, v é o vértice de menor grau.

Como G é crítico, temos que $\chi(G-v) < \chi(G) = k$, logo $G-v$ é um subgrafo $(k-1)$-colorível.

Seja $(V_1, V_2, \ldots, V_{k-1})$ uma $(k-1)$-coloração de $G-v$. Como $d(v) = \delta$, então v é adjacente a $\delta < k-1$ vértices, logo existe um conjunto V_j

cujos vértices não são adjacentes a v. Podemos, então, considerar a seguinte coloração para G: $(V_1, V_2, \ldots, V_j \cup \{v\}, \ldots, V_{k-1})$, ou seja, obtemos uma $(k-1)$-coloração para G. Uma contradição, pois $\chi(G) = k$. □

Corolário 5.4. *: Todo grafo G, k-cromático, possui pelo menos k vértices de grau $d(v) \geq k-1$.*

Demonstração. Se G é um grafo k-cromático, então G possui um subgrafo H que é k-crítico. Pelo Teorema 5.3, temos que todo vértice v de H possui $d_H(v) \geq k-1$, logo, o vértice v de G também terá $d_G(v) \geq k-1$. Como H é k-cromático, temos claramente que H tem mais do que k vértices. □

Corolário 5.5. *: Para qualquer grafo G, $\chi(G) \leq \Delta + 1$, onde Δ é o valor do grau máximo de G.*

Demonstração. Por contradição, suponha que $\chi(G) > \Delta + 1$, como G é k-cromático, temos $\chi(G) = k$, logo $k > \Delta + 1$. Pelo corolário 5.4, G possui vértices com grau maior ou igual a $k-1$.

Seja v um desses vértices, temos:

$$\begin{aligned} d(v) &\geq k-1 \\ d(v) &> \Delta+1-1 \\ d(v) &> \Delta \end{aligned}$$

Um absurdo, pois não pode existir um vértice com grau maior que Δ. □

Suponha que $\Delta = 2$, então, pelo Corolário 5.5, temos que $\chi(G) \leq 2+1$, logo o número cromático será menor ou igual a três. Portanto, não existe um grafo k-cromático $(k > 3)$ com grau máximo igual a dois.

Assim, um teste inicial para verificar a 3-coloração é aplicar o procedimento somente para grafos com $\Delta \geq 3$. Pois acabamos de ver que para grafos com $\Delta < 3$, todos podem ser coloridos com três cores.

Outro teste trivial que pode ser aplicado ao grafo inicial é a verificação da bipartição, ou seja, testar se G é um grafo bipartido. No caso afirmativo, G é 2-colorível, se negativo continua-se o processo. Note que o tempo gasto para efetuar estes dois testes iniciais não implicará em aumento significativo no tempo total, uma vez que ambos podem ser executados em tempo linear.

Como dito inicialmente, este algoritmo é baseado nas idéias de Zykov, assim sendo, ele também constrói uma árvore do tipo Z, porém, diferentemente do método original, esta construção obedecerá à algumas operações, as quais serão vistas ao longo desta seção.

Lembremos que no método original, o número cromático é determinado pela análise dos nós folha. Adaptando para o problema da 3-coloração, temos que a condição de parada é a existência de uma folha (grafo completo) com três vértices. Se existir uma folha com tal propriedade, o grafo é 3-colorível. Caso contrário, ou seja, todas as folhas da árvore possuem quatro vértices (ou mais), o grafo não é 3-colorível.

Sabendo que a complexidade do processo depende diretamente da altura da árvore de Zykov, buscaremos construir a mesma, de um modo que esta possua o menor número de níveis possível.

Para tal, sugerimos as seguintes três operações, que poderão ser aplicadas aos nós da árvore, a fim de diminuir a altura da mesma.

Como visto anteriormente, no passo inicial do algoritmo verificamos o grau máximo de G. Se $\Delta < 3$, podemos afirmar que G é 3-colorível. Caso contrário, G terá, pelo menos, um vértice com grau maior ou igual a três.

Em um segundo momento, testamos se G é um grafo que possui triângulos. Isso pode ser realizado em tempo de $O(nm)$, ou ainda em $O(n^{2.37})$, através do método de multiplicação de matrizes booleanas. Caso G não possua nenhum triângulo temos:

Operação semi-diamante: como G não possui triângulos, não é bipartido e $\Delta \geq 3$, então o subgrafo H apresentado na Figura 5.5 certamente ocorrerá em G.

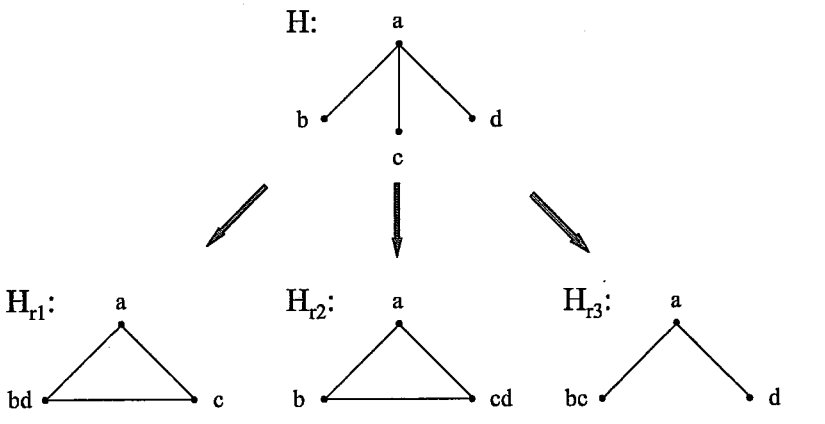

Figura 5.5: A operação semi-diamante.

