OBTENÇÃO DO NÚMERO CROMÁTICO DE UM GRAFO

Rosângela da Nóbrega Santos

TESE SUBMETIDA AO CORPO DOCENTE DA COORDENAÇÃO DOS PROGRAMAS DE PÓS-GRADUAÇÃO EM ENGENHARIA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO, COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS (M.Sc.).

Aprovada por:

JAYME LUIZ SZWARCFITER
(Presidente)

JOSÉ LUCAS MOURÃO RANGEL NETO

PAULO AUGUSTO SILVA VELOSO

RIO DE JANEIRO, - RJ

JANEIRO DE 1979

FICHA CATALOGRÁFICA.

SANTOS, ROSÂNGELA DA NÓBREGA

Obtenção do Número Cromático de um Grafo

[Rio de Janeiro] 1979.

XIII, 121 p. 29,7 cm (COPPE - UFRJ, M. Sc., Engenharia de Sistemas, 1979.

Tese - Universidade Federal do Rio de Janeiro - Faculdade de Engenharia.

1. Levantamento e análise dos algoritmos existentes para obtenção do número cromático de um grafo. I. COPPE/UFRJ II. Título (série).

CURRICULUM VITAE.

NOME.

Rosângela da Nóbrega Santos.

NASCIMENTO.

Na cidade de Araguari, MG, em 02 de novembro de 1952, filha de Hermilo Souto Nóbrega e Laura Gomes da Nóbrega.

ESTADO CIVIL.

Casada, com Márcio Santos, analista de sistemas.

FORMAÇÃO ACADÊMICA.

Bacharel em Matemática, pela Universidade de Brasília, em julho de 1975.

EXPERIÊNCIA PROFISSIONAL EM COMPUTAÇÃO.

Estagiária, programadora, analista de sistemas e professora,na Universidade de Brasília, no período de 1972 a 1975.
Professora no curso LTD-DATAMEC, em 1972 e 1973.
Analista de sistemas, no Instituto Brasileiro de Geografia e Estatística, de 1975 a 1977.

CURSOS AVULSOS REALIZADOS EM COMPUTAÇÃO.

Linguagem Assembler GP300 para o minicomputador L2000.
Linguagem de máquina B-500.
Linguagem Assembler PMB.
Linguagem Fortran IV.
Linguagem Assembler 1130.
Linguagem BASIC 1130.
Linguagem COBOL.
Sistema de Gerenciamento de Dados DMS-6700.
Linguagem de Controle de "Job" WFL.
"Data Comunication", Conceitos Básicos em Software e Hardware.
Estruturas de Dados e Métodos de Classificação.

Linguagem Algol.
Componentes de Hardware para Teleprocessamento.
Programação Estrutura.
OS/VS - Funções do Sistema.
Conceitos de VM/370.
Linguagem PLI para OS/VS.
Linguagem de Controle de "Job" JCL.
Linguagem Assembler 370 para OS/VS.

ENDEREÇO ATUAL.

SQS 410, Bloco O, Ap. 101-D
70 276 Brasília, DF.
Fone (061) 243-5523.

DEDICATÓRIA.

Ao Márcio,

pelo incentivo e compreensão.

AGRADECIMENTOS.

Aos professores que se propuseram a enriquecer este trabalho com suas sugestões.

Em particular,
ao professor Jayme Luiz Szwarcfiter,
por sua valiosa contribuição intelectual e
por sua inestimável contribuição humana.

Aos amigos e familiares, sem cujo apoio este trabalho não teria sido possível.

Especialmente,
ao meu pai,
por sua grande ajuda na revisão do texto
e na confecção das figuras.

A DEUS,
pelo DOM de termos podido realizar este trabalho.

RESUMO.

Esta tese introduz o problema de coloração de grafos, seguido por um estudo de sua complexidade e pelo levantamento dos algoritmos de coloração existentes, incluindo as heurísticas.

Procura-se abranger o maior número possível de algoritmos e apresentá-los de maneira uniforme, por exemplo utilizando-se em todos eles a notação "algol-like", para proporcionar melhor visão de conjunto e facilitar o estudo comparativo.

Um algoritmo de coloração ainda inédito é formulado e mostra-se que é possível implementá-lo em espaço que cresce linearmente com o tamanho do grafo.

ABSTRACT.

This thesis introduces the graph coloring problem, followed by a study of its complexity and a survey of the existing coloring algorithms, including heuristics.

The goals are to comprehend the most possible number of algorithms and to present them uniformly, for instance using algol-like notation in all of them, in order to give a better global sight and to facilitate comparative work.

An yet unpublished algorithm is formulated and it is shown the way of implementing it in space that grows linearly with the size of the graph.

ÍNDICE.

ÍNDICE DOS QUADROS.

ÍNDICE DAS FIGURAS.

I. INTRODUÇÃO.

Neste trabalho, tratamos o problema do número cromático de um grafo, tanto do ponto de vista da teoria em si, quanto da abordagem algorítmica para resolução do problema. A ênfase maior é dada a esse segundo aspecto, como pode ser depreendido do próprio título da tese, servindo a discussão inicial para montar o cenário, motivar o drama, fornecer terminologia adequada à conversação e determinar a qualidade e o papel representado pelos dois grandes grupos de artistas, os algoritmos de coloração aproximada e os algoritmos de coloração exata.

Inicia-se com a conceituação de número cromático de um grafo e o relato dos acontecimentos que levaram à busca do mesmo, sendo então definidos alguns termos básicos para se poder passar a um tratamento mais formal do assunto.

Um capítulo inteiro é escrito sobre possibilidades de aplicações do número cromático a situações cotidianas, frisando-se o grande interesse existente em torno de uma dessas aplicações, a saber, a construção de tabelas de horário para exames em escolas.

Em seguida, discutem-se as características de grafos, que podem ser relacionadas com seu número cromático, e os limites inferiores e superiores teoricamente deduzidos para o mesmo.

Um estudo de como o tempo despendido na execução de um algoritmo exato de coloração pode crescer, quando se aumenta o tamanho do grafo considerado, é apresentado sob o título Complexidade.

Além de separados em aproximativos e exatos, os algoritmos são postos em cena em pequenos grupos, caracterizados pela semelhança dos procedimentos que adotam. Essa tentativa de classificação é útil para se distinguir melhor os algoritmos que tratam o problema com um enfoque completamente novo, dos que consistem em versões modificadas de outros já existentes. Dentro desse espírito, os algoritmos de cada grupo são considerados por ordem cronológica de sua publicação.

Um dos objetivos desta tese é proporcionar a apresentação conjunta e uniforme dos algoritmos de coloração existentes. Acreditamos que, assim organizados, esses algoritmos possam suscitar futuros estudos comparativos de complexidade, mais detalhados e exatos que o nosso. Cabe salientar que estudos desse tipo ainda não existem na literatura.

Esperamos não ter prejudicado a essência dos algoritmos ao vesti-los com a nova roupagem. Por exemplo, procuramos utilizar variáveis de mesmo nome para armazenar informações idênticas, e formulamos todos os algoritmos na notação "algol-like".

Para dirimir quaisquer dúvidas quanto à origem das falhas ou imprecisões que possam vir a ser encontradas nos algoritmos aqui expostos, anexamos ao final do trabalho as suas formulações originais.

O outro objetivo visado com a presente tese é expresso pelo seu caráter de levantamento do assunto tratado, que pretendíamos fosse o mais abrangente possível. Embora admitindo que um ou outro algoritmo possa ter escapado à nossa pesquisa em bibliotecas do Rio e de Brasília, estamos convencidos de que o conjunto conseguido é representativo de todos os procedimentos já imaginados para colorir grafos, constituindo um trabalho de valor para pesquisadores que pretendam apresentar novas contribuições.

Das obras mais conhecidas, a que faz referência ao maior número de algoritmos de coloração é o livro de Christofides[9], em que são explicados vários procedimentos de coloração exata e de coloração aproximada. As demais obras se limitam a apresentar um algoritmo de coloração exata ou, no máximo, um algoritmo de cada tipo. Podemos citar os livros de Berge[2], Deo[15], Even[16], Furtado[19], Boaventura[4] e Nijenhuis e Wilf[37].

II. O NÚMERO CROMÁTICO.

O conceito de número cromático aparece dentro do assunto mais geral que é coloração de grafos. Iniciemos, portanto, esse trabalho, dizendo informalmente o que são grafos, em que consiste a sua coloração e como tal assunto foi abordado pela primeira vez. Mostraremos como se chegou ao problema de obtenção do número cromático de um grafo, tanto a partir de uma questão teórica, como a partir de uma necessidade prática.

II-1 . Grafo: Um Objeto Abstrato ou uma Figura Geométrica?

A idéia que temos de um grafo é a de um conjunto de elementos, entre cujos pares pode, ou não, haver uma determinada relação. Em geral, um grafo aparece sempre associado à sua representação gráfica. Um conjunto de pontos representa os seus elementos e uma linha, ligando dois desses pontos, simboliza a existência de relação entre os elementos correspondentes.

Da mesma forma que, na Aritmética, deixamos de lado a diferenciação entre número e numeral, também aqui não há prejuízo em nos referirmos à representação gráfica de um grafo como se fora ao próprio grafo. N. Deo[15] conceitua com precisão essas entidades, o grafo como objeto abstrato combinatório e o grafo como representação geométrica, afirmando, em seguida, que se passa por cima da distinção deliberadamente, em prol da simplicidade e clareza.

Na figura (II-1), a seguir, vemos um grafo G com seis pontos: $\{ v_1, v_2, \ldots, v_6\}$, dos quais os pares (v_1,v_2), (v_1,v_3), (v_1,v_5), (v_2,v_3), (v_2,v_4) e (v_4,v_6) estão relacionados entre si.

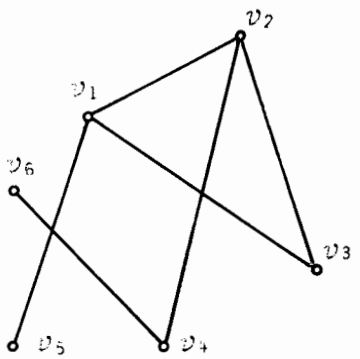

Figura II-1. Representação gráfica de um grafo.

II-2. Colorindo Grafos.

Colorir um grafo significa pintar cada um dos seus pon- com uma cor, de forma tal que, se dois pontos quaisquer estão ligados entre si por uma linha, eles recebem cores distintas. É claro que o termo "pintar com uma cor" é equivalente a "associ ar um dado número". A referência a cores é feita por motivos históricos, como veremos a seguir, além de ser valiosa para que se visualize melhor a questão.

Dado um grafo e um certo número de cores, pode não exis tir um modo de atribuir aos seus pontos as cores dadas, ou pode existir uma ou mais maneiras de fazê-lo. O grafo G apresentado na seção anterior pode ser colorido com quatro cores de muitas maneiras distintas, como vemos na figura (II-2).

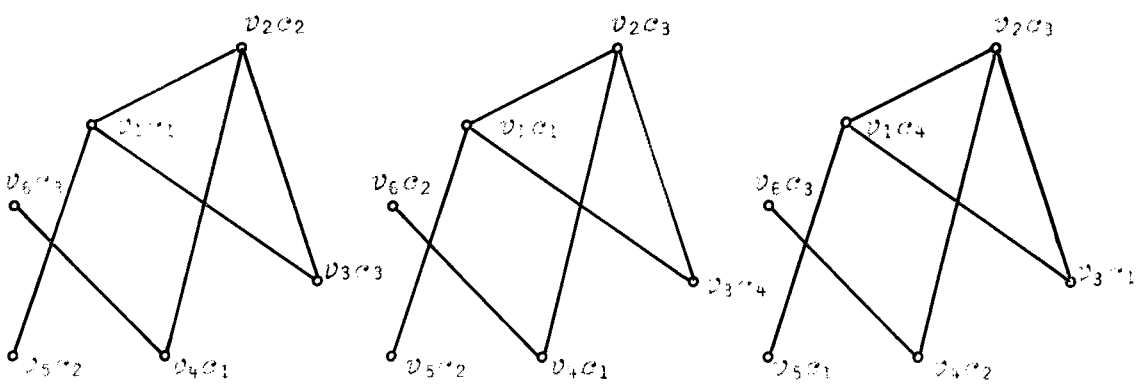

Figura II-2. Três maneiras de colorir G, usando quatro cores.

II-3. Origem Histórica.

Em 1852, o professor inglês De Morgan escreveu uma carta ao seu colega Sir Hamilton, expondo um problema que não conseguira solucionar. Ainda hoje, os professores se correspondem pelo mesmo motivo e formulam hipóteses, as quais são chamadas conjecturas até que venham a ser demonstradas ou derrubadas pela descoberta de contra-exemplos. Ao lançar uma conjectura, princi palmente se a mesma pode ser expressa de uma forma simples e concisa, o autor não imagina que ela possa originar tanta movimentação entre os cientistas, como foi o caso da conjectura de De Morgan (ou Conjectura de Guthrie, para fazer jus ao incômodo aluno que lhe apresentou a questão).

Um aluno havia perguntado a De Morgan qual a razão de

ser aparentemente possível colorir uma figura qualquer com, no máximo, quatro cores, sem que regiões vizinhas recebessem cores iguais. Observou De Morgan que em uma figura com cinco regiões, qualquer esforço para fazer com que cada uma das regiões tenha uma linha limítrofe com as demais,leva sempre a uma situação em que uma das regiões é completamente envolvida pelas outras três, o que impede a quinta região de se limitar com a mesma. Reproduzimos na figura (II-3) o desenho utilizado por Rouse-Ball[40], ainda no século passado, para argumentar a favor da conjectura, usando a dificuldade relatada acima. Qualquer disposição do distrito X impede a posterior adição de um distrito Y que seja vizinho aos distritos A, B, C, e X.

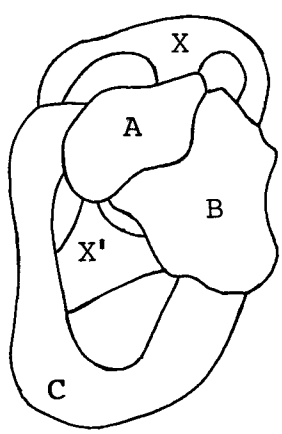

Figura II-3. Disposição de distritos em um mapa.

Ele conclui afirmando que "uma prova da proposição envolve dificuldades de alta ordem, as quais, até agora, têm frustrado todas as tentativas para superá-las".

II-4. Modelando por Grafo.

Cada região de uma figura pode ser associada a um ponto de um grafo, ligando-se dois pontos por uma linha sempre que as duas regiões correspondentes tiverem um limite em comum. Dessa forma modela-se o problema de De Morgan por grafos, podendo-se, então, formulá-lo com sua nova roupagem: "É possível colorir qualquer grafo obtido pelo processo acima com, no máximo, quatro cores?" (Obs.: "colorir" conforme definido na seção (II-2)).

Note que o grafo obtido pelo processo acima é sempre

um grafo planar, isto é, ele pode ser desenhado numa folha de papel sem que haja cruzamentos de suas linhas.

A equivalência entre o problema de colorir as regiões de uma figura, apresentado por De Morgan, e o problema de coloração de mapas planares pode ser formalmente demonstrada. Veja, por exemplo, a explicação da página 131 da obra de Harary[24].

II-5. O Desafio.

A "Conjectura das Quatro Cores" - conforme foi denominada a afirmação de que quatro cores são suficientes para se colorir qualquer grafo planar - deu muito o que fazer aos estudiosos de grafos, que tentaram estabelecer matematicamente a sua veracidade.

Por outro lado, não há dúvida que foram as dificuldades encontradas na demonstração dessa conjectura que motivaram um notável crescimento da Teoria dos Grafos e o desenvolvimento de resultados que, em geral e à primeira vista, nada têm a ver uns com os outros, ou com o problema original de coloração, mas que surgiram de diferentes abordagens dadas à Conjectura das Quatro Cores.

Embora Kempe, em 1879, já tivesse apresentado uma demonstração da conjectura, em 1890, Heawood mostrou que havia uma falha na mesma. Desde que foi inicialmente formulada, a Conjectura das Quatro Cores permaneceu inviolável por mais de um século, apresentando-se como um desafio atraente à sagacidade dos muitos teóricos de grafos, que mergulharam nela.

Foi apenas em 1977 que a questão perdeu o seu caráter conjecturístico, tendo sido demonstrada por Haken e Appel de uma forma pouco elegante, uma vez que requereu 1200 horas de computador, num trabalho brutal de verificação de um grande número de grafos (Bernhart[3]). Espera-se que essa primeira demonstração facilite o desenvolvimento de alguma prova curta e elegante.

II-6. Uma Razão Prática para Colorir Grafos.

Embora o problema de coloração de grafos planares tenha sido a grande estrela da história dos grafos, a sua extensão para grafos quaisquer é muito mais importante, do ponto de vista

da aplicabilidade a casos cotidianos.

A k- colorabilidade de um grafo é a propriedade de o mesmo poder ser colorido utilizando-se k cores. O problema a que nos referimos acima, que é uma generalização do problema das quatro cores de grafos planares, consiste em, dado um grafo qualquer G e um número inteiro k, saber se G é k-colorável.

Visto de um outro enfoque, o problema de colorabilidade de grafos pode ser apresentado como: "Encontrar o menor valor de k, tal que G seja k-colorável, e exibir um k-colorido de G". Nesse caso, k é denominado número cromático de G e o k-colorido é referenciado como colorido ótimo de G.

Na década de 60, foi estimulado o interesse por procedimentos para fornecer coloridos ótimos, ou quase ótimos, de um grafo, ao observar-se que isto poderia solucionar o problema de construção de tabelas de horário de escolas.