Como podemos ver na Figura 5.5, a operação semi-diamante consiste em: após localizado um subgrafo como H, criam-se três filhos de H, onde em dois deles H_{r1} e H_{r2}, força-se o aparecimento de um triângulo (clique de tamanho três), enquanto que o terceiro filho H_{r3} constitui-se em uma contração simples.

Vale ressaltar que, os filhos H_{r1}, H_{r2} e H_{r3} de H, ao serem formados, preservam a vizinhança dos vértices envolvidos na operação semi-diamante (ainda, o mesmo também ocorrerá nos subgrafos resultantes para as demais operações).

A Figura 5.6 mostra um comparativo entre um processo utilizando a operação semi-diamante e outro que utiliza as operações tradicionais de Zykov.

Ao analisarmos a figura comparativa, facilmente percebe-se a eficiência da operação semi-diamante em relação às originais, uma vez que, para pro-

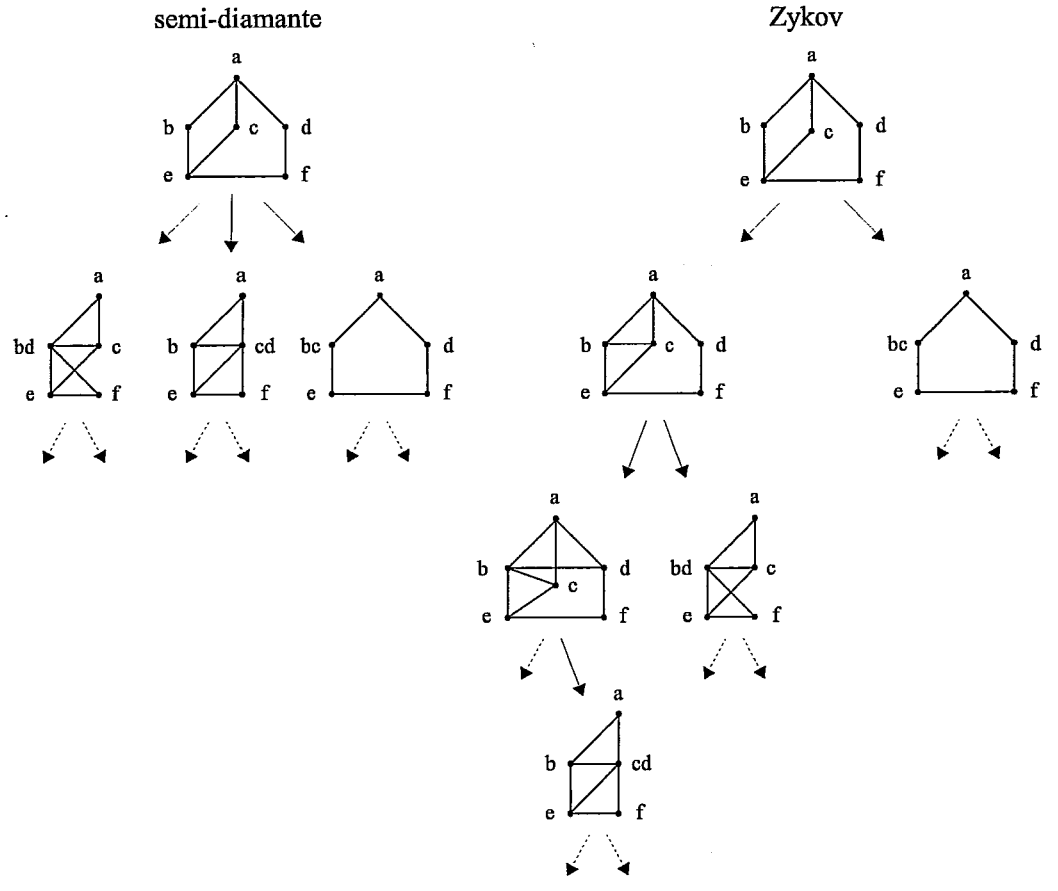

Figura 5.6: Comparação entre a operação semi-diamante e operações de Zykov.

duzir os mesmos nós, o processo de Zykov utiliza dois níveis a mais.

Acabamos de ver uma operação que pode ser aplicada aos nós da árvore caso o grafo G não possua triângulo. Porém, se G possuir um triângulo, teremos então, duas opções:

Operação diamante: neste caso, em G (ou qualquer outro subgrafo H) teremos uma estrutura como a vista na Figura 5.7.

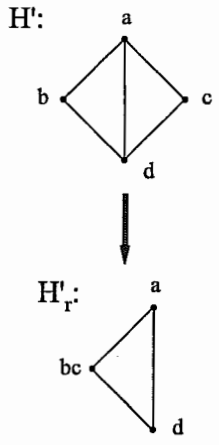

Figura 5.7: A operação diamante.

Perceba que nesta operação o subgrafo que irá corresponder ao nó pai possuirá apenas um único filho. E que, neste filho, a clique de tamanho três é mantida.

Assim como na operação anterior, na Figura 5.8 temos um comparativo entre a operação em questão e a utilizada no processo original.

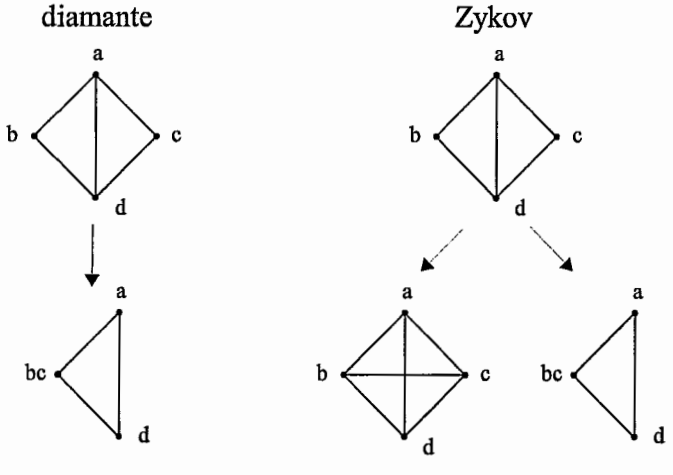

Figura 5.8: Comparação entre a operação diamante e operações de Zykov.

A operação diamante trata-se apenas de uma poda simples nos galhos da árvore, de modo a evitar os ramos que produzirão cliques de tamanho maior que três. Assim sendo, ela não diminui os níveis da árvore, mas sim, a torna mais simples e com menos ramificações.

Operação duplo K_3: supondo que G seja conexo, se a estrutura exibida na operação anterior não ocorrer em G, então, um subgrafo H'' como o da Figura 5.9 certamente existirá.

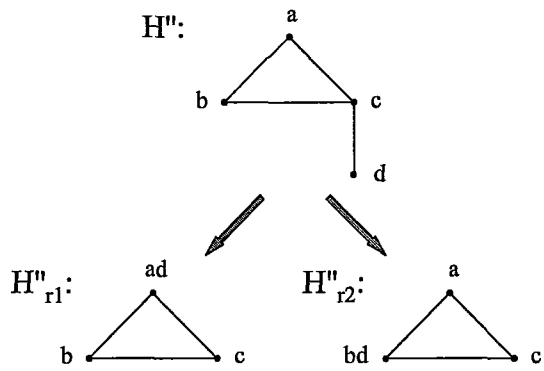

Figura 5.9: A operação duplo K_3.

Note que na operação duplo K_3 temos que o nó pai possuirá dois filhos, os quais, assim como nos casos anteriores, conterão um grafo completo de tamanho três.

Um comparativo entre a presente operação e a tradicional, proposta por Zykov, pode ser visto na Figura 5.10.

Ao fazermos a comparação entre a operação duplo K_3 e Zykov, vemos que com a primeira têm-se o ganho de um nível em relação à segunda.

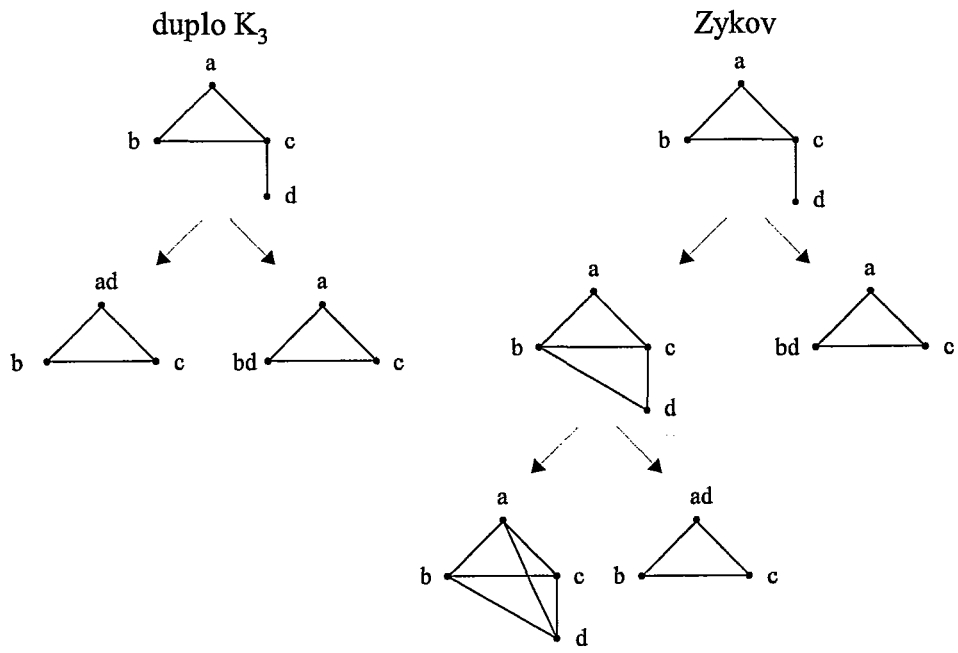

Figura 5.10: Comparação entre a operação duplo K_3 e operações de Zykov.

Apresentamos, através do Algoritmo 5.1, o pseudocódigo que representa as idéias propostas nesta seção. Inicialmente, iremos considerar $H = G$; no intervalo entre as linhas 4 e 8 encontra-se a operação semi-diamante; as operações diamante e duplo K_3 correspondem, respectivamente, aos intervalos entre as linhas 10 e 12 e entre 14 e 17. Na linha 21 temos o teste que verifica se o nó folha tem quatro (ou mais) vértices. Caso afirmativo, retorna-se para o nível mais recente da recursão e continua-se a busca. Se negativo, encontramos uma 3-coloração ótima para G.