Vários algoritmos aproximativos para colorir grafos foram desenvolvidos com esse fim, tendo-se notícia de que são utilizados com sucesso por colégios e universidades (Williams[48]).

O grande problema desses algoritmos aproximativos, ou heurísticas, é que não se conseguiu determinar a distância máxima entre a estimativa do número cromático, por eles obtida para um grafo qualquer, e o número cromático real desse grafo. Sabe-se apenas que, para qualquer um dos algoritmos aproximativos,existem situações em que se obtêm resultados arbitrariamente ruins.

Vários algoritmos exatos foram apresentados para coloração de grafos, havendo, para qualquer deles, grafos em que o tempo de execução cresce exponencialmente com o número de pontos, tornando inviável sua aplicação.

Mais adiante, avaliaremos com mais detalhes a qualidade de algoritmos aproximativos e algoritmos exatos, que podem ser desenvolvidos para coloração. Apresentaremos alguns dos algoritmos já desenvolvidos e tentaremos traçar uma comparação teórica de seu desempenho.

III. DEFINIÇÕES.

Passamos, agora, a relacionar as definições dos termos necessários à exposição que se seguirá. Deixaremos para apresentar as definições dos termos de aplicação restrita aos algoritmos juntamente com os mesmos. Adotamos, em geral, as mesmas definições do Harary[24].

III-1. O Grafo e seus Elementos.

Um grafo G é um conjunto finito V = V (G) de n pontos e um conjunto X de m linhas, sendo cada linha um par não ordenado de pontos distintos de V.

Seja x = (u,v) uma linha de G. Dizemos que x liga u e v e denotamos x por uv. Os pontos u e v são adjacentes porque estão ligados por uma linha. O ponto u e a linha x são incidentes, um com o outro, assim como o são o ponto v e a linha x.

Seja v_i um ponto de G. O grau de v_i, denotado por d_i é o número de linhas incidentes com v_i.

Chamamos de lista de adjacência de v, e denotamos por A(v), uma lista encadeada contendo todos os pontos adjacentes a v, em G. Uma estrutura de adjacência para G é a representação de G por meio das listas de adjacência de seus pontos. Em quase todos os algoritmos assumiremos que G está representado por uma estrutura de adjacência.

A matriz de adjacência de G é uma matriz Q de dimensão n x n, onde o elemento q_{ij} é definido assim:

$$q_{ij} = \begin{cases} 1, & \text{se } v_i \text{ e } v_j \text{ são adjacentes,} \\ 0, & \text{caso contrário.} \end{cases}$$

Um trajeto em G é um seqüência alternada de pontos e linhas $v_0, x_1, v_1, x_2, v_2, \ldots, x_{k-1}, v_{k-1}, x_k, v_k$, começando e terminando por pontos, na qual cada linha é incidente com o ponto que a precede e com o ponto que a segue. Em geral, denotamos o trajeto acima por $v_0 v_1 v_2 \ldots v_{k-1} v_k$ e dizemos que ele liga os pontos v_0 e v_n.

Um caminho é um trajeto em que todos os pontos são distintos.

Um ciclo é um trajeto em que todos os pontos são distintos, exceto o primeiro e o último. O comprimento de um ciclo é o número de linhas do mesmo. Um triângulo é um ciclo de comprimento 3.

III-2. Tipos de Grafos.

G é um grafo completo, se e somente se quaisquer dois pontos distintos de G forem adjacentes.

G é um grafo nulo, se e somente se o conjunto V for vazio.

G é um grafo conexo, se e somente se qualquer par de pontos for ligado por um caminho; caso contrário, G é um grafo desconexo.

G é um grafo totalmente desconexo, se e somente se o conjunto X for vazio.

G é um grafo bipartite, se e somente se existir uma partição $\{V_1, V_2\}$ de V, tal que qualquer linha de G liga um ponto de V_1 e um ponto de V_2.

Dizemos que um grafo está inserido em uma superfície orientável S, quando ele está desenhado em S de forma que suas linhas não se cruzam.

G é um grafo planar se ele pode ser inserido em um plano.

Dado G, seu grafo complementar $\bar{G}$ contem o mesmo conjunto V de pontos, mas dois pontos são adjacentes em $\bar{G}$, se e somente se eles não o são em G.

III-3. Subconjuntos.

Um subgrafo de G é um grafo cujos conjuntos de pontos e de linhas são subconjuntos de V e de X, respectivamente.

Uma clique de G é um subgrafo completo de G. Uma k-clique é uma clique com k pontos.

O número máximo de linhas de G é dado pela fórmula - $n(n-1)/2$. À razão entre m e esse valor denominamos densidade de linhas de G.

Um cluster de G é um subgrafo de G, que tem uma alta densidade de linhas. Um k-cluster é um cluster com k pontos.

Um subconjunto de pontos (ou de linhas) de G, com uma dada característica, é um subconjunto maximal em relação a essa característica, se não estiver propriamente contido em nenhum outro subconjunto de pontos (ou de linhas) de G, com a mesma característica.

Um subgrafo de G com uma dada característica é um subgrafo maximal em relação à mesma, se seus conjuntos de pontos e de linhas são subconjuntos maximais mantendo tal característica.

Um componente de G é um subgrafo conexo maximal de G.

Para qualquer subconjunto S de pontos de G, o subgrafo induzido $< S >$ é o subgrafo maximal de G, com conjunto de pontos S.

Um subconjunto de pontos S, de G, é um subconjunto independente de G, se e somente se não houver em S dois pontos adjacentes.

III-4. Coloração de Grafos.

Um colorido de um grafo é uma atribuição de cores aos seus pontos, de forma que cada ponto receba exatamente uma cor e que não haja dois pontos adjacentes com a mesma cor.

Uma classe de cor de um grafo G é um conjunto não vazio de pontos com a mesma cor, em um colorido de G. Vemos que, por definição, uma classe de cor é um subconjunto independente de G.

Um k-colorido de G é um colorido de G que usa k cores, particionando V em k classes de cor.

O número cromático de G, $\chi(G)$, é igual ao menor inteiro k, para o qual G tem um k-colorido. Observamos que o número cromático de um grafo completo é igual a n, uma vez que cada ponto seu é adjacente com todos os demais.

G é um grafo k-colorável, se pudermos colorir G com k, ou menos, cores.

G é um grafo k-cromático, se $k = \chi(G)$.

Dizemos que um k-colorido de G é um <u>colorido ótimo</u> para G, se k for igual a $\chi(G)$. Um k-colorido de G, com k no intervalo $\chi(G) < k \leq n$, é um <u>colorido aproximado</u> ou <u>não - ótimo</u> de G. É claro que, para $k > n$, não existe um k-colorido de G.

IV. APLICAÇÕES.

Grande parte do interesse existente em torno de coloração de grafos advém da existência de vários problemas práticos que podem ser solucionados pela obtenção de coloridos para grafos.

A característica comum a todos esses problemas é que se deseja distribuir eventos (ou objetos) no menor número possível de períodos (ou grupos), de modo que eventos (ou objetos) incompatíveis não fiquem juntos. São denominados problemas de escalamento ou problemas de particionamento cromático.

Formularemos, a seguir, alguns desses problemas, ressaltando que o primeiro deles, a construção de tabelas de horário, é o que tem sido mais intensamente tratado na literatura.

IV-1. Construção de Tabelas de Horário.

Em uma escola, devem-se aplicar exames finais em várias matérias. Alguns desses exames não podem ocorrer simultaneamente com outros, em vista de existirem alunos que vão participar de ambos. O problema consiste em escalar os exames no menor número possível de períodos, de tal forma que não ocorram conflitos de horário.

Os exames podem ser representados pelos pontos de um grafo, ligando-se dois pontos por uma linha, se houver um ou mais alunos que farão os dois exames correspondentes.

Um k-colorido desse grafo corresponde ao escalamento dos exames em k períodos distintos, um período para cada cor. A obtenção do número cromático do grafo corresponde à determinação do menor número de períodos em que é possível fazer tal escalamento.

Em 1966, foi publicado um programa da autoria de Peck e Williams[38], que resolvia o problema apresentado acima por um procedimento heurístico. Nos comentários desse programa, vemos que os autores já estabeleciam um paralelo com a coloração de grafos, usando a representação sugerida, com o acréscimo de um peso w_i, associado ao i-ésimo ponto, indicando o número de alunos que deveriam prestar o exame correspondente. O objetivo era

escalar os exames, de forma que não ocorressem choques de horário e que o número total de alunos em exame, num dado período, não ultrapassasse certo limite.

Trabalhos anteriores, como o de Cole[10] e o de Foxley e Lockyer[18], apesar de apresentarem critérios mais sofisticados para a construção da tabela de horário, utilizavam apenas parcialmente o enfoque de coloração de grafos.

Por outro lado, Welsh e Powell[46] e Wood[49], em 1967 e 1969, respectivamente, já explicitavam a conexão existente entre os dois problemas. Parece que foram os primeiros a se abstraírem dos detalhes e a darem ao problema de construção de tabela de horário para exames a abordagem definitiva de coloração de grafos. Analisaremos posteriormente os algoritmos que propuseram.

Antes de passarmos para as outras aplicações, queremos salientar que o título "Construção de Tabelas de Horário" é utilizado para denotar um outro problema, que começou a ser tratado por computador também na década de 60. Consiste em, dado um conjunto de classes, um conjunto de professores e uma relação que fornece o número de períodos por semana (ou por dia), em que cada professor deve atender a cada classe, construir uma tabela de horário que indique, para cada período, qual professor deverá estar com determinada classe. Um método apresentado, em 1962, por Gotlieb e revisto pelo mesmo juntamente com Csima[13], em 1964, desenvolve a tabela de horário na forma de uma matriz tridimensional S, cujo elemento s_{ijk} é 1 ou 0, conforme o professor i esteja ou não com a classe j, no período k.

A construção de tabelas para exames e a construção de tabelas de horário de escolas são dois problemas distintos. O último não é resolvido por coloração de grafos, tendo sido mencionado aqui apenas para esclarecimento.

Dempster[14] já havia percebido essa multiplicidade, tendo caracterizado três diferentes categorias de problemas, existentes sob o título de Tabelamento de Horário.

IV-2. Tabelamento da Produção.

Um certo número de tarefas com mesmo tempo de execução

devem ser realizadas. Existem, no entanto, restrições que impedem que algumas dessas tarefas estejam em execução simultaneamente. Por exemplo, devido à disponibilidade de recursos. O problema consiste em estabelecer o menor número de períodos pelos quais poderão ser distribuídas as tarefas, atendendo às condições desejadas.

Nesse caso, utiliza-se como modelo um grafo em que cada ponto representa uma das tarefas. Dois pontos são ligados por uma linha, se as tarefas correspondentes não puderem ser realizadas ao mesmo tempo.

O número cromático do grafo assim obtido é igual ao menor número possível de períodos para a realização das tarefas. Cada classe de cor, em um colorido do grafo, corresponde a um conjunto de tarefas a serem realizadas simultaneamente.

IV-3. Alocação de Freqüências.

Em uma localidade, existem k canais disponíveis para serem usados por n estações de televisão. Estações próximas umas das outras não podem usar o mesmo canal sem causar interferência. O problema consiste em alocar um canal para cada estação de modo que estações que não possam utilizar o mesmo canal recebam canais diferentes. Esse problema pode também ser considerado para estações emissoras de rádio.

Construímos um grafo em que cada ponto representa uma estação de televisão, havendo uma linha ligando dois pontos sempre que a distância entre as duas estações correspondentes for menor que um certo limite.

Um k-colorido para esse grafo corresponde a uma alocação dos k canais, atendendo às condições estabelecidas. Tal colorido só é possível se o número cromático do grafo construído for menor ou igual a k.

IV-4. Armazenamento de Produtos.

Um estoque de produtos químicos deve ser armazenado de forma que certos produtos não devem ser postos em um mesmo compartimento por motivos de segurança.

Representando cada produto por um ponto de um grafo e

ligando por uma linha todo par de pontos, cujos produtos correspondentes não possam ficar juntos, podemos encontrar o menor número de compartimentos necessários para armazenar todo o estoque, calculando o número cromático do grafo assim obtido. Além disso, um conjunto de pontos com uma mesma cor corresponderá a um conjunto de produtos químicos a ser colocado em um dado compartimento.

IV-5. Planarização de Circuitos Impressos.

O problema que vamos formular, agora, está relacionado com o projeto de circuitos impressos. Nesses circuitos existem junções em posições fixas, algumas das quais deverão ser ligadas entre si por fios de conexão impressos na superfície de uma placa feita de material não condutor. Uma vez que esses fios só podem se encontrar nas junções, quantas superfícies serão necessárias para se dispor todos os fios do circuito sem que eles se cruzem?

O problema apresentado acima poderia ser considerado também do ponto de vista de planaridade de grafos.

Para coloração, desenhamos, inicialmente, em uma folha de papel todo o circuito e definimos, em seguida, um grafo em que cada ponto representa um fio de conexão e cada linha liga dois pontos cujos fios correspondentes se cruzam no desenho.

O problema proposto fica, dessa forma, substituído pelo de encontrar o número cromático do grafo obtido. Cada classe de cor conterá os pontos correspondentes aos fios a serem dispostos em cada uma das superfícies.

IV-6. Uma Questão de Diplomacia.

Após atuar vários anos em uma embaixada e devendo retornar em breve ao seu país, um diplomata deseja promover algumas recepções para se despedir dos seus conhecidos. Entretanto, sabendo que por um motivo ou outro várias dessas pessoas não se dão muito bem e querendo evitar quaisquer encontros desagradáveis, ele resolve convidar para cada recepção um grupo de pessoas no qual não haja qualquer incompatibilidade. Quantas recepções serão necessárias para que o diplomata em questão se despeça de todos os seus conhecidos e quem deverá ser convidado

em cada uma dessas ocasiões?

Ora, basta definir um grafo com um ponto para cada pessoa conhecida do diplomata e uma linha ligando dois pontos se as pessoas correspondentes não se dão bem.

O número cromático desse grafo e as classes de cor definidas por um colorido utilizando esse número de cores correspondem à solução das duas perguntas acima formuladas.

IV-7. OUTRAS.

Apresentaremos, em seguida, mais duas aplicações de coloração de grafos, mencionadas por Hammer e Rudeanu[23].

Um centro de comutação de telefones é feito por muitos blocos, que são ligados, aos pares, por fios de várias cores. Para facilitar a detecção de erros, é desejável que quaisquer dois fios, tendo uma das extremidades no mesmo bloco, recebam cores diferentes e que o número de cores seja mínimo.

A passagem para o problema de coloração de grafos é imediata.

Uma outra aplicação, devida a Marcus e Vasiliu, surge na área de linguística. Eles introduziram um grafo, cujos pontos são as 20 consoantes da língua romena e cujas linhas são definidas da seguinte forma. Para qualquer ponto v, o conjunto A(v) consiste de todas as consoantes w, tais que existe uma palavra romena cujo primeiro grupo consonantal inclui a seqüência vw. São, ao todo, 79 linhas.

Por algumas razões lingüísticas é interessante determinar-se as classes de cor, para um colorido ótimo do grafo assim obtido.

V. CARACTERIZAÇÕES E LIMITES.

Considerando que colorir um grafo desconexo significa colorir cada um dos seus componentes separadamente, vamos nos referir, daqui por diante, apenas a grafos conexos.

Para não termos que analisar a todo instante o que aconteceria na situação trivial de o grafo não conter pontos, vamos assumir que G é um grafo não-nulo, a não ser que explicitamente declaremos o contrário.

Conforme já dissemos anteriormente e esperamos ter ilustrado no capítulo (IV), o problema central em coloração de grafos consiste em, dado um grafo qualquer G, obter seu número cromático e algum dos coloridos correspondentes.

Apresentaremos, a seguir, os casos,que infelizmente são poucos, em que determinadas características facilmente identificáveis estão presentes em um grafo, se e somente se tal grafo tem um número cromático específico.

Depois, apresentaremos resultados mais fracos que esses, em que a detecção de determinadas características implica que o número cromático de um grafo está dentro de certos limites, embora não sejam características necessárias para que isso ocorra.

Por fim, apresentaremos alguns limites obtidos teoricamente para $\chi(G)$.

V-1. Grafos 1-cromático e 2-cromático.

Gostaríamos, aqui, de resolver o problema de coloração através de outro problema cuja solução seja já conhecida ou facilmente determinável. Isto é, dada a proposição P_1 "G é um grafo k-cromático", gostaríamos de ter uma outra proposição P_2, de fácil verificação, de forma que P_1 ocorre se e somente se P_2 ocorrer.

Será mostrado no próximo capítulo que, para $k>2$, qualquer proposição P_2 que fosse equivalente a P_1 seria necessariamente tão arredia a uma verificação como a própria P_1.

Resta-nos, portanto, apresentar P_2 para os casos $k=1$ e $k=2$.

Para k=1, o problema é trivial. Qualquer grafo que tenha, pelo menos, uma linha necessitará de, no mínimo, duas cores para ser colorido, uma cor para cada um dos dois pontos incidentes com essa linha.

Podemos afirmar portanto que:

TEOREMA V-1.

G é um grafo 1-cromático, se e somente se for totalmente desconexo.

A verificação de que um grafo G qualquer é, ou não, totalmente desconexo pode ser feita em tempo constante, se m for dado, ou em tempo linear com n, no caso contrário.

A caracterização de grafos 2-cromáticos também é bastante simples, se aplicarmos o conceito de grafo bipartite.

Seja G um grafo bipartite. Podemos colorir com a cor 1 os pontos do conjunto V_1 e com a cor 2 os pontos do conjunto V_2. Isso nos dá um 2-colorido de G. Logo G é 2-colorável.