Ao analisarmos as operações propostas, percebe-se facilmente que, em todas elas, ramos que levariam para grafos completos de tamanho quatro ou mais, são evitados (podados).

Algoritmo 5.1: 3-coloração (Faria et al., 2005).

```
1  REDUZA (H, n)    /* grafo H e n seu número de vértices */
2  verificar se Δ(H) ≥ 3 e testar se H não é bipartido;
3  se (H não é um grafo completo) então
4      se (H não possui triângulo) então
5          determine H_r1, H_r2 e H_r3;
6          REDUZA (H_r1, n − 1);
7          REDUZA (H_r2, n − 1);
8          REDUZA (H_r3, n − 1);
9      fim
10     senão se (existe em H o subgrafo H') então
11         determine H'_r;
12         REDUZA (H'_r, n − 1);
13     fim
14     senão se (existe em H o subgrafo H'') então
15         determine H''_r1 e H''_r2;
16         REDUZA (H''_r1, n − 1);
17         REDUZA (H''_r2, n − 1);
18     fim
19 fim
20 senão
21     se (n ≥ 4) então
22         retorne;
23     senão PARE, G é 3-colorível;
24 fim
```

Percebemos também que o número de vértices existentes em todos os nós pertencentes a um determinado nível é maior, em exatamente uma unidade, em relação aos nós pertencentes ao nível imediatamente posterior. Sendo assim, a altura máxima da árvore será de $n-3 = O(n)$.

Ainda, no pior caso, a árvore construída através das operações propostas acima, terá como raiz o grafo G (sem triângulo) e, no nível seguinte, os três filhos de G. Note que os nós formados pelos subgrafos H_{r1} e H_{r2} (subgrafos que possuem triângulo) terão no máximo dois filhos e que o nó formado por H_{r3} (sem triângulo) possuirá três filhos ou será podado por não conter um vértice com $\Delta \geq 3$.

Apresentamos, na Figura 5.11, uma representação da árvore de pior caso. Perceba que no nível k, $0 \leq k \leq n-3$, existem $2^{k+1}-1$ nós. Este resultado é válido se $k = 0$, suponha verdadeiro para $k < n-3$. Por construção, existe, em cada nível, um único nó correspondente a um subgrafo eventualmente sem triângulo (na figura, nós com um círculo). Assim, no nível $k+1$ da árvore existem, no máximo, $2(2^{k+1}-2)+3 = 2^{k+2}-1$ nós.

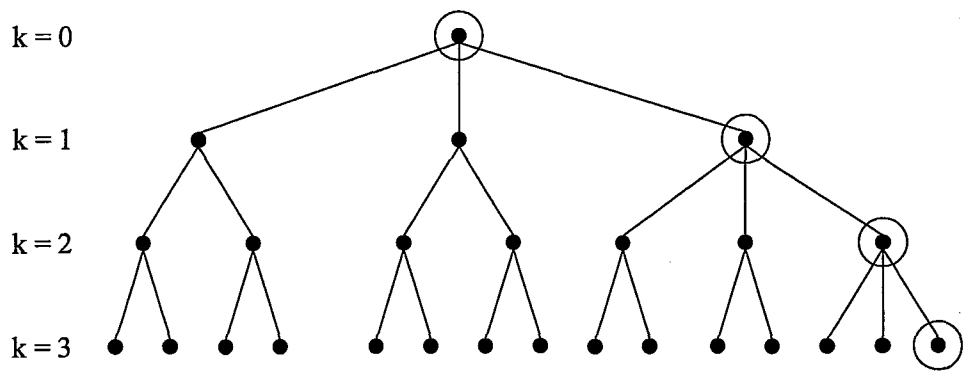

Figura 5.11: Árvore de pior caso para o novo algoritmo.

Portanto, o limite superior para o número total de nós pode ser calculado como em:

$$\begin{aligned}\sum_{i=0}^{n-3}(2^{i+1}-1) &= 2^{n-1}-2-(n-2)\\ &= 2^{n-1}-n\\ &= O(2^n)\end{aligned}$$

Da mesma forma que o algoritmo de Zykov, a complexidade de tempo deste novo método também será proporcional ao número total de nós da árvore, a qual é da ordem de $O(2^n)$.

A análise da complexidade de espaço é bastante semelhante à realizada para o algoritmo de Zykov: perceba que neste algoritmo a recursão irá definir, implicitamente, uma árvore Z reduzida (pois ramos que produzirão cliques de tamanho maiores ou iguais a quatro são ignorados). A altura desta árvore será de $O(n)$ e em cada nível da recursão gasta-se um espaço da ordem de $O(n+m)$ para armazenar cada subgrafo. Assim, a complexidade de espaço é de $O(nm)$.

Através dos resultados vistos acima, fica evidente que a idéia do autor foi a de explorar a árvore de Zykov e com isso, obter um algoritmo cuja complexidade de tempo é inferior a $O(2^m)$. Somado a isso, é conveniente destacar que este novo método possui uma mecânica elegante e simples, a qual contribui para um fácil entendimento do processo.

Capítulo 6

Algoritmos para 4-coloração

Estudaremos neste capítulo os algoritmos exatos pertencentes à última classe de problemas de coloração em grafos abrangidos por esta dissertação, a 4-coloração.

Assim como no capítulo anterior, o primeiro algoritmo apresentado é o proposto por LAWLER [32]. Tal método foi um dos primeiros a solucionar e fornecer a análise de complexidade para o problema em questão.

Na seção 6.2, mostramos o procedimento de BEIGEL e EPPSTEIN [3, 18], o qual baseia-se nas idéias e conceitos elaborados para o algoritmo de 3-coloração também proposto pelos mesmos autores (seção 5.2).

Encerrando o capítulo, temos o método de NIELSEN [36], que consiste em uma extensão do algoritmo para 3-coloração de Lawler (seção 5.1).

6.1 O método de Lawler

No mesmo artigo que trata do número cromático, LAWLER [32] apresenta uma idéia bastante simples e intuitiva para testar a 4-colorabilidade de um grafo G qualquer:

(1) gerar todos os 2^n subgrafos induzidos de G;

(2) para cada subgrafo H gerado, formar dois conjuntos de vértices, um conterá os vértices do subgrafo H e o outro, os vértices de $G \backslash H$;

(3) testar se os subgrafos induzidos contidos nos dois conjuntos são bipartidos;

(4) se encontrarmos algum par de conjuntos de vértices que satisfaça (3), então G é 4-colorível.

A complexidade deste método pode ser calculada da seguinte maneira: no primeiro passo, gasta-se um tempo de $O(2^n)$; no passo seguinte, os conjuntos podem ser formados em tempo constante; no terceiro, temos um tempo de $O(m+n)$ para cada teste de bipartição. Logo a complexidade de tempo do processo é de $O((m+n)2^n) \approx O^*(2^n)$.

Para medirmos o espaço utilizado pelo algoritmo, lembremos que este irá gerar todos os 2^n subgrafos de G (passo 1) porém, se garantirmos que cada subgrafo seja gerado uma única vez, armazenaremos, em cada iteração, somente os subgrafos H e $G \backslash H$, ou seja, todo o grafo. Assim, a complexidade de espaço é de $O(n+m)$.

6.2 O método de Beigel e Eppstein

Utilizando as idéias desenvolvidas em seu algoritmo para 3-coloração (vide Seção 5.2), BEIGEL e EPPSTEIN [3, 18] propõem um algoritmo exato aleatório (probabilístico) para o problema da 4-coloração em grafos. Tal procedimento está fundamentado no teorema abaixo:

Teorema 6.1 (Beigel e Eppstein, 2001). *Existe um algoritmo aleatório que encontra a solução para qualquer instância resolvível $(d, 2)$-CSP, com $d > 3$, em tempo (estimado) de $O^*((0.4518d)^n)$.*

Demonstração. Para cada vértice (variável) pertencente ao $(d, 2)$-CSP, escolhe-se (aleatoriamente) quatro das d cores disponíveis. Então, transformamos o problema em um $(4, 2)$-CSP, o qual é resolvido utilizando o algoritmo para CSP visto na Seção 5.2.2.

Repetimos o processo com uma nova escolha (também aleatória) de cores, até encontrarmos uma instância $(4, 2)$-CSP resolvível. Tais repetições serão efetuadas uma quantidade suficiente de vezes de modo a garantir uma probabilidade de falha pequena.

Esse número de tentativas apresenta como limite o valor de $(d/4)^n$, uma vez que, a atribuição aleatória de uma cor para um vértice possui uma probabilidade de $4/d$ de manter a solucionabilidade do problema.

Cada tentativa gasta um tempo de $O^*(1.36443^{(2-\epsilon)n}) \approx O^*(1.8072^n)$. O valor $(2 - \epsilon)$ representa o tamanho da instância $(4, 2)$-CSP composta exclusivamente por vértices com quatro cores (vide Seção 5.2.2). Logo, o tempo (estimado) total é de $O^*((d/4)^n \cdot 1.8072^n) \approx O^*((0.4518d)^n)$. □

O problema da 4-coloração pode ser transformado diretamente para um $(4, 2)$-CSP empregando a mesma idéia utilizada no caso da 3-coloração (Figura 5.1).

Desta forma, se aplicarmos o Teorema 6.1 para $d = 4$, conseguimos resolver o $(4, 2)$-CSP e consequentemente a 4-coloração em tempo da ordem de $O^*(1.8072^n)$.

Por fim, vale ressaltar ainda que, assim como no algoritmo para 3-coloração, também proposto pelos mesmos autores, este não utiliza o conceito de conjuntos independentes maximais para a obtenção de uma solução do problema.

6.3 O método de Nielsen

Descreveremos nesta seção as idéias e o funcionamento do algoritmo para 4-coloração proposto por NIELSEN [36]. Este procedimento, desenvolvido em 2002, é, dentre os existentes atualmente na literatura, o que consegue responder, em menor tempo, se um grafo é ou não 4-colorível.

Para obter essa menor complexidade, o autor utiliza a Técnica 6.2 desenvolvida por LAWLER [32], a qual verifica a k-colorabilidade de um grafo.

Técnica 6.2 (Lawler, 1976). *Para decidir se um grafo G é k-colorível, geramos todos os conjuntos independentes maximais S de tamanho maior ou igual a $\lceil n/k \rceil$ e, para cada S gerado, verificamos se $G[V \backslash S]$ é $(k-1)$-colorível.*

Se supormos que $k = 3$, percebe-se facilmente que a técnica descrita acima irá corresponder ao algoritmo para 3-coloração de Lawler (Seção 5.1). Para um k qualquer, aplicaremos a Técnica 6.2 recursivamente, até obtermos um subgrafo 2-colorível, o qual pode ser detectado em tempo linear. Assim sendo, um limite superior para a complexidade de tempo (ignorando fatores não exponenciais) exigida por esta técnica pode ser representada pela seguinte expressão:

$$\sum_{L=\lceil \frac{n}{k} \rceil}^{n} \left| S^{=L}(G) \right| \cdot T_{k-1}(n-L),$$

onde $T_{k-1}(n)$ é o tempo gasto para calcular a $(k-1)$-coloração.