Por outro lado, seja G um grafo 2-colorável. Dado um 2-colorido de G, podemos construir uma partição $\{ V_1, V_2 \}$ de V, de acordo com as duas classes de cor definidas pelo colorido. Como cada linha de G liga um elemento de V_1 e um elemento de V_2, G é um grafo bipartite.

Isso nos leva aos seguintes resultados:

LEMA.

G é um grafo 2-colorável, se e somente se for bipartite.

TEOREMA V-2.

G é um grafo 2-cromático, se e somente se ocorrem as duas condições abaixo:

(i) G não é 1-cromático e

(ii) G é bipartite.

Um grafo é bipartite, se e somente se não possui ciclos de comprimento ímpar. (Para uma demonstração dessa equivalência, veja, por exemplo, a página 18 do Harary[24].) Logo, podemos determinar se G é bipartite, através de algum algoritmo que verifique, no caso de G ter ciclos, se o comprimento de todos eles é um número par. Em quanto tempo isso pode ser feito?

O algoritmo apresentado no quadro (V-1) descobre se um grafo G é bipartite, dando a cor 1 a um ponto de G e, em seguida, tentando atribuir a cor 2 a todos os pontos adjacentes a pontos com a cor 1 e vice-versa. Isso é feito até todos os pontos terem recebido uma das cores ou até ser encontrado algum ponto ainda sem cor, que seja adjacente a um ponto com a cor 1 e a um ponto com a cor 2. No primeiro caso, G é bipartite. No segundo, foi identificado um ciclo de comprimento ímpar e G não é bipartite.

Esse algoritmo executa o que se chama "backtracking" totalmente restringido, visitando exatamente 1 vez cada um dos pontos e cada uma das linhas do grafo, em um tempo O(m+n). Não são analisados todos os ciclos do grafo, mas apenas um conjunto de ciclos, a partir dos quais todos os outros podem ser obtidos por combinação linear. Isso é o bastante, pois a existência de um ciclo qualquer de G com comprimento ímpar implica que pelo menos um dos ciclos básicos considerados tenha comprimento ímpar.

Quadro V-1. Algoritmo que determina se G é bipartite.

```
begin                                                          L1
    comment  c1, c2,...,cn guardará a cor atribuída a          L2
             cada um dos pontos 1, 2,...,n de G.               L3
    procedure  B (v,cor); integer value (v,cor);               L4
    begin                                                      L5
        cv:=cor;                                               L6
        if cor = 1 then cor:=2 else cor: =1;                   L7
        marque v;                                              L8
        for w ε A(v) do                                        L9
            if w desmarcado then B(w,cor)                      L10
            else if cv = cw then                               L11
                begin                                          L12
                    output "G não é bipartite";                L13
                    stop                                       L14
                end                                            L15
    end B;                                                     L16
    desmarque todos os pontos;                                 L17
    c1,c2,...,cn:=0;                                           L18
    B(z,1); comment z é um ponto inicial dado;                 L19
    output  "G é bipartite"                                    L20
end                                                            L21
```

V-2. Grafos Planares.

Desde o século passado, sabe-se, devido a um outro teorema de Heawood, que cinco cores bastam para colorir qualquer grafo planar. Para uma demonstração desse fato, veja as páginas 188 e 189 da obra de Deo[15].

Por outro lado, pode ser facilmente observado que existem grafos planares tais que três cores não são suficientes para colori-los. Isso ocorre, por exemplo, com o grafo completo de quatro pontos.

Portanto, o mínimo para o qual se poderia abaixar o resultado de Heawood seria quatro cores. No entanto, foi apenas recentemente que se conseguiu fazer isso.

Hoje, já podemos anunciar o

TEOREMA V-4.
Se G é um grafo planar, então $\chi(G) \leq 4$.

Para uma explicação em linhas gerais da demonstração construída por Haken e Appel, veja o artigo de Bernhart[3].

Observemos que, além de termos restringido nosso resultado para uma classe especial de grafos, já não contamos com a força de um "se somente se". Na figura (V-1), vemos um grafo não planar que pode ser colorido com apenas 2 cores.

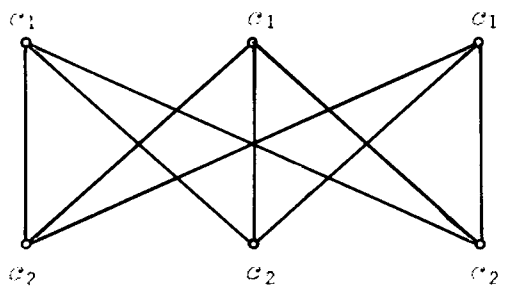

Figura V-1. Grafo não-planar 2-colorável.

Suponhamos que se deseja saber se um dado grafo G é 4-colorável. Podemos aplicar um algoritmo que em tempo linear O(n) nos mostre se G é ou não um grafo planar (Tarjan[43]). No primeiro caso, concluímos que G é 4-colorável. No caso de G não ser planar, nada poderemos concluir.

TEOREMA V-5.
Seja G um grafo planar. Se G tem menos de quatro triân-

gulos, então $\chi(G) \leq 3$.

O teorema acima foi demonstrado por Grünbaum, sendo referenciado na página 131 do Harary[24].

Novamente, apresentamos um exemplo, na figura (V-2), mostrando que a recíproca não é verdadeira.

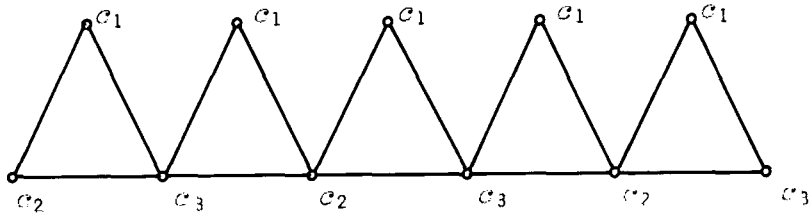

Figura V-2. Grafo planar 3-colorável, com mais de quatro triângulos.

Resultados do tipo "se então", como os que acabamos de ver, são de interesse estritamente teórico.

V-3. Limites Inferiores para $\chi(G)$.

Se G contém um subgrafo completo com k pontos, obviamente o seu número cromático deverá ser maior ou igual a k. O número de pontos no maior subgrafo completo de G é, portanto, um limite inferior bastante intuitivo para G.

Cabe notar, no entanto, que existem grafos nos quais esses dois valores se distanciem tanto quanto se queira.

Seja r o número de pontos contidos no maior conjunto independente de G. Então, r será também o maior número possível de pontos a receberem a mesma cor em um colorido de G. Conclui-se que $\lceil \frac{n}{r} \rceil$ é um limite inferior para $\chi(G)$, onde a notação $\lceil x \rceil$ significa o maior inteiro menor ou igual a x.

Dois outros limites inferiores para o número cromático de G, relacionando-o com o número cromático do grafo complementar de G, $\bar{G}$, são: $\chi(G) \geq \lfloor 2\sqrt{n} \rfloor - \chi(\bar{G})$ e $\chi(G) \geq n/\chi(\bar{G})$, onde $\lfloor x \rfloor$ significa o menor inteiro maior ou igual a x.

Esses resultados foram desenvolvidos por Nordhaus e Gaddum, estando demonstrados, por exemplo, na página 129 do Harary[24].

V-4. Limites Superiores para $\chi(G)$

O primeiro limite superior em que se pode pensar para o número cromático de G é o próprio número n, uma vez que não se podem gastar mais cores do que o número de pontos que existem para serem coloridos. Entretanto, é óbvio que $\chi(G)$ e n podem estar tão distantes entre si quanto se queira.

O limite seguinte relaciona o número cromático com o maior grau dos pontos de G, tendo sido calculado por Brooks[6]:

$$\chi(G) \leqslant \max_{1 \leqslant i \leqslant n} \{ d_i \} + 1,$$

onde d_i é o grau do ponto v_i.

Um limite superior melhor que esse, também baseado nos graus dos pontos de G, foi apresentado por Szekeres e Wilf[41]:

$$\chi(G) \leqslant 1 + \max_{v_i \in H} (\min)(d_i)),$$

onde H é qualquer subgrafo induzido por um subconjunto de pontos de G e d_i é o grau de v_i em H. Uma demonstração mais simples do que a original, para o limite que acabamos de ver, é fornecida por Bondy[5].

Limites superiores também foram apresentados por Nordhaus e Gaddum, relacionando $\chi(G)$ com $\chi(\bar{G})$:

$$\chi(G) \leqslant n + 1 - \chi(\bar{G}) \quad \text{e} \quad \chi(G) \leqslant \lceil (\tfrac{n+1}{2})^2 \rceil / \chi(\bar{G})$$

Para uma demonstração, veja Harary[24].

VI. COMPLEXIDADE.

Em 1972, a obtenção do número cromático de um grafo qualquer foi incluída, por Karp[26], em uma lista de problemas NP-completos. Iniciaremos esse capítulo expondo como esse fato predestina qualquer algoritmo, que venha a ser feito para resolver o problema, a ser um algoritmo ruim. Em seguida, explicaremos a demonstração construída por Karp.

Veremos, ainda, que, mesmo passando do problema de k-colorabilidade de um grafo qualquer para o subproblema de 3-colorabilidade de uma classe restrita de grafos - ou afrouxando a exigência de se obter o número cromático exato de G para que se obtenha uma estimativa que seja, no pior dos casos, o dobro do valor exato - continuamos dentro do conjunto NP-completo.

VI-1. Conjuntos P, NP e NP-completo.

Para uma definição formal dos conjuntos de linguagens P, NP e NP-completo, indicamos, por exemplo, o capítulo 10 do livro de Aho et al[1].

Informalmente, sabemos que fazem parte de P todos os problemas para os quais existe algum algoritmo que forneça a solução em tempo polinomial. Isto é, o tempo gasto na execução de tal algoritmo é uma função polinomial do tamanho do problema.

Em NP estão todos os problemas para os quais existe algum algoritmo não determinístico, que forneça a solução em tempo polinomial. Um algoritmo não determinístico é um processo que, ao ser executado, pode atingir situações em que existem várias alternativas para prosseguimento. Nesses casos, o processo gera cópias de si mesmo, junto com as quais passa a tratar, simultaneamente, cada uma das alternativas. Cópias das cópias podem vir a ser geradas, e todas continuam em execução até que uma das cópias forneça a solução do problema, quando então termina todo o processo. Algoritmos não-determinísticos são abstrações e sua implementação é feita através de algoritmos que executam todas as combinações possíveis de alternativas, por meio da técnica de "backtracking" (veja como isso é feito no artigo de Floyd[17]),em geral gastando, para isso, um tempo exponencial

em relação ao tamanho do problema. Podemos mostrar que um problema pertence a NP, exibindo um algoritmo não-determinístico, que o resolva em tempo polinomial.

P é um subconjunto de NP, i.e., $P \subseteq NP$. Não se sabe se P=NP. Uma classe de problemas chamados NP-difíceis desempenha um papel de suma importância para a determinação dessa igualdade. Sabe-se que, se existir um algoritmo que resolva qualquer um deles em tempo polinomial, então existem algoritmos com tempo polinomial para todos os problemas pertencentes a NP, ou seja, P=NP.

Formam o conjunto NP-completo todos os problemas que, além de pertencerem a NP, são NP-difíceis.

Já foram identificados centenas de problemas no conjunto NP. O fato de muitos cientistas experientes terem se empenhado no estudo de alguns deles, sem obter uma solução polinomial, constitui uma evidência de que o futuro dos problemas NP-completos é continuarem sendo resolvidos por algoritmos exponenciais.

VI-2. K-colorabilidade ε NP-completo.

Para provar que um problema é (ou pertence ao conjunto) NP-completo devemos mostrar que tal problema está em NP e que qualquer problema NP-completo pode ser transformado, em tempo polinomial, no problema dado. Por definição, todos os problemas NP-completos podem ser transformados uns nos outros. Por isso, é suficiente mostrarmos que um problema NP-completo pode ser polinomialmente transformado no problema dado, ficando a segunda condição acima satisfeita por transitividade.

O primeiro problema reconhecido como NP-completo foi o de "satisfiability", que consiste em, dada uma expressão booleana, saber se a mesma é "satisfiable", ou seja, se existe alguma atribuição de 0's e 1's às variáveis da expressão, de forma que seu resultado seja igual a 1.

Essa etapa inicial é devida a Cook[11]. Ele mostrou, ainda, que, dada uma expressão booleana E, pode-se obter, em tempo proporcional ao seu comprimento, uma outra expressão booleana E', contendo, no máximo, três variáveis em cada fator, de modo que E' é "satisfiable", se e somente se a expressão original E

for "satisfiable". Com isso, Cook transformou o problema de "satisfiability" no problema denominado "3-satisfiability", o que implica que esse último também é NP-completo.

O que mais nos interessa aqui é um resultado obtido posteriormente por Karp[26,27] e enunciado no seguinte teorema:

TEOREMA VI-1.

O problema de "3-satisfiability" é polinomialmente transformável no problema de k-colorabilidade de um grafo qualquer.

Vejamos como isso pode ser feito.

Seja F uma expressão booleana com k variáveis e t fatores, sendo que cada fator tem, no máximo, três variáveis. Construímos, a partir de F e em tempo polinomial em max (n,t), um grafo G com 3k+t pontos, sendo que G pode ser colorido com k+1 cores, se e somente se F é "satisfiable".

Mais especificamente, sejam $x_1, x_2, \ldots, x_k$ as variáveis de F e chamemos de $F_1, F_2, \ldots, F_t$ os seus fatores. Assumimos $k \geqslant 4$, uma vez que para três ou menos variáveis a expressão pode ser resolvida em tempo polinomial. Tomaremos como pontos de G:

- x_i, $\bar{x}_i$ e v_i, para $1 \leqslant i \leqslant k$ e
- F_i, para $1 \leqslant i \leqslant t$.

Serão as linhas de G:

- (v_i, v_j) para todo $i \neq j$,
- (v_i, x_j) e $(v_i, \bar{x}_j)$ para todo $i \neq j$,
- $(x_i, \bar{x}_i)$ para $1 \leqslant i \leqslant k$,
- (x_i, F_j) se x_i não está no fator F_j e
- $(\bar{x}_i, F_j)$ se $\bar{x}_i$ não está no fator F_j.

Na figura (VI-1) vemos o grafo G, formado a partir da expressão booleana $F = (x_1+x_2)\ (\bar{x}_1+x_3)$.

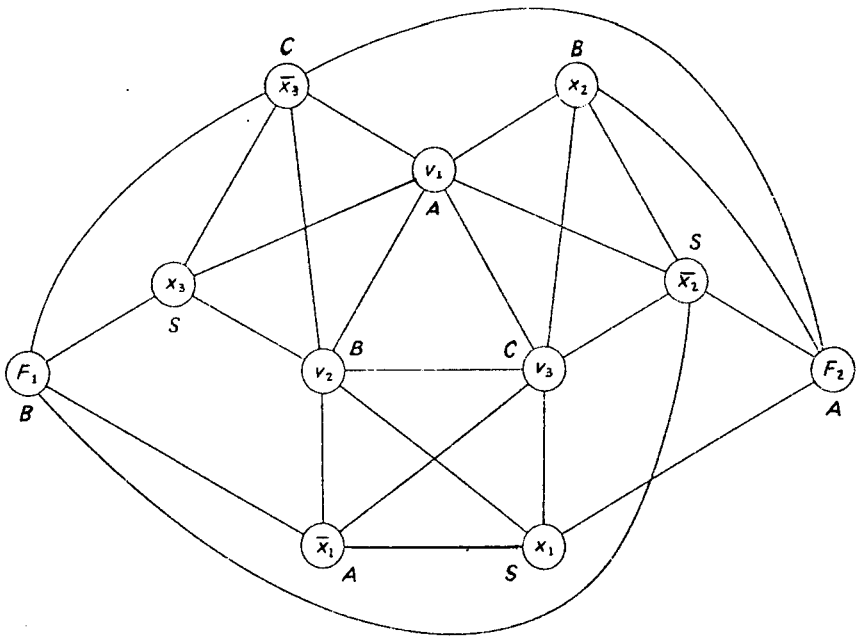

Figura VI-1. Grafo obtido da expressão $(x_1+x_2)(\bar{x}_1+x_3)$.

Os pontos $v_1, v_2, \ldots, v_k$ formam um subgrafo completo de G, gastando k cores, $c_1, c_2, \ldots, c_k$, para serem coloridos. Como x_i e $\bar{x}_i$ estão ambos ligados a todos os v_j, exceto para j=i, um deles recebe a mesma cor que v_i e o outro terá que receber uma nova cor, que chamamos cor especial, c_e. Uma vez que $k \geqslant 4$ e cada fator tem, no máximo, três variáveis, para cada j existe um i, tal que F_j é adjacente a ambos x_i e $\bar{x}_i$, de onde conclui-se que F_j não pode ser colorido com a cor especial.

Associamos à cor especial o valor 0 e às demais cores o valor 1.

O grafo G poderá ser colorido com as k+1 cores definidas, se e somente se cada fator F_j contiver uma variável y, tal que $\bar{y}$ recebeu a cor especial, pois nesse caso F_j não é adjacente aos pontos com a mesma cor que y, podendo receber tal cor. Mas, isso corresponde a exigir que em cada fator exista uma variável que receba o valor 1, o que torna a expressão F "satisfiable".

Acabamos de ver que o problema de "3-satisfiability" po

de ser transformado, em tempo polinomial, no problema de colorabilidade de grafos. Para demonstrarmos que esse último é NP-completo, falta, ainda, determinar se ele está em NP. Essa é a parte mais fácil. No quadro (VI-1) apresentamos um algoritmo não-determinístico, que verifica a k-colorabilidade de um grafo qualquer em tempo polinomial.