Teorema 6.3 (Nielsen, 2002). *A 4-coloração pode ser verificada em um tempo da ordem de $O(1.7504^n)$.*

Demonstração. Para obter esse valor, aplicamos a Técnica 6.2 para $k = 4$. Assim sendo, somente iremos considerar os conjuntos independentes maxi-

mais de tamanho maior ou igual a $\lceil n/4 \rceil$. Ainda, lembremos que, o Teorema 2.9 garante que o número máximo de CIM's $S \subseteq V$ de tamanho no máximo L é de $3^{4L-n}4^{n-3L}$.

Após gerados os CIM's de G, utilizamos o algoritmo de Beigel e Eppstein para 3-coloração (vide Seção 5.2) a fim de testar a 3-colorabilidade de cada grafo $G[V \backslash S]$ restante.

Desta forma, o limite superior (visto anteriormente) para a complexidade de tempo exigido pela Técnica 6.2 resulta em:

$$\sum_{L=\lceil \frac{n}{4} \rceil}^{n} 3^{4L-n}4^{n-3L} \cdot 1.3289^{n-L}$$

que equivale a $O(4^{n/4} \cdot 1.3289^{3n/4}) = O(1.7504^n)$. □

Recentemente, foi descrito um outro algoritmo para 4-coloração [12] o qual é baseado na geração de todos os subgrafos maximais bipartidos de um dado grafo. Cada um desses subgrafos pode ser asssociado a duas classes de cores da 4-coloração ótima. Neste trabalho é descrito um método para gerar todos os subgrafos maximais bipartidos em tempo $O(1.8613^n)$. Isto conduz a um algoritmo dessa mesma complexidade, o qual contudo apresenta uma complexidade maior do que o método acima descrito.

Antes de finalizarmos, gostaríamos de esclarecer ao leitor que Nielsen é também Byskov. Não sabemos o motivo pelo qual ele utilizou este pseudônimo, mas manteremos a autoria do algoritmo ao primeiro nome, uma vez que este foi o empregado pelo autor no artigo mencionado.

Capítulo 7

Conclusões

Neste capítulo, apresentaremos os resultados obtidos por esta dissertação. A partir destes, proporemos alguns problemas e forneceremos direções para trabalhos futuros.

O primeiro resultado deste trabalho é o estudo minucioso dos diversos algoritmos exatos existentes na literatura para problemas de coloração de vértices em grafos.

O objetivo visado com este estudo foi a realização de um levantamento do assunto tratado, uma vez que este abrange os problemas importantes relacionados à coloração de grafos. Procuramos apresentar os algoritmos com a maior riqueza de detalhes possível, de modo a facilitar o entendimento do leitor. Embora admitindo que um ou outro algoritmo possa ter escapado à nossa pesquisa, estamos convictos que a gama de métodos estudados é representativa e constitui-se em um trabalho de grande valia para novas pesquisas e/ou desenvolvimento de novos métodos.

Além deste levantamento, citamos a execução dos cálculos de complexidade para vários algoritmos, dentre estes, merecem destaque a análise de métodos tradicionais, tais como o de CHRISTOFIDES [13] e o de ZY-

KOV [46], as quais, segundo a bibliografia consultada, ainda não tinham sido efetuadas.

O outro resultado consiste na elaboração de um novo algoritmo exato para determinar se um grafo qualquer pode ser colorido com três cores. Este método apresenta como novidade o fato de empregar idéias bastante conhecidas, mas que ainda não tinham sido utilizadas para a resolução do problema da 3-coloração.

Além disto, outra característica, presente neste método é a de que este, assim como o de BEIGEL e EPPSTEIN [2, 3], não utiliza o conceito de conjuntos independentes maximais na busca por uma solução ótima. Em outras palavras, não utiliza um algoritmo auxiliar para a detecção de tais conjuntos.

Ainda, vale ressaltar que, se compararmos com o anteriormente citado, este novo algoritmo apresenta uma idéia extremamente simples e bastante intuitiva.

Dentro do contexto abordado pela presente dissertação, alguns problemas podem ser formulados. Dentre estes citamos: é possível desenvolver um algoritmo para a 4-coloração que esteja fundamentado nos conceitos propostos por Zykov? Caso afirmativo, além de buscarmos uma menor complexidade, estaríamos enriquecendo o conjunto de algoritmos para este tipo de problema, uma vez que, pelo que pudemos perceber, a 4-coloração é, dentre as classes estudadas, a que possui o menor número de soluções propostas.

Outro problema, cuja importância merece destaque, é: existe um algoritmo exato que encontre o número cromático de um grafo qualquer, tal que suas complexidades de tempo e espaço sejam, respectivamente, $O^*(c^n)$ e $O(n^k)$, para c e k constantes? Ou seja, estaríamos interessados em um método que, mesmo não possuindo a menor complexidade de tempo entre os

já existentes, tivesse uma complexidade de espaço polinomial, pois, segundo vários autores, com o avanço nos limites para a complexidade de tempo, as medidas de espaço atuais tornar-se-ão o fator dominante na eficiência dos algoritmos.