Logo, k-colorabilidade está em NP e também em NP-completo.

Quadro VI-1. Algoritmo não-determinístico para k-colorir G.

```
begin                                                          L1
    comment  Q é a matriz de adjacência de                     L2
             G, q_ij = 1 se e somente se v_i e v_j             L3
             são adjacentes;                                   L4
    for  i:=1 to n do c_i:= escolha (1,2,...,k);               L5
    comment  escolhidas as cores, falta                        L6
             verificar se foi obtido um                        L7
             k-colorido;                                       L8
    sucesso:=true;                                             L9
    for  i:=2 to n e j:=1 to (i-1) do                          L10
         if (q_ij=1 and  c_i = c_j)                            L11
         then  sucesso = false;                                L12
    if   sucesso then  output  (c_1,c_2,...,c_n)               L13
end                                                            L14
```

VI-3. O Que Fazer.

Os resultados vistos na seção anterior "sugerem fortemente" ser impossível descobrir um bom algoritmo para a obtenção do número cromático de G, isto é, um algoritmo que resolva o problema em um tempo limitado por uma função polinomial do número de pontos de G.

Encontramo-nos, portanto, frente a um problema computa-

cionalmente intratável e temos que adotar uma das duas seguintes abordagens.

1) Projetamos um algoritmo para obter um colorido de G com exatamente k cores, onde $k = \chi(G)$. Limitamos o tamanho dos grafos a serem analisados pelo algoritmo, visto termos conhecimento de que o tempo de execução do mesmo pode, no pior caso, crescer em uma razão exponencial com o aumento do número de pontos de G. Devemos procurar algoritmos cujo tempo de execução seja limitado por uma função exponencial de ordem relativamente baixa e cujo espaço de memória requerido não cresça exponencialmente com o tamanho do grafo. E isso é tudo que podemos fazer. Estudaremos os algoritmos de coloração exata conhecidos no capítulo(VIII).

2) Seja N(G) o número de cores usadas por um algoritmo de coloração N quando aplicado a G. Nessa abordagem, abrimos mão da exigência $N(G)=\chi(G)$. Procuramos um algoritmo eficiente, que garanta que N(G) esteja próximo de $\chi(G)$.

Em vários outros problemas NP-completos adotou-se essa segunda solução. Em coloração de grafos muitos algoritmos foram desenvolvidos, mesmo antes do trabalho de Karp, que obtém colorações aproximadas. Quase todos foram motivados pelo problema de construção de tabelas de exames, que comumente envolve um grafo com muitos pontos e que admite sem grande prejuízo uma solução quase ótima.

Entretanto, para nenhum desses algoritmos foi possível garantir um valor N(G) próximo de $\chi(G)$, para qualquer G. Isso só foi conseguido no caso restrito de coloração de grafos planares.

Veremos, na seção seguinte, que também não é provável que se desenvolvam bons algoritmos aproximativos para coloração de grafos quaisquer. No próximo capítulo estudaremos alguns dos algoritmos aproximativos existentes.

VI-4. Qualidade das Aproximações.

Qualquer que seja um algoritmo aproximativo para coloração, existe sempre um conjunto de grafos tal que a razão $N(G)/\chi(G)$ não é limitada.

O seguinte teorema, demonstrado por Garey e Johnson[21], indica que muito dificilmente se chegará a um algoritmo em tempo polinomial, que garantidamente utilize, no máximo, um número de cores igual ao dobro do número cromático, quando aplicado a um grafo qualquer. Pois isso implicaria a existência de um algoritmo exato de coloração em tempo polinomial.

TEOREMA VI-2.

Se, para alguma constante $r < 2$ e constante d, existir algum algoritmo de coloração de grafos N, com tempo limitado por um polinômio e que garanta $N(G) \leq r.\ \chi(G) + d$, então existe um algoritmo de coloração de grafos $\bar{N}$, com tempo limitado por um polinômio, e que garanta $\bar{N}(G) = \chi(G)$.

O resultado acima é surpreendente, pois de uma vez desacredita todos os algoritmos aproximativos já existentes e desestimula qualquer esforço no sentido de se obterem novos e melhores algoritmos.

Ao apresentarmos os algoritmos aproximativos no capítulo seguinte, mostraremos que para qualquer um deles existem situações em que a solução fornecida é arbitrariamente ruim.

Com base no caráter intercambiável dos problemas NP-completos, poderíamos propor que se aproveitassem métodos aproximativos desenvolvidos para um dos problemas, para resolver outros. No caso específico de coloração, vimos que já se conhece uma fórmula para transformar "3-satisfiability" em k-colorabilidade.

Dada uma expressão booleana F, contendo k variáveis com no máximo três variáveis por fator, construiríamos um grafo G, como na seção (VI-2), e procuraríamos uma atribuição da cor especial c_e aos pontos x_i e $\bar{x}_i$, através de um algoritmo aproximativo, que resultasse em um k+1-colorido para G. Caso se obtivesse tal colorido concluir-se-ia a "satisfiability" de F. No caso contrário, no entanto, não saberíamos, baseados no resultado conseguido, qual seria a chance de F ser ou não "satisfiable".

A necessária obscuridade dos algoritmos aproximativos de coloração, sugerida pelo teorema (VI-2), torna-os inúteis para aplicação da idéia acima proposta.

VI-5. Subproblemas NP-completos.

Uma demonstração construtiva, como a que expusemos em (VI-2), foi formulada por Garey, Johnson e Stockmeyer[20], mostrando que o problema de "3-satisfiability" pode ser transformado no de 3-colorabilidade de grafos quaisquer em tempo polinomial. Esse problema consiste em, dado G qualquer, determinar se G é 3-colorável. Apesar de ser um subproblema da k-colorabilidade e intuitivamente parecer mais fácil, a 3-colorabilidade também é NP-completo.

Citaremos, a seguir, mais dois resultados semelhantes a esse, ambos produzidos também por Garey et al. Na figura (VI-2), mostramos um esquema dos problemas NP-completos mencionados. Uma seta partindo do problema P1 para o problema P2 significa que na demonstração de que P2 é NP-difícil usou-se a transformação de P1 em P2.

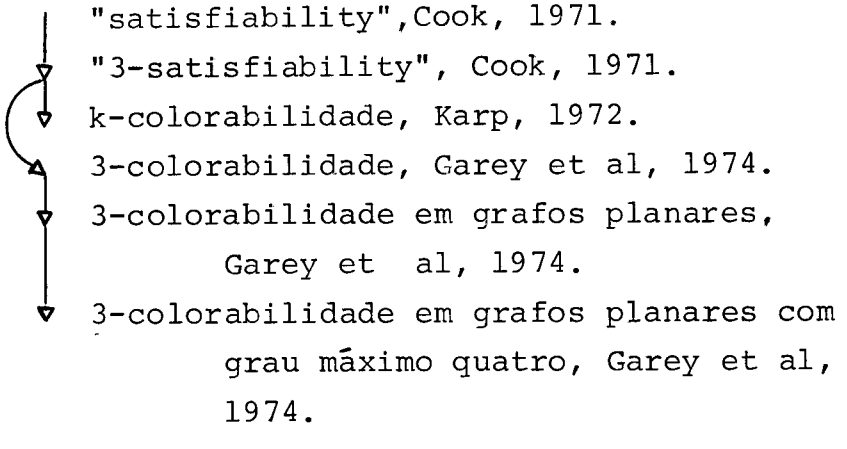

Figura VI-2. Alguns problemas NP-completos.

Seja G um grafo planar. O problema de se determinar se G é 3-colorável é NP-completo.

Seja G um grafo planar, cujos pontos apresentam grau máximo 4. Então, o problema de se determinar se G é 3-colorável também é NP-completo.

VII. APROXIMAÇÕES.

A seguir, apresentaremos várias heurísticas, desenvolvidas para fornecer uma estimativa do número cromático de um grafo e um colorido do mesmo, utilizando esse número de cores.

Alguns desses algoritmos foram testados na prática para resolverem o problema da tabela de horário de exame, para grafos com mais de 700 pontos (Williams[47]).Conforme pode ser observado em uma tabela de tempo de execução (Tarjan[43]), o cálculo do número cromático exato de grafos com mais de 20 pontos pode vir a ser irremediavelmente lento, utilizando-se um algoritmo de complexidade exponencial. E, ao que parece, essa será sempre a única forma de se obterem coloridos ótimos.

Todos os algoritmos aproximativos encontrados na literatura têm complexidade polinomial, na maioria das vezes igual a $O(n+m)$.

Foram reunidos em um mesmo grupo, os algoritmos que dispensam um tratamento semelhante aos pontos do grafo.

Começaremos pelos coloridos sequenciais, cuja idéia foi formalizada por Matula et al[34], em 1972.

Em seguida, veremos algoritmos que efetuam coloridos por classe, devidos a Welsh e Powell[46] e Williams[48], e publicados em 1967 e 1974, respectivamente. Será demonstrado que coloridos por classes são equivalentes a coloridos sequenciais, no sentido de que as classes de cor, produzidas por ambos para um dado grafo, são iguais.

Um algoritmo realmente peculiar será apresentado, que fornece coloridos aos pares, para um grafo G qualquer. Ele foi criado por Wood[49], em 1969, que verificou experimentalmente a superioridade do procedimento sobre os existentes, para grafos grandes e de alta densidade de linhas.

Finalmente, mostraremos algoritmos que produzem coloridos em "batch", idealizados por Lipton e Miller[32], em 1978, para uma classe específica de grafos - os grafos planares.

Encerraremos o capítulo, mostrando como fabricar grafos, para os quais os algoritmos aproximativos não funcionam bem (Mitchem[35]).

VII-1. Coloridos Seqüenciais.

O procedimento mais intuitivo para se colorir um grafo consiste em tomar os pontos um a um e tentar colocá-los nas classes de cor já iniciadas. Caso isso não seja possível, abre-se uma nova classe de cor com o ponto em questão.

Formalizando essa idéia, damos a definição de colorido seqüencial, apresentada por Matula, Marble e Isaacson[34].

DEFINIÇÃO VII-1.

Um colorido seqüencial de G, dada uma ordenação - $v_1, v_2, \ldots, v_n$ de seus pontos, é um k-colorido de G, no qual as cores $1,2,\ldots,k$ são atribuídas da seguinte forma:

(i) A cor 1 é atribuída a v_1; com isso, $<\{v_1\}>$ recebe um 1-colorido;

(ii) $<\{v_1, v_2, \ldots, v_{i-1}\}>$ recebeu um j-colorido; atribuem-se as mesmas cores a $v_1, v_2, \ldots, v_{i-1}$ em $< \{ v_1, v_2, \ldots, v_i \} >$ e atribui-se a v_i a cor r, onde $r \leq j+1$ é o menor inteiro positivo para o qual a cor r não ocorre em pontos adjacentes a v_i em $< \{v_1, v_2, \ldots, v_i\} >$. Isso fornece um j-colorido de $< \{ v_1, v_2, \ldots, v_i \} >$ para $r \leq j$, ou um (j+1)-colorido para $r = j+1$.

No quadro (VII-1), temos uma implementação do colorido seqüencial.

Quadro VII - 1 . Algoritmo aproximativo S1.

```
begin                                                          L1
    c_1 := 1;                                                  L2
    k := 1;                                                    L3
    for i:= 2 until n do                                       L4
    begin                                                      L5
        r:= min (j | A_i ∧ C_j = φ) ;                          L6
        c_i:= r; comment colore v_i com a cor r;               L7
        k: = max (k,r);                                        L8
    end                                                        L9
end                                                            L10
```

NOMENCLATURA.

Na descrição dos algoritmos utilizaremos, sempre que necessário, as seguintes variáveis:

G, o grafo dado;

V, o conjunto de pontos de G;

n, a cardinalidade de V;

$v_1, v_2, \ldots, v_n$ ou $1,2,\ldots,n$, os elementos de V;

$c(v_i)$ ou c_i, a cor atribuída ao ponto v_i; no início, $c_i=0$;

$C(v_i)$ ou C_i, a i-ésima classe de cor, contendo os pontos que receberam a cor i; assumimos que no início do procedimento, $C_i = \phi$, para $1 \leqslant i \leqslant n$;

$A(v_i)$ ou A_i, a lista de adjacência de v_i;

$d(v_i)$ ou d_i, o grau do ponto v_i;

k, a estimativa do número cromático de G, a cada instante.

COMPLEXIDADE DE S1.

É possível calcular-se o valor de r, na linha L6, em tempo O(n+m). Basta verificar, para cada $w \in A_i$, a cor c(w). O número de consultas será igual a $|A_i|$ para cada iteração, dando um total de 2m.

Portanto, a complexidade de S1 é O(n+m).

DESEMPENHO DE S1.

Observemos o grafo apresentado na figura (VII-1). Se os pontos do grafo estiverem ordenados como em (a), o algoritmo S1 fornecerá k=4. Se os pontos estiverem ordenados como em (b), teremos k=3.

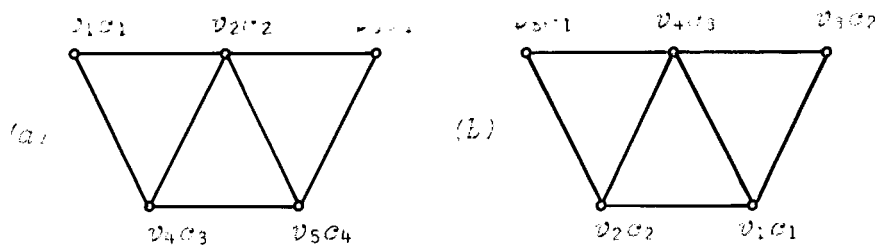

Figura VII-1. Aplicações do algoritmo S1.

Portanto, o desempenho de S1 depende da ordem em que fo

rem considerados os pontos do grafo.

ORDENANDO OS PONTOS.

Os algoritmos seqüenciais que veremos em seguida procuram arrumar os pontos, numa tentativa de obterem melhores coloridos.

Existem n! formas distintas de se disporem os n pontos de um grafo G. Se, para cada uma dessas disposições, executássemos o algoritmo S1, pelo menos em uma das execuções obteríamos um colorido exato de G. Isso ocorreria, por exemplo, se os pontos estivessem ordenados pelas classes de cor de qualquer colorido ótimo de G.

Considerando que n! é um valor proibitivo, uma idéia seria selecionar-se uma amostra de tamanho n. Executar-se-ia, então, n vezes o algoritmo de coloração seqüencial, utilizando-se cada uma das ordenações da amostra, e escolher-se-ia o melhor colorido resultante.

Recorrendo novamente à intuição, parece-nos que os pontos mais difíceis de serem coloridos em um grafo qualquer são os que possuem maior número de pontos adjacentes. É razoável que procuremos iniciar o colorido por esses pontos mais difíceis.

DEFINIÇÃO VII-2.

Uma ordenação "largest-first" dos pontos de G é uma ordenação $v_1, v_2, \ldots, v_n$ tal que $d_i \geqslant d_{i+1}$ para $1 \leqslant i \leqslant p$.

O algoritmo S2, de autoria de Matula et al[34], executa um colorido seqüencial de G para uma ordenação "largest-first " de seus pontos. Veja o quadro (VII-2).

Quadro VII-2. Algoritmo aproximativo S2.

begin	*L1*
ordene V de acordo com uma ordenação	*L2*
"largest-first";	*L3*
execute S1;	*L4*
end	*L5*

COMPLEXIDADE DE S2.

Pode-se efetuar a classificação, linhas L2 - L3, pelo método de "bucket sort", que será O(n+m), ou simplesmente O(n), se forem conhecidos os graus dos pontos. Como já vimos, a complexidade da linha L4 é O(n+m). Portanto, a complexidade de S1 é O(n+m).

DESEMPENHO DE S2.

Ao ser colorido o i-ésimo ponto do grafo, terão sido empregadas, no pior dos casos, i cores distintas. Seja $t_i = 1 + d_i$, o grau do i-ésimo ponto, v_i, mais 1. Então, na pior situação, v_i foi colocado na t_i-ésima classe de cor, uma vez que cada uma das classes anteriores pode conter um dos pontos adjacentes a v_i. Se t_i for menor que i, já sabemos que v_i não usará a i-ésima cor.

Portanto, min { i, $1+d_i$ } é um limite superior para o número de classes de cor já iniciadas após ter sido colorido o i-ésimo ponto. Escolhendo o maior desses valores para todos os pontos do grafo, obtemos:

$$k \leq \max_{1 \leq i \leq n} \min \{ i, 1+d_i \},$$

onde k é o valor estimado pelo algoritmo S2 para o número cromático de G.

O grafo mostrado na figura (VII-2) é denominado grafo estrela com n pontos. É claro que S2 atribuirá a um grafo estrela o colorido ótimo utilizando duas cores. Para esse caso, o limite superior acima atua muito melhor que o limite estabelecido anteriormente por Brooks[6], $\chi(G) \leq \max_{1 \leq i \leq n}\{d_i\} + 1$.

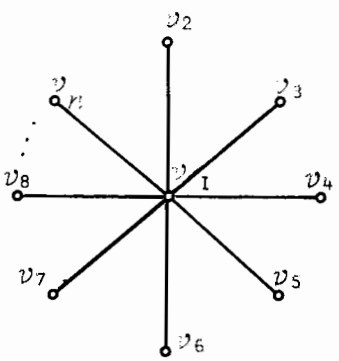

Figura VII-2. Grafo estrela.

Observemos que o maior grau do grafo estrela pode ser aumentado à vontade, sem que com isso altere o número cromático do grafo.

Como pode haver mais de um ponto de mesmo grau em um grafo G qualquer, uma ordenação "largest-first" não é única, podendo-se obter coloridos distintos para diferentes ordenações utilizadas.

Matula[33] demonstrou que, entre as n! possíveis ordenações dos pontos de G, qualquer ordenação do tipo definido a seguir minimiza o limite superior de k, dado por $\max_{1 \leq i \leq n} \{1+d_i\}$, sendo d_i tomado em $<\{v_1, v_2, \ldots, v_i\}>$

DEFINIÇÃO VII-3.

Uma ordenação "smallest - last" dos pontos de G é uma ordenação $v_1, v_2, \ldots, v_n$ tal que $d_i = \min_{1 \leq i \leq j} dj$, sendo d_i e d_j tomados no subgrafo induzido $< \{v_1, v_2, \ldots, v_i\} >$, para $1 \leq i \leq n$.

Uma ordenação desse tipo pode ser construída da seguinte maneira:

(i) escolha para v_n um ponto de menor grau em G;

(ii) para i = n-1, n-2,...,2,1, escolha para v_i um ponto de menor grau no subgrafo $<V - \{ v_n, v_{n-1}, \ldots, v_{i+1} \}>$.

O algoritmo S3, quadro (VII-3), desenvolvido também por Matula et al, é um colorido seqüencial que trata os pontos na ordem definida acima.

Quadro VII-3. Algoritmo aproximativo S3.

```
begin                                              L1
     ordene V de acordo com uma ordenação          L2
     "smallest - last";                            L3
     execute S1;                                   L4
end                                                L5
```

COMPLEXIDADE DE S3.

A classificação "smallest-last" também pode ser feita em O(n+m) tempo. A idéia é ordenar inicialmente os pontos de V, por "bucket sort". Em seguida, para cada menor ponto retirado,

subtrair 1 do grau dos pontos adjacentes, o que pode exigir que se faça, no máximo, uma troca de pontos, por subtração, para manter a lista ordenada.

Logo, a complexidade de S3 é $O(n+m)$.

DESEMPENHO DE S3.

O algoritmo S3 fornece um k-colorido de G, com $k < 1 + \max_H \min_{v_i \in V(H)} \{d_i\}$, onde d_i é tomado em H, um subgrafo qualquer de G. Szekeres e Wilf[41] já haviam encontrado esse limite, partindo dos autovalores da matriz de adjacência de G. Para grafos com grande variedade de graus, ele se mostrou melhor do que o limite superior estabelecido pelo algoritmo S2.

MELHORANDO ATRAVÉS DA TROCA BICROMÁTICA.

Ao acrescentar o ponto v_i ao subgrafo j-colorido $< \{ v_i, v_2, \ldots, v_{i-1} \} >$, a (j+1) - ésima cor só é necessária se v_i é adjacente a pontos com cores 1,2,...,j. Ora, se nenhum desses pontos faz parte de um subgrafo completo de G com j pontos, pode ser possível trocar a cor de alguns deles, de forma a se obter um outro j-colorido de $<\{v_1, v_2, \ldots, v_{i-1}\}>$ tal que o número de cores distintas nos pontos adjacentes a v_i seja, no máximo, j-1, liberando, portanto, uma cor para v_i. Isso é feito tentando-se aplicar uma troca bicromática em momentos críticos do processo de coloração.

DEFINIÇÃO VII-4.

Dado um grafo G e um k-colorido de G utilizando as cores 1,2,...,k, seja C_i a classe de cor definida pela cor i. Para i ≠ j, o subgrafo ij-cromático de G é o subgrafo $<C_i \cup C_j>$ e um ij-componente de G é um componente de $<C_i \cup C_j>$.

DEFINIÇÃO VII-5.

Uma troca bicromática em um grafo G k-colorido é uma troca de cores de todos os pontos de um ij-componente de G para algum i ≠ j; i,j < k.

C. Berge[2] sugere uma heurística de coloração em que, a partir de um colorido aproximado de G, procura-se eliminar uma das cores de cada vez, aplicando-se trocas bicromáticas.

Matula et al[34] introduziram mais esse refinamento na im

plementação de coloridos seqüenciais.

DEFINIÇÃO VII-6.

Um colorido seqüencial com troca de G, correspondente a uma ordenação $v_1,v_2,\ldots,v_n$ de seus pontos, é um k-colorido de G, utilizando as cores 1,2,...,k determinadas recursivamente como se segue:

(i) A cor 1 é atribuída a v_1, o que fornece um 1-colorido de $<\{ v_1 \}>$;

(ii) Se $<\{v_1,v_2,\ldots,v_{i-1}\}>$ recebeu um j-colorido usando cada uma das cores 1,2,...,j e se r é o menor inteiro positivo tal que a cor r não ocorre em pontos de $<\{ v_1,v_2,\ldots,v_{i-1} \}>$ que são adjacentes a v_i em G, então:

Para r < j, atribui-se em $<\{v_1,v_2,\ldots,v_i\}>$ a mesma cor recebida anteriormente para os pontos $1,2,\ldots,v_{i-1}$ e a cor r para o ponto v_i, obtendo-se, assim, um j-colorido de $<\{v_1,v_2,\ldots,v_i\}>$;

Para r = j+1, seja $T \subseteq \{ 1,2,\ldots,j \}$ o conjunto de cores tal que $\alpha \in T$ se e somente se exatamente um ponto adjacente a v_i em $<\{v_1,v_2,\ldots,v_i\}>$ recebeu cor α em $<\{v_1,v_2,\ldots,v_{i-1}\}>$. Se, para algum $\alpha,\beta \in T$, $\alpha \neq \beta$, um $\alpha\beta$-componente de $<\{v_1,v_2,\ldots,v_{i-1}\}>$ tem exatamente um ponto adjacente a v_i em $<\{v_1,v_2,\ldots,v_{i-1}\}>$ então efetua-se uma troca bicromática no $\alpha\beta$-componente de $<\{v_1,v_2,\ldots,v_{i-1}\}>$ e atribui-se em $<\{v_1,v_2,\ldots,v_i\}>$ as mesmas cores do novo colorido para $v_1,v_2,\ldots,v_{i-1}$ e a cor que ficou disponível, α ou β, para v_i, obtendo-se um j-colorido de $<\{v_1,v_2,\ldots,v_i\}>$
senão, atribui-se em $<\{v_1,v_2,\ldots,v_i\}>$ aos pontos $v_1,v_2,\ldots,v_{i-1}$ as mesmas cores que tinham em $<\{v_1,v_2,\ldots,v_{i-1}\}>$ e atribui-se a cor r para o ponto v_i, obtendo-se, assim, um (j+1)-colorido de $<\{v_1,v_2,\ldots,v_i\}>$.

No quadro (VII-4), mostramos uma implementação do colorido seqüencial com troca, algoritmo S4. Como em qualquer colorido sequencial, a ordem em que os pontos são analisados é importante para a obtenção de um bom colorido. Isso será considerado no algoritmo S5, quadro (VII-5), proposto por Matula et al[34], que consiste em um colorido seqüencial com troca, para uma ordenação "smallest-last" dos pontos de G.

Quadro VII-4. Algoritmo aproximativo S4.

```
begin                                                              L1
    k:=1;                                                          L2
    for i:=2 until n do                                            L3
    begin                                                          L4
        r:= min { j | A_i ∧ C_j := φ };                             L5
        if r=k+1 then                                              L6
        begin                                                      L7
            T := { α t.q. |A_i ∧ C_α| = 1, para 1≤α≤k } ;           L8
            if (∃α,β∈T t.q. H, um αβ-componente,                   L9
                    tem exatamente 1 ponto v_p t.q.                L10
                    v_p ∈ A_i e p < i ) then                       L11
            begin                                                  L12
                    efetue uma troca bicromática em H;             L13
                    r:= cor de v_p;                                L14
            end                                                    L15
            else k:=k+1                                            L16
        end                                                        L17
        c_i:=r                                                     L18
    end                                                            L19
end                                                                L20
```

COMPLEXIDADE DE S4.

O cálculo de r, linha L5, pode ser feito em O(n+m), como já vimos para o algoritmo S1. O mesmo vale para a construção de T, linha L8.

Os subgrafos αβ-cromáticos podem ser construídos simultaneamente, gastando para isso um tempo O(n+m). Além disso, os αβ-componentes podem ser identificados, em um grafo com n' pontos e m' linhas, em O(n'+m'), sendo que cada ponto pode ir sendo verificado quanto às propriedades desejadas. Logo, para cada iteração, o teste L9-L11 pode tomar um tempo limitado por O(n+m), o que dá um total de O(n(n+m)).

A troca bicromática, linha L13, pode ser efetuada em O(n).

Concluímos que a complexidade do algoritmo S4 é - O(n(n+m)).

Quadro VII-5. Algoritmo aproximativo S5.

begin	*L1*
ordene V de acordo com uma ordenação	*L2*
"smallest-last";	*L3*
execute S4	*L4*
end	*L5*

COMPLEXIDADE DE S5.

A complexidade de S5 é a mesma de S4, ou seja, - O(n(n+m)).

DESEMPENHO DE S5.

Além de, para um mesmo grafo, esse algoritmo poder produzir diferentes colorações devido a ordenações "smallest-last" distintas, ele é também influenciado pelo particular αβ-componente escolhido para se efetuar a troca bicromática.

A aplicação de S5 a grafos randômicos de até 100 pontos mostrou uma atuação significativamente melhor do que os coloridos seqüenciais sem troca, mas o tempo gasto foi muito maior.

Quanto ao limite superior para o número cromático obti

do, os autores demonstram que o algoritmo A4 colore qualquer grafo planar com cinco ou menos cores e qualquer grafo planar sem subgrafo de grau mínimo 5, com quatro ou menos cores.

VII-2. Coloridos por Classe.

Os algoritmos que veremos nessa seção apresentam a característica de comporem uma classe de cor, de cada vez, verificando-se todos os pontos ainda não coloridos quanto a ser possível incluí-los na mesma. Só então é que se passa a considerar a próxima classe de cor, repetindo-se o procedimento até que todos os pontos tenham sido coloridos.

Os coloridos por classe, como os coloridos seqüenciais, também acessam seqüencialmente os pontos do grafo, com a diferença de que aqui isso é feito separadamente para cada classe de cor composta. Já podemos antever, com isso, que também nos coloridos por classe seja útil dar alguma ordenação aos pontos no início do procedimento.

O seguinte teorema mostra que o valor estimado para o número cromático de um grafo qualquer pelo método seqüencial é o mesmo valor estimado pelo método por classes.

TEOREMA VII-1.

Dada uma seqüência qualquer $v_1, v_2, \ldots, v_n$ dos pontos de G, um colorido seqüencial de G (sem troca) e um colorido de G por classes são equivalentes.

DEMONSTRAÇÃO.

(i) $v_1 \in C_1$ em ambos os coloridos;

(ii) suponhamos que, na execução do colorido seqüencial, $<\{v_1, v_2, \ldots, v_{i-1}\}>$ recebeu um j-colorido. Uma nova classe de cor, C_{j+1}, será iniciada com v_i, se e somente se houver pelo menos um ponto v_p, para $p < i$, que seja adjacente a v_i em cada uma das j classes já iniciadas. Caso contrário, v_i será incluído na primeira classe C_r que não contenha pontos adjacentes a v_i. Da mesma forma, na execução de um colorido por classes, o ponto v_i será incluído numa classe C_j, após terem sido completadas as classes $C_1, C_2, \ldots, C_{j-1}$, se e somente se j for o menor inteiro positivo tal que os pontos adjacentes a v_i em $\{v_1, v_2, \ldots, v_{i-1}\}$ já foram alocados em $C_1, C_2, \ldots, C_{j-1}$.

O primeiro algoritmo que veremos, quadro (VII-6), é devido a Welsh e Powell[46], sendo um dos trabalhos pioneiros no assunto. O outro é de Williams[48], quadro (VII-7), que esteve por mais de dez anos envolvido com construção de tabelas de horário para escolas e universidades.

Quadro VII-6. Algoritmo aproximativo C1.

```
begin                                                          L1
    ordene  v1,v2,...,vn  segundo uma ordenação                L2
    "largest-first";                                           L3
    p:=0;  comment contador de pontos já coloridos;            L4
    for  k:=1  step  1  do                                     L5
        for  (vi ε lista de pontos não coloridos) do           L6
            if  (Ck ∧ Ai) = φ  then                            L7
            begin                                              L8
                Ck:=Ck U{vi} ;                                 L9
                p:= p+1;                                       L10
                if  p=n  then  stop;                           L11
            end                                                L12
end                                                            L13
```

COMPLEXIDADE DE C1.

Já vimos, no algoritmo S1, que a operação da linha L7 pode ser efetuada em tempo O(n+m), contadas todas as iterações.

Portanto, a complexidade de C1 é O(n+m).

DESEMPENHO DE C1.

C1 executa o mesmo colorido que o algoritmo S2 visto na seção anterior, tendo ambos a mesma complexidade. O limite superior que, na análise de S2, apresentamos para $\chi(G)$ ($k \leq \max_{1 \leq i \leq n} \min (i, d_i)$), foi estabelecido inicialmente por Welsh e Powell, com base no algoritmo C1.

Apliquemos C1 ao grafo da figura VII-3.

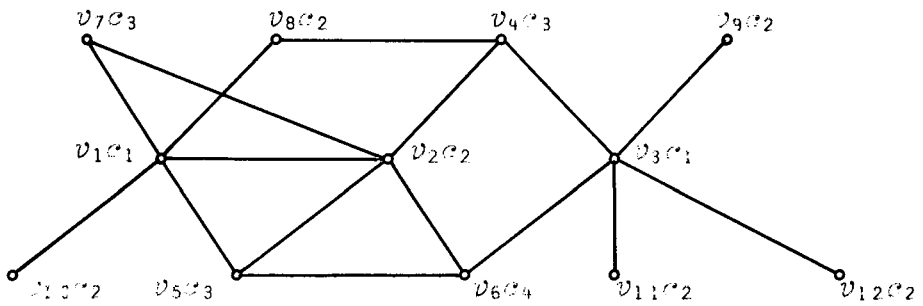

Figura VII-3. Exemplo de aplicação do algoritmo C1.

Observe que os pontos já estão numerados segundo uma ordenação "largest-first". O resultado obtido é um 4-colorido de G, com as seguintes classes de cor: $\{v_1, v_3,\}$, $\{v_2, v_8, v_9, v_{10}, v_{11}, v_{12}\}$, $\{v_4, v_5, v_7,\}$, $\{v_6\}$. Se tivéssemos usado uma ordenação "largest-first", permutando v_1 e v_2 acima, teríamos obtido um 3-colorido de G com as seguintes classes de cor: $\{v'_1, v_3, v_8, v_{10}\}$, $\{v'_2, v_4, v_6, v_9, v_{11}, v_{12}\}$, $\{v_5, v_7\}$, onde $v'_1 = v_2$ e $v'_2 = v_1$ na figura (VII-3).

Qual a propriedade de v_2 que nos levaria a classificá-lo antes de v_1? Observamos que o somatório dos graus dos pontos adjacentes a v_2 é 5+3+3+3+2=16, sendo maior do que o somatório para os pontos adjacentes a v_1, 5+3+2+2+1 = 13. Foi certamente levado por essa observação que Williams propôs o algoritmo C2, apresentado no quadro (VII-7).

DEFINIÇÃO VII-7.

Seja $\underline{d^{(1)}} = d$ o vetor $(d_1, d_2, \ldots, d_n)$ dos graus dos pontos de G. Definimos o vetor $\underline{d^{(p)}}$, para $p > 1$, como sendo o vetor obtido recursivamente pela relação:

$$d_i^{(p)} = \sum_j q_{ij}\, d_j^{(p-1)} \qquad 1 \leqslant i,j \leqslant n,$$

onde q_{ij} é um elemento da matriz de adjacência Q do grafo G.

OBSERVAÇÕES.

(i) $d^{(p)}$ é uma aproximação do autovetor dominante de Q, convergindo para o mesmo quando $p \to \infty$ (Williams[47]).

(ii) Williams[47] verificou que, em geral, após um valor de $p = \lfloor \sqrt[3]{n} \rfloor$, a posição relativa dos elementos no vetor, de acordo com seu valor, permanece constante.

Quadro VII-7. Algoritmo aproximativo C2.

begin	L1
comment $p = \lfloor \sqrt[3]{n} \rfloor$;	L2
calcule $d^{(p)}$;	L3
disponha os pontos de G numa sequência	L4
$v_1, v_2, \ldots, v_n$ *tal que* $d_1^{(p)} \geq d_2(p) \geq \ldots \geq d_n^{(p)}$;	L5
(o restante é idêntico ao algoritmo C1,	L6
linhas L4-L12)	L7
end	L8

COMPLEXIDADE DE C2.

O vetor $d^{(p)}$ pode ser calculado em um tempo - $O(p(n+m))$. Essa será a complexidade do algoritmo, a não ser se considerarmos p constante (= 10, por exemplo), o que reduz a complexidade de C2 a $O(n+m)$.

DESEMPENHO DE C2.

Segundo Williams, C2 produz resultados muito satisfatórios.

SOBRE MELHORAR OU NÃO O ALGORITMO C2.

"Se funciona é melhor não mexer". Pelo título de seu artigo, Williams[48] já antecipa a sua conclusão de que realmente não vale a pena gastar muito tempo tentando melhorar procedimentos heurísticos. Ele afirma isso, não sem antes ter executado uma série extensiva de testes com os algoritmos C1 e C2 apresentados e mais três outros algoritmos com critérios mais rebuscados de ordenação de pontos.