Para finalizar, mostraremos as direções para trabalhos futuros que surgem com os resultados alcançados por este trabalho. A primeira está relacionada com o fato deste possuir um caráter de levantamento. Mais precisamente, poderíamos utilizar o material pesquisado para realizar um estudo comparativo (diferença entre complexidades, qualidade das soluções encontradas, etc.) entre os métodos heurísticos e os exatos.

Uma segunda direção, diz respeito ao novo algoritmo desenvolvido e resume-se a: verificar se o novo método produzirá bons resultados para uma classe específica de grafos. Acreditamos que, como o algoritmo fundamenta-se nas idéias de Zykov, este deve apresentar um bom desempenho para grafos densos, uma vez que, para este tipo de grafos, um pequeno número de reduções deverão ser realizadas.

Encerramos, apresentando, na Tabela 7.1 um quadro cronológico dos algoritmos estudados. Note que, em cada método, a célula superior corresponde a complexidade de tempo, enquanto que a inferior representa a complexidade de espaço.

	$\chi(G)$	3-coloração	4-coloração
Zykov (1949)	$O(2^{n^2})$	-	-
	$O(n^2m)$	-	-
Christofides (1971)	$O^*(1.4422^{n^2})$	-	-
	$O(2^n)$	-	-
Roschke e Furtado (1973)	$O^*(1.4422^{n^2})$	-	-
	$O(n(3^{n/3}))$	-	-
Lawler (1976)	$O^*(2.4422^n)$	$O^*(1.4422^n)$	$O^*(2^n)$
	$O(2^n)$	$O(n+m)$	$O(n+m)$
Szwarcfiter (1978)	$O^*(1.4422^{n^2})$	-	-
	$O(n+m)$	-	-
Nielsen (2002)	-	-	$O(1.7504^n)$
	-	-	
Eppstein (2003)	$O^*(2.4150^n)$	-	-
	$O(2^n)$	-	-
Byskov (2004)	$O(2.4023^n)$	-	-
	$O(2^n)$	-	-
Beigel e Eppstein (2005)	-	$O^*(1.3289^n)$	$O^*(1.8072^n)$
	-		
Novo algoritmo (2005)	-	$O(2^n)$	-
	-	$O(nm)$	-

Tabela 7.1: Relação cronológica dos métodos.

Bibliografia

[1] APPEL, K., HAKEN, W., "A proof of the four color theorem", *Discrete Mathematics*, v. 16, pp. 179–180, 1976.

[2] BEIGEL, R., EPPSTEIN, D., "3-coloring in time $o(1.3446^n)$: a no-mis algorithm", In: *Proc. 36th Symp. Foundations of Computer Science, IEEE*, pp. 444–453, 1995.

[3] BEIGEL, R., EPPSTEIN, D., "3-coloring in time $O(1.3289^n)$", *Journal of Algorithms*, v. 54, pp. 168–204, 2005.

[4] BOMZE, I. M., BUDINICH, M., PARDALOS, P. M., *et al.*, "The Maximum Clique Problem", In: *Handbook of Combinatorial Optimization (Supplement Volume A)* (DU, D.-Z., PARDALOS, P. M., eds.), pp. 1–74, Boston, Massachusetts, U.S.A., Kluwer Academic, 1999.

[5] BONDY, J. A., MURTY, U. S. R., *Graph Theory with Applications.* 1 ed. New York, North Holland, 1976.

[6] BRÈLAZ, D., "New methods to color the vertices of a graph", *Communications of the Assoc. of Comput. Machinery*, v. 22, pp. 251–256, 1979.

[7] BROWN, J. R., "Chromatic scheduling and the chromatic number problem", *Management Science*, v. 19, pp. 456–463, 1972.

[8] BYSKOV, J. M., "Chromatic Number in Time $O(2.4023^n)$ Using Maximal Independent Sets", Tech. Rep. RS-02-45, BRICS Research Series, 2002.

[9] BYSKOV, J. M., "Algorithms for k-colouring and finding maximal independent sets", In: *Proc. 14th Symp. Discrete Algorithms, ACM and SIAM*, pp. 456–457, 2003.

[10] BYSKOV, J. M., "Enumerating maximal independent sets with applications to graph colouring", *Operations Research Letters*, v. 32, n. 6, pp. 547–556, 2004.

[11] BYSKOV, J. M., *Exact Algorithms for Graph Colouring and Exact Satisfiability*, Ph.D. dissertation, University of Aarhus, Aarhus, Denmark, 2004.

[12] BYSKOV, J. M., MADSEN, B. A., SKJERNAA, B., "On the number of maximal bipartite subgraphs of a graph", *Journal of Graph Theory*, v. 48, n. 2, pp. 127–132, 2005.

[13] CHRISTOFIDES, N., "An Algorithm for the Chromatic Number of a Graph", *The Computer Journal*, v. 14, n. 1, pp. 38–39, 1971.

[14] CORMEN, T. H., LEISERSON, C. E., RIVEST, R. L., *et al.*, *Introduction to Algorithms*. 2 ed. Massachusetts and Boston, The MIT Press and McGraw-Hill Book Company, 2001.

[15] CORNEIL, D. G., GRAHAM, B., "An algorithm for determining the chromatic number of a graph", *SIAM Journal on Computing*, v. 2, pp. 311–318, 1973.