Conforme o trabalho de Johnson, já citado anteriormente, por melhor que seja um procedimento heurístico de coloração ele poderá sempre produzir resultados arbitrariamente ruins. É a mesma conclusão a que chegou Williams, descrevendo os resultados dos seus testes: "Em nenhum exemplo de tamanho ou densidade (número de pontos e de linhas, respectivamente) foi uma das heurísticas de coloração consistentemente melhor do que outra e em

todos os casos foram encontrados exemplos em que a heurística produziu o que vêm a ser resultados arbitrariamente ruins".

VII-3. Coloridos aos Pares.

O método de coloração que analisaremos agora foi desenvolvido por Wood[49]. Ele observou que, em um colorido por classes, utilizando o algoritmo Cl, o número de cores pode ser desnecessariamente aumentado, dependendo da forma como os pontos estejam ordenados, já verificamos isso para o grafo da figura (VII-1). Wood ilustrou o fato com um exemplo muito simples, que reproduzimos na figura (VII-4).

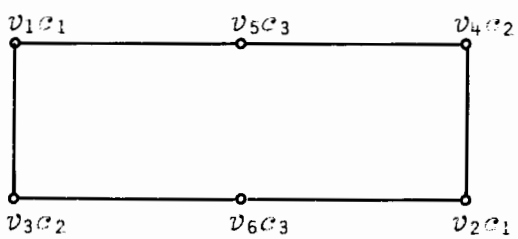

Figura VII-4. Uma atuação ruim do algoritmo Cl.

Como poderia ser evitado que os pontos v_1 e v_2 fossem alocados na mesma classe e como seria possível forçar que v_1 e v_4 o fossem, independentemente da ordenação inicial desses pontos?

Wood resolve esse problema, lembrando uma técnica usada em taxonomia, em que se forma uma matriz de similaridades para indicar quais pares de objetos devem ser postos no mesmo grupo, devido a apresentarem a maior similaridade.

Analogamente, uma matriz de similaridades S, de dimensão nxn, é construída para determinar quais pares de pontos deverão receber a mesma cor. A similaridade entre os pontos v_i e v_j é medida através da matriz de adjacência do grafo, assim:

$$s_{ij} = \begin{cases} 0, \text{ se } q_{ij} = 1 \text{ e} \\ \sum_{1 \le p \le n} b_p, \text{ se } q_{ij} = 0, \text{ onde } b_p = \begin{cases} 1, \text{ se } q_{ip} = q_{jp} = 1, \\ 0, \text{ caso contrário.} \end{cases} \end{cases}$$

Com isso, a similaridade entre dois pontos não adjacentes, v_i e v_j, fica definida como sendo o número de outros pon-

tos v_p que são adjacentes a ambos v_i e v_j.

No algoritmo P1, apresentado no quadro (VII-8), o nível de similaridade l é inicializado com o maior valor existente em S, sendo considerados todos os pares que apresentam tal nível. Depois, o nível é decrescido de 1 e o procedimento é repetido e assim sucessivamente, até que todos os pontos "difíceis" tenham sido coloridos.

Pontos que têm grau menor ou igual ao número de classes de cor já iniciadas, são deixados sem colorir, considerando se que poderão ser facilmente coloridos posteriormente. Wood sugere, ainda, que, para grafos grandes, o nível de similaridade l seja abaixado apenas até 5, por exemplo, pois pares de pontos com similaridade menor que 5 poderão também ser coloridos posteriormente, sem dificuldades.

Nota-se que um outro tipo de ponto, que pode ser deixado sem colorir pelo algoritmo P1, é aquele para o qual não houve um par adequado, isto é, um par que não fosse colorido antes ou que entrasse para uma classe de cor conveniente a ambos.

Quadro VII-8. Algoritmo aproximativo P1.

```
begin                                                              L1
    construa a matriz S; z:= maior valor de S;                     L2
    k:=0;                                                          L3
    for l:=z step (-1) until 0 do                                  L4
    begin                                                          L5
        for i,j:=1 to n do                                         L6
            if s_ij = 1 then                                       L7
                if (v_i e v_j não estão coloridos) then            L8
                begin                                              L9
                    if (d_i > or d_j > k) then                     L10
                    begin                                          L11
                        r:=min {p | C_p Λ(A_i U A_j)=φ} ;          L12
                        C_r:= C_r U { v_i,v_j} ;                   L13
                        k:= max { k,r } ;                          L14
                    end                                            L15
                end else                                           L16
                if (apenas 1,v_i ou v_j, está colorido) then       L17
                begin comment seja v_a colorido e                  L18
                    v_b não colorido, para a,b ε{i,j};             L19
                    r:= c(v_a);                                    L20
                    if (d_b > k and C_r Λ A_b = φ )                L21
                    then C_r := C_r U { v_b}                       L22
                end                                                L23
    end                                                            L24
end                                                                L25
```

COMPLEXIDADE DE P1.

A matriz de similaridades S pode ser encontrada em tempo O(nm).

O teste da linha L7 será executado, no máximo, mz vezes, se acessarmos apenas os pares de pontos adjacentes.

O cálculo de r, linha L12, gasta um tempo O(n) por iteração, visto que os C_p's são disjuntos e $|UC_p| = O(n)$.

O teste da linha L21 leva, no total, O(n).

Considerando que z=O(n), concluímos que a complexidade de P1 é $O(mn^2)$.

DESEMPENHO DE P1.

Em testes comparativos de P1 com C1, utilizando dados randômicos, Wood tirou as seguintes conclusões:

(i) em 25% dos casos, P1 deu resultados melhores do que C1;

(ii) em grafos de até 20 pontos é indiferente usar-se um ou outro método. Quando o número de pontos é da ordem de 100, P1 é consideravelmente melhor do que C1, para os casos em que a matriz de adjacência é menos esparsa.

Analisando o algoritmo P1, Mitchem[35] observou que alguns pontos - por exemplo, um ponto de grau n-1 - podem ainda estar sem cor, após a execução do algoritmo, e exigirem a utilização de novas cores, para se completar o colorido de G.

Mitchem lembrou, também, que existem grafos com número cromático arbitrário, os quais não contêm ciclos de comprimento 4. Nesses grafos, cada par de pontos tem similaridade 0 ou 1,de forma que a sugestão de Wood, de se abaixar l apenas até o valor 5, corresponderia a deixar sem cor todos os pontos de tais grafos.

VII-4. Coloridos em "Batch".

Veremos, agora, algoritmos que colorem um grafo planar qualquer com, no máximo, sete cores, em um tempo O(n), ou com, no máximo, cinco cores, em um tempo O(n$l$ogn) (Lipton e Miller[32]). Eles foram publicados seis anos depois do algoritmo S5 que, con

forme vimos, colore um grafo planar qualquer com, no máximo, cinco cores, em um tempo O(n(m+n)). Por sinal, um dos autores daquele algoritmo, Matula, orientou Lipton e Miller na produção do seu artigo.

A técnica empregada nesses algoritmos baseia-se na idéia de processar de uma só vez uma coleção de pontos. Daí ter sido denominada por seus autores de "batching method".

O algoritmo B1, quadro (VII-9), reduz o grafo inicial sucessivas vezes, retirando de cada vez, um conjunto independente de pontos, que pode ser encontrado rapidamente. Esses pontos têm grau menor ou igual a seis no subgrafo considerado, o que facilita a atribuição de cores aos mesmos quando eles estão sendo acrescentados de volta ao grafo.

Quadro VII-9. Algoritmo aproximativo B1.

```
begin                                                              L1
    procedure PINTE (H,p)                                          L2
    begin  comment  H é um grafo com conjunto de                   L3
           pontos W, sendo p:=|W|;                                 L4
           if  p < 7 then                                          L5
           (atribua a cada ponto uma cor diferente) else           L6
           begin                                                   L7
                encontre em H um conjunto com r > p/28             L8
                pontos independentes com grau ≤ 6;                 L9
                comment seja v_i1, v_i2, ..., v_ir, tal conjunto   L10
                e seja H'=<W'> onde W'=W-{v_i1, v_i1, ..., v_ir};  L11
                PINTE (H', n-r);                                   L12
                for  j:=1  until r do                              L13
                     atribua a cada ponto v_ij  uma cor            L14
                     não usada em A_ij                             L15
           end                                                     L16
    end  PINTE;                                                    L17
    PINTE  (G,n); comment  G é um grafo planar                     L18
end                                                                L19
```

OBSERVAÇÃO.

É possível fazer a atribuição indicada em L14-L15, pelo seguinte motivo. Todos os conjuntos A_{i_j}, $1 \leq j \leq r$, estão contidos em W', uma vez que $v_{i_1}, v_{i_2}, \ldots, v_{i_r}$ é um conjunto independente e $W' = W' - \{ v_{i_1}, v_{i_2}, \ldots, v_{i_r} \}$. Logo, todos os pontos em A_{i_j}, $1 \leq j \leq r$, são coloridos através da chamada PINTE (H',n-r). Como A_{i_j} tem seis ou menos elementos, existe, no mínimo, uma dentre as sete cores que não foram utilizadas em A_{i_j}, que pode, portanto, ser atribuída a v_{i_j}.

Lipton e Miller[32] demonstraram, utilizando a fórmula de Euler, que em um grafo com n pontos é possível encontrar em um tempo O(n), um conjunto de r pontos independentes, todos de grau menor ou igual a 6, para $r < p/28$.

Foi demonstrado, ainda, que o algoritmo B1 tem complexidade O(n). A demonstração foi construída por indução no número de pontos de G.

DESEMPENHO DE B1.

Enquanto que, para o caso geral de coloração de grafos quaisquer, não se conseguiu qualquer algoritmo aproximativo que permitisse uma boa estimativa de erro, temos, aqui, para o caso particular de grafos planares, um algoritmo cuja atuação nos dará, no pior dos casos, uma estimativa para o número cromático distando de 6 do valor real.

APLICANDO A TROCA BICROMÁTICA.

Lipton e Miller[32] demonstraram que, sempre que for necessário utilizar a sexta cor pelo algoritmo B1, é possível efetuar uma troca bicromática que efetivamente libere uma das cores utilizadas, dispensando a introdução da sexta cor. O momento em que tal troca é feita, está mostrado no algoritmo B2, quadro (VII-10).

Quadro VII-10. Algoritmo aproximativo B2.

```
begin                                                              L1
    procedure PINTE (H,p)                                          L2
    begin comment H,W e p são como em B1;                          L3
        if p ≤ 5 then                                              L4
        (atribua a cada ponto uma cor diferente) else              L5
        begin                                                      L6
            construa { v_{i_1}, v_{i_2}, ..., v_{i_r} } e H' como em B1   L7
            PINTE (H',n-r);                                        L8
            for j:=1 until r do                                    L9
                if (os pontos de A_{i_j} foram coloridos           L10
                    com menos de 5 cores)                          L11
                then (atribua a v_{i_j} uma das cores              L12
                    que restam)                                    L13
                else (faça uma troca bicromática                   L14
                    em H',de forma a liberar uma                   L15
                    das cores que coloriam os pontos de            L16
                    A_{i_j}, e atribua a v_{i_j} a cor liberada    L17
        end                                                        L18
    end PINTE;                                                     L19
    PINTE (G,n)                                                    L20
end                                                                L21
```

COMPLEXIDADE DE B2.

O algoritmo B2 tem complexidade $O(n\ell ogn)$.

Isso foi demonstrado no artigo de Lipton e Miller[32]. Devido à complexidade da demonstração, não a reproduziremos aqui.

DESEMPENHO DE B2.

Ainda não foi criado nenhum algoritmo eficiente para colorir um grafo planar qualquer com quatro ou menos cores. Sabemos, no entanto, que isso é possível, após a demonstração obtida por Haken e Appel. O algoritmo B2 nos deixa a um passo do desejado, garantindo um máximo de cinco cores.

VII-5. Casos Arbitrariamente Ruins.

Para qualquer par de inteiros $p > k > 3$, é possível construir um grafo G tal que $\chi(G)=k$, de forma que os algoritmos de coloração genérica vistos produzem um colorido com, pelo menos, p cores.

Esse resultado, devido a Mitchem[35], justifica o fato de não termos apresentado exemplos de pior desempenho de cada algoritmo analisado. Tais exemplos simplesmente não existem, uma vez que é possível formular um caso tão ruim quanto se queira.

CONSTRUINDO GRAFOS RUINS.

O grafo G a que nos referimos é obtido da seguinte forma.

$V(G) = W \cup Z$, onde $W=W_1 \cup W_2 \ldots \cup W_k$ e $Z=Z_1 \cup Z_2 \ldots \cup Z_k$. Por sua vez, $W_i=\{w_{i,1};w_{i,2};\ldots w_{i,p}\}$ e $Z_i=\{z_{i,1};z_{i,2};\ldots;z_{i,jp^2}\}$. Esses conjuntos são mutuamente disjuntos, de forma que G possui um total de $n=kp + k^2 p^2$ pontos.

Os pontos $w_{i,j}$ e $w_{r,s}$ são adjacentes para todo $i\neq r$ e $j\neq s$. Os pontos $z_{i,j}$ e $z_{r,s}$ são adjacentes para todo $i\neq r$. Além disso, para cada $j > 1$, $w_{i,j}$ é adjacente a cada um dos pontos $z_{2,(j-2)kp+1};\ldots;z_{2,(j-2)kp + (j-1)k}$. Para cada $j>1$ e cada $i>1$, $w_{i,j}$ é adjacente a cada um dos pontos $z_{1,(j-2)kp+1};\ldots;z_{1,(j-2)kp+(j-1)k}$.

Como para i = 1,2...,k os pontos com primeiro subscrito i não são adjacentes, podemos colori-los com a cor i. Portanto $\chi(G)=k$.

COLORIDO SEQÜENCIAL "LARGEST-FIRST" (S2).

O grau de cada ponto de Z é, pelo menos, $kp^2(k-1)$, e o grau de cada ponto de W, $w_{i,j}$, é dado por $(k-1)(p-1)+(j-1)k$, que é menor do que $kp^2((k-1)$. Logo, numa ordenação "largest-first" os pontos de Z ficarão antes dos pontos de W.

O subgrafo <Z> receberá um k-colorido, sendo que os pontos de $z_{1,j}$ serão coloridos com a cor 1 e os pontos $z_{2,j}$ com a cor 2. Então, os pontos $w_{1,p}$ receberão a cor 1 e, para i=2,...,k, os pontos $w_{i,p}$ receberão a cor 2. Também, $w_{1,p-1}$ receberá a cor 1 e cada $w_{i,p-1}$ a cor 3.

Analogamente, para j=2,...,p-1; os pontos $w_{1,p-j}$ serão coloridos com a cor 1 e, para i=2,...,k; os pontos $w_{i,p-j}$ serão coloridos com a cor j+2.

Isso resultará, portanto, em um (p+1)-colorido de G.

COLORIDO SEQÜENCIAL "SMALLEST-LAST" (S3).

Ordenando os pontos de G de acordo com uma ordenação "smallest-last", os pontos $w_{i,1}$ serão os últimos k elementos, os pontos de $w_{i,2}$ serão os k penúltimos e assim sucessivamente para os pontos $w_{i,3}, w_{i,4}, \ldots, w_{i,p}$. Os pontos de Z serão os k^2p^2 primeiros elementos da ordenação e k cores serão gastas para colori-los.

Suponhamos que os pontos $z_{i,j}$ receberam a cor s e os pontos $z_{2,j}$ receberam a cor t. Temos, então, dois casos a considerar:

(i) s < t. Os pontos $w_{i,p}$ serão coloridos com a cor 1, $w_{i,p-1}$ com a cor 2, e assim sucessivamente, até os pontos $w_{i,p-s+2}$, que serão coloridos com a cor s-1. O ponto $w_{1,p-s+1}$ será colorido com a cor s e os demais $w_{i,p-s+1}$ com a cor s+1. Em seguida, para cada j<p - s+1, os pontos $w_{1,j}$ serão coloridos com a cor s e, também, para cada i > 1 os pontos $w_{i,j}$ receberão a cor p-j+2. Obtemos, portanto, um (p+1)-colorido de G.

(ii) t < s. Os pontos $w_{i,p}$ receberão a cor 1, $w_{i,p-1}$ a cor 2 e assim sucessivamente até $w_{i,p-t+2}$, que receberão a

cor t-1. O ponto $w_{1,p-t+1}$ será colorido com a cor t+1 e os demais $w_{i,p-t+1}$ serão coloridos com a cor t+1. Os pontos $w_{i,j}$ serão coloridos com a cor p-j+2 para i > 1 e $(p-s+2)<j<(p-t+1)$. Para i > 1 e $j \leq p-s+2$, os pontos $w_{i,j}$ receberão a cor p-j+3. Isso nos dá um (p+2)-colorido de G.

COLORIDO SEQÜENCIAL COM TROCA (S5).

Os pontos de Z são coloridos com k cores. Ao colorir os pontos de W, sempre que uma nova cor é necessária, o conjunto T (das cores atribuídas a exatamente um ponto adjacente ao que está sendo colorido) contém, no máximo, um elemento, o que impossibilita a troca bicromática. Portanto, esse algoritmo atribuirá a G um colorido equivalente ao atribuído pelo algoritmo sem troca, utilizando mais de p cores.

COLORIDO AOS PARES (P1).

Seja S(z,w) a similaridade de dois pontos distintos z e w em G. Então para i=r, $S(z_{i,j}; z_{r,s}) \geq (k-1)\,kp^2$;

para $i \neq r$, $S(z_{i,j}, z_{r,s}) = 0$;

para $i \neq 1$, $S(w_{1,j}, w_{i,j}) = (k-2)\,(p-1)$;

para $i > r > 1$, $S(w_{i,j}; w_{r,j}) = (k-2)\,(p-1) + (j-1)k$;

para $j \neq s$, $S(w_{i,j}; w_{i,s}) = (k-1)\,(p-2)$;

para $i \neq r$, $j \neq s$, $S(w_{i,j}; w_{r,s}) = 0$ e

$S(w_{i,j}; z_{r,s}) \leq (j-1)\,k + k - 1$.

O algoritmo P1 tratará os pontos de Z em primeiro lugar, gastando com os mesmos k cores. Como

$(k-2)\,(p-1) + (j-1)\,k > (k-1)\,(p-2)$ se e somente se $j \geq p/k$, segue-se que, para i > 1, os pontos $w_{i,p}$ serão coloridos com a cor 1; $w_{i,p-1}$, com a cor 2; e assim sucessivamente, até os pontos $w_{i,p-t+2}$, que receberão a cor t-1. Os pontos $w_{i,p-t+1}$ receberão a cor t+1; $w_{i,p-t}$ a cor t+2;...; $w_{i,r}$ a cor p-r+2 para $r = \{p/k\}$. Os pontos restantes são coloridos e o algoritmo usa, pelo menos, $p-r+2 > p - (p/k-1)+2 > p(k-1)/k$ cores. Portanto, para $p > k > 3$, o grafo G tem número cromático k e o algoritmo P1 fornece um colorido com, pelo menos $q(k-1)/k \geq p$ cores, onde $q = kp/(k-1)$.

E DAÍ?

Pelo envolvimento do exemplo desenvolvido por Mitchem

e que acabamos de reproduzir, sentimos que na prática coisas tão ruins não têm muita chance de acontecer.

Por outro lado, embora tenhamos visto a atuação de apenas três dos algoritmos apresentados, temos certeza que exemplos ruins podem ser construídos para qualquer deles, à exceção dos algoritmos B1 e B2, que são restritos a grafos planares.

A importância desse resultado é esclarecer que situações ruins podem surgir na aplicação dos algoritmos, e mostrar que não é possível identificar o pior caso para os mesmos.

As heurísticas de coloração existentes constituem uma alternativa viável para a obtenção do número cromático e de coloridos para grafos quaisquer e têm sido adotadas com bons resultados no problema de construção de tabelas de horário de exames.

VIII. OBTENÇÃO DE $\chi(G)$.

Descobriram-se várias maneiras diferentes de se obter o número cromático exato de um grafo G qualquer. O objetivo,em todas elas, deriva-se da própria definição de número cromático e de classes de cor. Consiste em se encontrar o menor número k de conjuntos independentes de G, C_1, C_2,...,C_k, de forma que sua união contenha todos os pontos de V, $U_{i=1,k}C_i = V$. Nesse caso, dizemos que $C_1,C_2,...,C_k$ 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, $\chi(G)$.

Agruparemos os algoritmos de acordo com o enfoque principal, por eles adotados, para tratamento de G. Assim, na seção (VIII-1), veremos o problema de coloração, formulado do ponto de vista algébrico. Daremos o nome de coloridos algébricos aos coloridos obtidos por algoritmos desse tipo. Serão apresentados um algoritmo sugerido por Berge[2], em 1958, um algoritmo de Hammer e Rudeanu[23], publicado em 1968, e um outro sugerido por Christofides[9], em 1975.

Os coloridos independentes de um grafo, a serem apresentados na seção (VIII-2), são todos eles fundamentados no procedimento desenvolvido em 1971, por Christofides[8]. Seguem-no o trabalho de Furtado et al[39], publicado em 1973, o de Wang[45], de 1974, o de Lawler[30], de 1976 e, finalmente, o algoritmo de Szwarcfiter[42], desenvolvido em 1978 e ainda não publicado.

Veremos, na seção (VIII-3), algoritmos que utilizam "backtracking" para chegarem a um colorido ótimo, a partir da enumeração de várias soluções aproximadas. Denominamos o grupo de coloridos por enumeração. São trabalhos devidos a Brown[7],de 1972, e a Nijenhuis e Wilf[37], de 1975.

Serão apresentados, então, na seção (VIII-4), dois algoritmos de 1973, que abordam o problema, utilizando um processo de redução de grafos, proposto por Zikov em 1949. São os coloridos por redução, devidos a Corneil e Graham[12].

Por fim, veremos, na seção (VIII-5), um algoritmo publicado em 1972 por Gavril[22],que atribui um colorido seqüencial ótimo a uma classe específica de grafos, os grafos cordais.

VIII-1. Coloridos Algébricos.

Passaremos rapidamente pelos algoritmos desse grupo. Eles não são adequados para resolução do problema de número cromático, por computadores.

MÉTODO ANALÍTICO.

Dados um grafo G e um número inteiro k, Berge[2] sugere que se verifique analiticamente se G é k-colorável, pelo seguinte procedimento. A cada k-colorido fazemos corresponder os números E(i,q), para $1 \leqslant i \leqslant n$ e $1 \leqslant q \leqslant k$, da seguinte forma:

$$E(i,q) = \begin{cases} 1, & \text{se o ponto } v_i \text{ tiver a cor q,} \\ 0, & \text{caso contrário.} \end{cases}$$

Definimos a matriz R de dimensão n x m, por

$$r_{ij} = \begin{cases} 1, & \text{se a linha } u_i \text{ for incidente com o ponto } v_j, \\ 0, & \text{caso contrário.} \end{cases}$$

O problema de se determinar a k-colorabilidade de G pode então ser expresso pelo algoritmo A1, descrito no quadro (VIII-1). A1 consiste de um sistema de desigualdades lineares, que deve ser resolvido, aplicando-se os métodos usuais de programação linear.

Quadro VIII-1. Algoritmo exato A1.

begin	L1
encontre os números inteiros E(i,q), tais que:	L2
(i) E(i,q)>0, para 1<i<n e 1<q<k;	L3
(ii) $\sum_{j=1}^{q} E(i,j)=1$, *para 1<i<n;*	L4
(iii) $\sum_{i=1}^{n} r_{ij} E(i,q) < 1$, *para 1<j<m e 1<q<k*	L5
end	L6

CONJUNTOS INDEPENDENTES MAXIMAIS.

Já sabemos que um colorido ótimo para G pode ser obtido pelo seguinte procedimento básico:

(i) encontram-se todos os conjuntos independentes de G, que denotaremos por CI;

(ii) identifica-se um conjunto mínimo de CI's que cubra G. Cada CI assim obtido é uma classe de cor e o número de CI's é o número cromático de G.

Dado um grafo G, qualquer CI de G está contido em um conjunto independente maximal de G, que denotaremos por CIM. Por isso, se determinarmos todos os CIM's de G, teremos indiretamente determinado todos os seus CI's.

O teorema seguinte, devido a Weissman e apresentado na obra de Hammer e Rudeanu[23], garante que é suficiente considerarmos os conjuntos independentes maximais, na obtenção do número cromático de G.

TEOREMA VIII-1.

O número cromático χ de um grafo G qualquer é igual ao menor número de conjuntos independentes maximais que cobrem G. Se $S_1, S_2, \ldots, S_k$ são k conjuntos independentes maximais cobrindo G, onde $K=\chi(G)$, então as classes de cor $C_1, C_2, \ldots, C_k$, de um colorido ótimo de G, podem ser obtidas pelas seguintes relações:

$$C_1 = S_1,$$

$$C_2 = S_2 - S_1,$$

$$\ldots$$

$$C_i = S_i - \bigcup_{j=1}^{i-1} S_j$$

$$\ldots$$

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

DEMONSTRAÇÃO.

Para uma demonstração do teorema acima, consultem-se as páginas 228-229 do Hammer e Rudeanu[23].

Na prática, as classes de cor acima podem ser obtidas por recoloração de pontos, fazendo-se $C_k=S_k$, $C_{k-1}=S_{k-1}, \ldots, C_1=S_1$.

DETERMINAÇÃO DOS CIM's DE G.

O problema de se determinarem os CIM's de um grafo qualquer deve incluir necessariamente a solução de um problema NP-completo. Todos os algoritmos conhecidos para fazê-lo têm complexidade exponencial. Um algoritmo muito eficiente para ob-

tenção dos CIM's foi desenvolvido por Tsukiyama, Ide, Arujoshi e Ozaki, tendo complexidade O(mnp) (Tsukiyama et al[44]), onde $p \leq 3^{n/3}$ é o número de CIM's do grafo considerado (o valor $3^{n/3}$ foi determinado por Moon e Moser[36]).

Ao apresentarmos os próximos algoritmos de coloração, teremos algumas etapas que solicitam a determinação dos CIM's de algum grafo. Assumiremos que o algoritmo dos quatro japoneses, mencionado acima, será utilizado na implementação de tais etapas.

COLORINDO POR MEIO DE EQUAÇÕES BOOLEANAS.

O algoritmo de coloração que consideraremos agora foi apresentado por Hammer e Rudeanu[23], em 1968. A partir de um grafo G qualquer e dos seus p conjuntos independentes maximais, constrói-se uma equação booleana E com p variáveis, de maneira que uma solução de E, com o menor número possível de 1's, corresponde à seleção de um conjunto mínimo de CIM's cobrindo G.

DEFINIÇÃO VIII-1.

Sejam $S_1, S_2, \ldots, S_p$ todos os CIM's de G. A matriz de cobertura de G é uma matriz booleana n x p, $C=(c_{ij})$, assim definida:

$$c_{ij} = \begin{cases} 1, \text{ se } i \in S_j, \\ 0, \text{ se } i \notin S_j. \end{cases}$$

Observemos que a j-ésima coluna de C é igual ao vetor característico de S_j em relação a V.

DEFINIÇÃO VII-2.

Seja $S = \{S_1, S_2, \ldots, S_p\}$ o conjunto de todos os CIM's de G. A cada subconjunto N de S associamos o vetor $Z=(z_1, z_2, \ldots, z_p)$, assim definido:

$$z_j = \begin{cases} 1, \text{ se } S_j \in N, \\ 0, \text{ se } S_j \notin N. \end{cases}$$

Z é o vetor característico de N em relação a S.

O algoritmo A2, mostrado no quadro (VIII-2), é a aplicação imediata do seguinte teorema:

TEOREMA VIII-2.

Uma condição necessária e suficiente para que um conjun

to N de CIM's cubra G é que o vetor $(z_1,z_2,\ldots,z_p)$ associado a N satisfaça a seguinte equação booleana:

$$\prod_{l=1}^{n} \bigcup_{j=1}^{p} c_{ij}\, z_j = 1.$$

DEMONSTRAÇÃO.

O ponto i de G pertence a, pelo menos, um dos cojuntos independentes maximais de N, se e somente se

$$\bigcup_{j=1}^{p} c_{ij} z_j = 1.$$

Como todos os pontos de G têm que ser cobertos por N, fazemos a conjunção dessa relação para os valores de i correspondentes.

Quadro VIII-2. Algoritmo exato A2.

begin	*L1*
encontre $\{ S_1, S_2, \ldots, S_p \}$;	*L2*
construa a matriz de cobertura C de G;	*L3*
resolva a equação booleana $\prod_{l=1}^{n} \bigcup_{j=1}^{p} c_{ij} z_j = 1$ *com*	*L4*
o menor número possível de 1's	*L5*
end	*L6*

OBSERVAÇÃO.

Sejam $z_{i_1}, z_{i_2}, \ldots, z_{i_k}$ os 1's da solução $(z_1,z_2,\ldots,z_p)$, obtida pela execução das linhas L4-L5 de A2. Então, o número cromático de G é k e um colorido exato de G, a partir dos CIM's $S_{i_1}, S_{i_2}, \ldots, S_{i_k}$, pode ser conseguido construindo-se as classes de cor $C_{i_1}, C_{i_2}, \ldots, C_{i_k}$, conforme a indicação do teorema (VIII-1).

CONSIDERAÇÕES SOBRE A2.

Métodos para se resolver a equação booleana do algoritmo são apresentados na obra já mencionada. Não entraremos em detalhes acerca desses métodos, pois estão fora do escopo de nosso trabalho. Queremos somente frisar que os mesmos foram desen-

volvidos visando à computação manual, sendo altamente ineficientes.

Outros algoritmos para cobertura por conjunto já tinham sido desenvolvidos na década de 50, para aplicações em teoria de chaveamento. Um representante desses algoritmos é apresentado por Even[16], que, ao final, sugere a adaptação do mesmo para utilização em coloração de grafos.

OUTRA FORMULAÇÃO ALGÉBRICA.

Uma outra formulação do problema, em termos de programação inteira, foi proposta por Christofides[9], consistindo em uma alternativa para as equações e restrições de Berge, que Christofides mostra ser mais eficiente.

Seja E a matriz que relaciona os n pontos de G com as k cores dadas. Seja L um inteiro positivo maior que n. Associamos a cada cor j a penalidade p_j, tal que $p_{j+1} > hp_j$, onde h um limite superior para o maior número de pontos independentes de G. Uma solução E(i,j) do algoritmo A3, quadro (VIII-3), corresponde a um colorido ótimo de G.

Quadro VIII-3. Algoritmo exato A3.

begin	L1
minimize $z = \sum_{j=1}^{q} \sum_{i=1}^{n} p_j E(i,j)$, *atendendo às*	L2
restrições:	L3
(i) $\sum_{j=1}^{q} E(i,j) = 1$, *para todo* $1<i<n$;	L4
(ii) $L\,(1-E(i,j)) - \sum_{k=1}^{n} a_{ik} E(k,j) > 0$, *para* $1<i<n$ *e* $1<j<q$	L5
end	L6

OBSERVAÇÕES.

Christofides mostra que, do ponto de vista computacional, é mais vantajoso utilizar-se as restrições do algoritmo A3 do que as do algoritmo A1. Maiores explicações podem ser obtidas nas páginas 66-68 do Christofides[9].

VIII-2. Coloridos Independentes.

Um resultado decorrente do teorema (VIII-1) é que, se G for um grafo k-cromático, então existirá, pelo menos, um colorido de G em que uma das classes de cor, designada por C_1 no caso do teorema, é um CIM de G.

Observemos que o mesmo pode ser dito a respeito do grafo $<V-C_1>$, ou seja, $<V-C_1>$ é um grafo (k-1)-cromático e existe um (k-1)-colorido de $<V-C_1>$ em que uma das classes de cor, D_1, é um CIM de $<V-C_1>$.

O mesmo se aplica a $<V-(C_1 \cup D_1)>$, e, assim, sucessivamente, até obter-se um grafo nulo.

Sabemos, pelo teorema, que existem os coloridos a que nos referimos. Entretanto, como não temos qualquer indicação de qual seja o CIM que atua como uma classe de cor de um colorido ótimo, temos que considerar, a cada etapa, todos os CIM's possíveis.

DEFINIÇÃO VIII-1.

Chamamos de colorido independente de G um k-colorido 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 $<V-C_1>$, e, generalizando, a i-ésima classe de cor é um CIM de $<V-C_1-C_2-\ldots-C_{i-1}>$, para $1 \leq i < k$.

Consideremos um colorido ótimo qualquer de um grafo G, com classes de cor $C_1, C_2, \ldots, C_k$, onde $k = \chi(G)$. Podemos obter um colorido de G com classes $D_1, D_2, \ldots, D_k$, no qual D_1 é um CIM de G, da seguinte forma. Tomamos como D_1 o conjunto independente maximal contendo Cl. Para $1 < i \leq k$, tomamos como D_i o conjunto C_i menos os elementos já pertencentes a D_1, isto é, $D_i = C_i - D_1$. Se aplicarmos esse procedimento a G, a $<V-D_1>$, a $<V-(D_1 \cup E_1)>$, e assim sucessivamente, obteremos um colorido independente ótimo de G, a partir do colorido ótimo inicial. Temos, portanto, o seguinte teorema:

TEOREMA VIII-3.

Se $\chi(G) = k$, então existe um k-colorido independente de G.

Christofides foi quem primeiro procurou um colorido ó-

timo de G entre os seus coloridos independentes, devendo-se a ele a formulação original do teorema (VIII-3). Os algoritmos I1 a I5, que veremos nesta seção, trabalham apenas com os coloridos independentes de G.

DEFINIÇÃO VIII-2.

Um galho de uma árvore é uma linha que liga dois nodos da mesma. Dizemos que um galho g pertence ao nível $\ell+1$ da árvore, se g liga um nodo do nível ℓ com um nodo do nível $\ell+1$.

DEFINIÇÃO VIII-3.

A árvore de subgrafos de G é uma árvore T, cujos nodos são identificados com algum subgrafo de G e cujos galhos são identificados com conjuntos de pontos de G, da seguinte forma:

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

(ii) Seja H um nodo de T, localizado no nível t, com conjunto de pontos W, isto é, $H = \langle W \rangle$. Se W for um conjunto vazio, então H é um grafo nulo, sendo uma folha de T. suponhamos que W não seja vazio. Sejam $S_1, S_2, \ldots, S_p$ os CIM's de H, $p \geq 1$.

Então H tem p filhos, no nível t+1, a saber, $\langle W-S_1 \rangle$, $\langle W-S_2 , \ldots, W-S_p \rangle$, e p galhos $S_1, S_2, \ldots, S_p$, ligando H a cada um dos seus filhos, respectivamente.

O algoritmo de Christofides, cuja idéia delineamos no quadro (VIII-4), através do algoritmo I1, resolve o problema genérico de coloração, construindo, sucessivamente, os níveis da árvore de subgrafos de G, até atingir a primeira folha de T. O nível em que se encontra essa folha é o número cromático de G. As classes de cor de um colorido independente ótimo estão determinadas pela seqüência de galhos que liga a raiz de T à folha alcançada.