[16] CROITORU, C., "On stables in graphs", In: *Proc. 3rd Coll. Operations Research*, pp. 55–60, 1979.

[17] DUTTON, R. D., BRIGHAM, R. C., "A New Graph Colouring Algorithm", *The Computer Journal*, v. 24, n. 1, pp. 85–86, 1981.

[18] EPPSTEIN, D., "Improved algorithms for 3-coloring, 3-edge-coloring and constraint satisfaction", In: *Proc. of 12th Symposium Discrete Algorithms, ACM and SIAM*, pp. 329–337, 2001.

[19] EPPSTEIN, D., "Small Maximal Independent Sets and Faster Exact Graph Coloring", *Journal of Graph Algorithms and Applications*, v. 7, n. 2, pp. 131–140, 2003.

[20] FARIA, L., Comunicação particular.

[21] FRANçA, F. M. G., PROTTI, F., SZWARCFITER, J. L., "On Computing All Maximal Cliques Distributedly", *Lecture Notes in Computer Science*, v. 1253, pp. 37–48, 1997.

[22] GAREY, M. R., JOHNSON, D. S., *Computers and Intractability: A Guide to the Theory of NP-Completeness.* San Francisco, W. H. Freeman, 1979.

[23] GRAHAM, R. L., KNUTH, D. E., PATASHNIK, O., *Concrete Mathematics: A Foundation for Computer Science.* 2 ed. New York, Addison-Wesley, 1994.

[24] HAMMER, P. L., RUDEANU, S., *Boolean Methods in Operations Research.* Berlim, New York, Springer-Verlag, 1968.

[25] IMPAGLIAZZO, R., PATURI, R., ZANE, F., "Which problems have strongly exponential complexity?", In: *Proc. of the 39th Annual Symposium on Foundations of Computer Science (FOCS'1998)*, pp. 653–663, 1998.

[26] JOHNSON, D. S., YANNAKAKIS, M., "On generating all maximal independent sets", *Information Processing Letters*, v. 27, pp. 119–123, 1980.

[27] KARP, R. M., "Reducibility among combinatorial problems", In: *Complexity of Computer Communications*, pp. 85–103, Plenum Press, 1972.

[28] KARP, R. M., "On the computational complexity fo combinatorial problems", *Networks*, v. 5, pp. 45–68, 1975.

[29] KLOTZ, W., "Graph Coloring Algorithms", *Mathematik-Bericht, TU Clausthal*, v. 5, pp. 1–9, 2002.

[30] KULLMANN, O., LUCKHARDT, H., "Algorithms for SAT/TAUT decision based on various measures", 1999.

[31] KUMAR, V., "Algorithms for constraint satisfaction problems: a survey", *AI Magazine*, v. 13, pp. 32–44, 1992.

[32] LAWLER, E. L., "A note on the complexity of the chromatic number problem", *Information Processing Letters*, v. 5, n. 3, pp. 66–67, 1976.

[33] LAWLER, E. L., LENSTRA, J. K., KAN, A. H. G. R., "Generating all maximal independent sets: NP-hardness and polynomial-time algorithms", *SIAM Journal on Computing*, v. 9, pp. 558–565, 1980.

[34] MADSEN, B. A., NIELSEN, J. M., SKJERNAA, B., "On the number of maximal bipartite subgraphs of a graph", Tech. Rep. RS-02-16, BRICS Research Series, 2002.

[35] MOON, J. W., MOSER, L., "On cliques in graphs", *Israel Journal of Mathematics*, v. 3, pp. 23–28, 1965.

[36] NIELSEN, J. M., "On the number of maximal independent sets in a graph", Tech. Rep. RS-02-15, BRICS Research Series, 2002.

[37] PAPADIMITRIOU, C. H., STEIGLITZ, K., *Combinatorial Optimization: algorithms and complexity.* New York, Dover Publications, 1998.

[38] PAULL, M. C., UNGER, S. H., "Minimizing the number of states in incompletely specified sequential switching functions", *IRE Trans. Electron. Comput.*, v. 8, pp. 356–367, 1959.

[39] ROSCHKE, S. I., FURTADO, A. L., "An Algorithm for Obtaining the Chromatic Number and an Optimal Coloring of a Graph", *Information Processing Letters*, v. 2, n. 2, pp. 34–38, 1973.

[40] SANTOS, R. N., *Obtenção do número cromático de um grafo*, Tese de M.Sc., COPPE/UFRJ, Rio de Janeiro, RJ, Brasil, 1979.

[41] SCHöNING, U., "A probabilistic algorithm for k-sat and constraint satisfaction problems", In: *Proc. 40th IEEE Symp. Foundations of Computer Science*, pp. 410–414, 1999.

[42] SZWARCFITER, J. L., Comunicação particular.

[43] TSUKIYAMA, S., IDE, M., ARIYOSHI, H., *et al.*, "A new algorithm for generating all the maximal independent sets", *SIAM Journal on Computing*, v. 6, n. 3, pp. 505–517, 1977.

[44] WANG, C. C., "An Algorithm for the Chromatic Number of a Graph", *Journal of the ACM*, v. 21, n. 3, pp. 385–391, 1974.

[45] WOEGINGER, G. J., "Exact algorithms for NP-hard problems: a survey", *Combinatorial Optimization: "Eureka, you shrink", LNCS*, v. 2570, pp. 185–207, 2003.

[46] ZYKOV, A. A., "On some properties of linear complexes", *Mat. Sb.*, v. 24, pp. 163–188, 1949.