Quadro VIII-4. Algoritmo exato I1.

```
begin                                                              L1
    coloque G como nodo raiz, no nível 0;                          L2
    t:=0;                                                          L3
    forever do                                                     L4
    begin                                                          L5
        encontre os CIM's dos nodos H=<W>do nível t;               L6
        for (S_i um CIM de H ) do                                  L7
        begin                                                      L8
            coloque <W-S_i> como um nodo do nível t+1;             L9
            ligue <W-S_i> a H pelo galho S_i;                      L10
            if (<W-S_i>é um grafo nulo) then stop;                 L11
            if (<W-S_i> já apareceu no nível t+1)                  L12
            then descontinue o ramo de <W-S_i>                     L13
        end                                                        L14
        t:=t+1;                                                    L15
    end                                                            L16
end                                                                L17
```

OBSERVAÇÕES SOBRE I1.

O algoritmo I1 constrói a árvore de subgrafos de G numa ordem "breadth-first", pois passa para o nível t+1 somente após completar o nível t, como vemos no quadro (VIII-4.)

Ramificações da árvore, que chegam a um nodo contendo um subgrafo já obtido em outro nodo, são abandonadas, linhas L12-L13, pois forneceriam coloridos equivalentes de G.

EXEMPLO DE APLICAÇÃO DE I1.

Vejamos com atua I1, quando aplicado ao grafo da figura VIII-1.

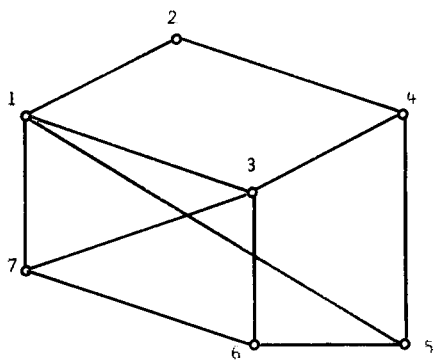

Figura VIII-1. Grafo para aplicação de I1.

Inicialmente, G é colocado no nível 0 da árvore.

Os CIM's de G são {1,4,6}, {2,3,5}, {2,7,5} e {2,6} . São criados os quatro nodos correspondentes, ligados a G pelos galhos que contêm esses CIM's. Formado o nível 1, passa-se à construção do nível 2. A árvore de subgrafos de G é apresentada na figura VIII-2. A parte tracejada não chega a ser construída pelo algoritmo.

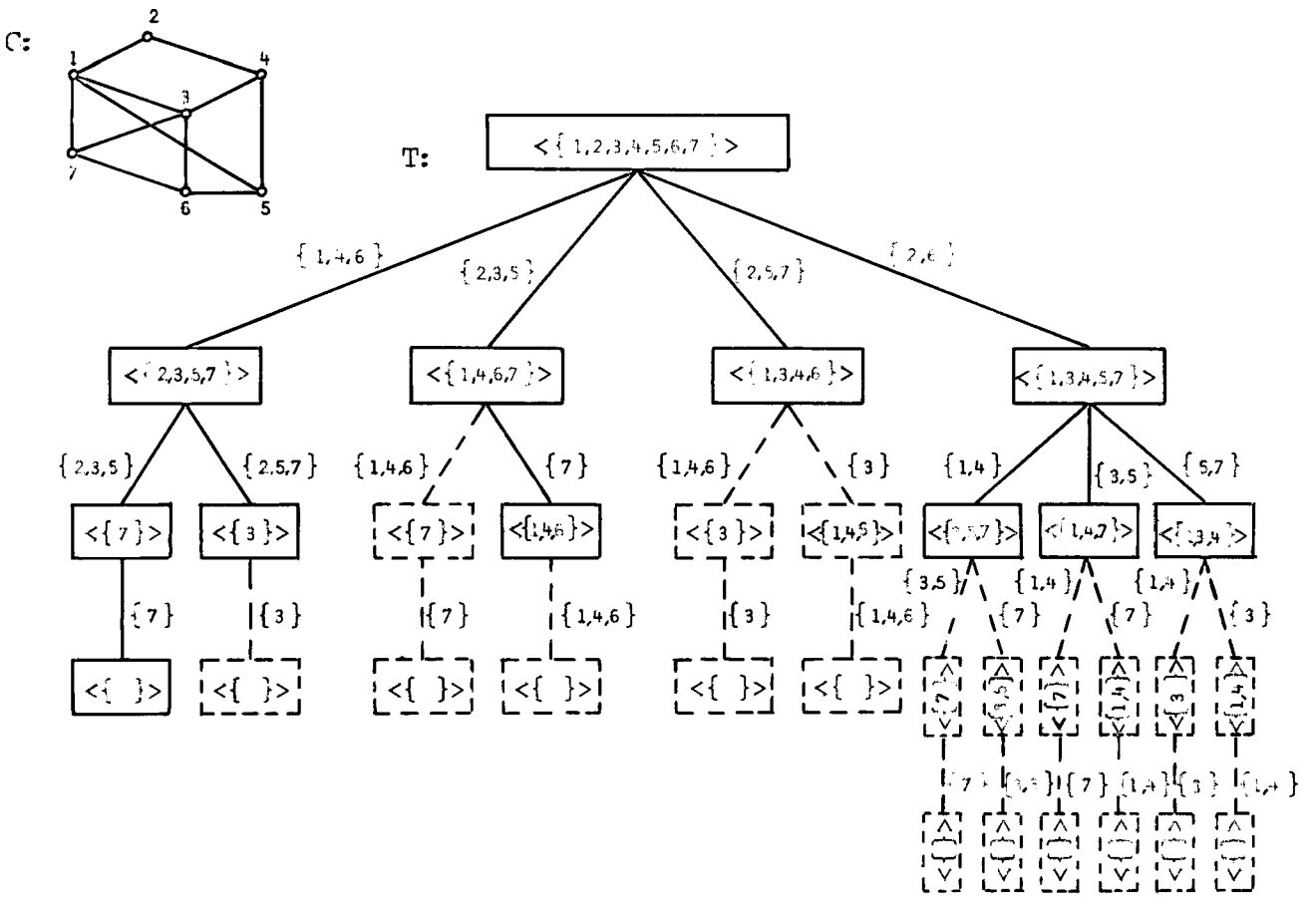

Figura VIII-2. Árvore de subgrafos de G.

O número cromático de G é 3 e as classes de cor, obtidas pelo algoritmo, são $C_1=\{1,4,6\}$, $C_2=\{2,3,5\}$ e $C_3=\{7\}$.

COMPLEXIDADE DE I1.

O número de vezes que o "loop" principal, L4-L16, será executado é igual à altura da árvore de subgrafos de G que, por sua vez, é O(n), no pior dos casos.

Considerando o resultado, devido a Moon e Moser[36], de que o número de CIM's de um grafo qualquer não ultrapassa $3^{n/3}$, um limite superior, para o número de nodos no nível t é $O((3^{n/3})^t)$. Como os CIM's de um grafo com n pontos podem ser identificados em $O(mn(3^{n/3}))$ (Lawler[31]), chegamos ao resultado $O(mn(3^{n/3})^{t+1})$ para o cálculo dos CIM's dos nodos do nível t. Maximizando e somando para as n possíveis iterações, temos:

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

Concluímos que a complexidade de I1 é dada por $O(mn^2(3^{1/3})^{n^2})$. Esse resultado serve para qualquer dos algoritmos que trabalham com a árvore de subgrafos.

Um valor mais refinado foi obtido por Lawler[30], e será estudado juntamente com o algoritmo I4.

"BRANCH AND BOUND".

O algoritmo I2, que estudaremos em seguida, foi desenvolvido por Furtado, Roschke, Santos e Bauer[39], em 1972, consistindo em uma alteração do algoritmo de Christofides, que utiliza a técnica de branch and bound".

A partir de um nodo H da árvore de subgrafos de G, I2 escolhe um vértice v presente no menor número r de CIM's de H, S_1, S_2,...,S_r, numerados por ordem decrescente de cardinalidade. Os únicos nodos de H, construídos por I2, são $<W-S_1>$, $<W-S_2>$, ..., $<W-S_r>$. Veremos, mais na frente, que Wang[45] chegou a essa mesma idéia, que ele chamou de podar a árvore de subgrafos de G.

Como no algoritmo de Christofides, a pesquisa é feita em breadth-first, e, ao ser encontrada a primeira folha, é encerrado o processo, sendo que as classes de cor são identificadas voltando-se da folha para os seus nodos antecessores, até

atingir a raiz.

Outra idéia introduzida por Christofides et al consiste em calcular os CIM's apenas para G. Os CIM's dos demais subgrafos da árvore T seriam obtidos por diferença de conjuntos. Entretanto, levando em consideração o trabalho gasto em eliminar as duplicatas e os conjuntos não maximais que surgiriam por esse método, acreditamos que sairia mais em conta aplicar-se em cada subgrafo o algoritmo de geração dos CIM's, principalmente em se tratando do algoritmo dos quatro japoneses.

No quadro (VIII-5), delineamos o algoritmo I2.

Quadro VIII-5. Algoritmo exato I2.

```
begin                                                          L1
    coloque G como nodo raiz, no nível 0;                      L2
    t:=0;                                                      L3
    forever do                                                 L4
    begin                                                      L5
        for (H=<W>, um nodo do nível t) do                     L6
        begin                                                  L7
            escolha v ε W, presente no menor                   L8
            número r de CIM's de H;                            L9
            comment sejam S1,S2,...,Sr tais CIM's              L10
            por ordem decrescente de cardinalidade;            L11
            for i:=1 until r do                                L12
            begin                                              L13
                coloque <W-Si> como um nodo do ni-
                                vel t+1;                       L14
                ligue <W-Si> a H pelo galho Si;                L15
                if <W-Si> é um grafo nulo then stop;           L16
                if (<W-Si> é repetição) then abandone
                                <W-Si>                         L17
            end                                                L18
        end                                                    L19
        t:=t+1;                                                L20
    end                                                        L21
end                                                            L22
```

COLORINDO GRAFOS GRANDES E POUCO DENSOS.

Mencionaremos, aqui, outra estratégia descrita por Roschke e Furtado[39], que torna viável a coloração exata de grafos grandes - de até 100 pontos - desde que tenham uma baixa densidade de linhas. Ao descrever esse método, suporemos que G possa ser um grafo desconexo.

G é um grafo biconexo, se e somente se, dados três pontos quaisquer, v_i, v_j, v_k, de G, existe um caminho que liga v_i a v_j e que não contém v_k.

Um nodo de corte de G é um ponto que, se for retirado, juntamente com as linhas com ele incidentes, aumenta o número de componentes de G.

A idéia que queremos apresentar consiste em construir um grafo B com os componentes biconexos de G, identificados nos componentes de G que contém um dado ponto v. Dois pontos desse grafo são ligados por uma linha se os componentes biconexos correspondentes, B_i e B_j, compartilham um ponto de corte de B.

O grafo B é percorrido em "depth-first". Para cada nodo B_i visitado, se ele contiver mais de três pontos, é aplicado o algoritmo I2. O número cromático de G é o maior número cromático dos componentes biconexos considerados.

PODANDO A ÁRVORE DE SUBGRAFOS.

Um ou outro algoritmo de coloração baseado na idéia original de Christofides foi apresentado por Wang[45], em 1974. Wang demonstrou que, para cada nodo da árvore de subgrafos de G, é suficiente considerar um subconjunto de todos os seus CIM's, para obter uma coloração ótima de G. Assim, o número de ramificações a partir de cada nodo, em alguns casos, pode ser bastante reduzido. Vimos que tal procedimento já havia sido adotado, também, por Furtado et al.

TEOREMA VIII-4.

Para todo ponto v de G, sejam $C_1, C_2, \ldots, C_r$ os CIM's de G contendo v; então temos que:

$$\chi(G) = 1 + \min_{1 \leq i \leq k} \{ \chi(\langle V - C_i \rangle) \}.$$

DEMONSTRAÇÃO

Esse teorema foi demonstrado por Wang[45].

Devido a esse resultado, a árvore de subgrafos pode ser podada, colocando-se como filhos de um nodo apenas os CIM's que contêm um ponto v escolhido nesse nodo. Então, escolhemos v como o ponto que participa do menor número de CIM's do nodo considerado. Chamamos de árvore de subgrafos podada a árvore assim obtida. Ela não é única, ao contrário do que ocorre quanto à árvore completa, mas contém, pelo menos, um dos caminhos raiz-folha daquela, que representam coloridos independentes ótimos de G.

Uma implementação recursiva do trabalho de Wang é apresentada no quadro (VIII-6), algoritmo I3. Embora o algoritmo original de Wang seja iterativo, achamos mais adequado apresentá-lo recursivamente, porque a construção da árvore de subgrafos podada é efetuada na ordem "depth-first".

Um conjunto de classes de cor, $\{G_1,G_2,\ldots,G_k\}$, é tratado pelo algoritmo como um guia, representando o melhor colorido encontrado até o momento. Ao surgir um novo colorido com menos classes de cor, $P_1,P_2,\ldots,P_t$, onde $t<k$, o guia é substituído. No início do procedimento, inicializa-se o guia colocando-se um ponto em cada classe de cor, que é o pior caso possível.

Quadro VIII-6. Algoritmo exato I3.

```
begin                                                              L1
    procedure DESCE (H,t);                                         L2
    begin                                                          L3
        t:=t+1;                                                    L4
        if t > k then return; comment chegou no guia;              L5
        calcule os CIM's de <H> ;                                  L6
        if (<H> tem um único CIM S1) then                          L7
        begin comment chegou numa folha;                           L8
            P_t:=S_1;                                              L9
            k:=t; comment guarda o novo guia;                      L10
            for i:=1 to t do G_i:=P_t;                             L11
            return                                                 L12
        end                                                        L13
        escolha v presente no menor número de CIM's;               L14
        comment sejam S_1,S_2,...,S_r os CIM's contendo v;         L15
        for i:=1 to r do                                           L16
        begin                                                      L17
            P_t:=S_i;                                              L18
            DESCE (H-S_i,t);                                       L19
        end                                                        L20
    end DESCE;                                                     L21
    k:=n;                                                          L22
    for i:=1 to k do G_i:={i};                                     L23
    comment inicializou o guia;                                    L24
    DESCE (V,0)                                                    L25
end                                                                L26
```

OBSERVAÇÕES SOBRE I3.

A poda efetuada na árvore de subgrafos, pelo algoritmo I3, não diminui a complexidade em relação ao algoritmo anterior, uma vez que na pior situação a árvore "podada" será igual à árvore completa.

O fato de a pesquisa ser realizada em "depth-first" pode aumentar o número de nodos tratados pelo algoritmo, mas, na maioria das vezes diminui o espaço utilizado no armazenamento da árvore. Enquanto que em "breadh-first" tínhamos que manter armazenados todos os nodos de todos os níveis anteriores ao tratado, usando "depth-first" mantemos no nível j apenas os nodos dos filhos de exatamente um nodo do nível j-1, para $1 \leqslant j \leqslant t$, onde t é o nível pesquisado no momento.

No entanto, o espaço gasto pelo algoritmo I3 ainda é exponencial, como nos algoritmos anteriores.

USANDO PROGRAMAÇÃO DINÂMICA.

Em um artigo de 1976, a respeito de complexidade de algoritmos para grafos, Lawler[31] delineia um algoritmo para construir um colorido independente ótimo de um grafo qualquer,usando programação dinâmica.

Já vimos, no teorema (VIII-1), que uma das classes de cor de um colorido de G pode ser identificada com um CIM de G. Portanto, $\chi(G) = 1 + \chi(G-S)$, para algum CIM S de G.

Estendendo o resultado acima para o subgrafo <H>, onde H é um subconjunto não vazio de V, temos que $\chi(<H>)=1+\chi(<H-S>)$, para algum CIM S de <H>. Precisamos encontrar um conjunto independente maximal S de <H> tal que o valor $\chi(<H-S>)$ seja mínimo. Para isso, Lawler considera cada um dos CIM's de H, como já fazia Christofides, mas apresenta a solução por meio das seguintes equações de programação dinâmica:

$$\chi(<\phi>) = 0;$$

$$\chi(<H>) = 1 + \min \{\chi(<H-S>)\}, \quad H \neq \phi.$$

O algoritmo I4, mostrado no quadro (VIII-7), é uma implementação dessas equações.

Quadro VIII.7. Algoritmo exato I4.

```
begin                                                              L1
    procedure P(H,p); value H,p; set H;                            L2
    begin                                                          L3
        if K_H = - 1 then                                          L4
        begin                                                      L5
            if p = 0 then K_H = 0 else                             L6
            begin                                                  L7
                calcule todos os CIM's de <H>;                     L8
                comment sejam eles S_1,S_2,...,S_r;                L9
                k:=p;                                              L10
                for i:=1 until r do                                L11
                    k:=min (k,P(<H-S_i>, p-|S_i|));                L12
                K_H = k+1                                          L13
            end                                                    L14
        end                                                        L15
        return (K_H)                                               L16
    end P;                                                         L17
    inicialize o vetor K com -1;                                   L18
    P(G,n)                                                         L19
end                                                                L20
```

OBSERVAÇÃO.

O vetor K utilizado no algoritmo I4 é a "memória" da programação dinâmica. O elemento K_H é utilizado para armazenar o número cromático de <H>, quando esse número for calculado. K tem 2^n elementos e pode ser indexado pelo vetor característico de H em relação a G.

COMPLEXIDADE DE I4 (E DE TODOS OS ALGORITMOS Ik).

Seja $p = |H|$. O algoritmo dos quatro japoneses gasta, no pior dos casos, um tempo $O(m_p\ p3^{\ p/3})$ para gerar todos os CIM's de H, onde m_p é o número de linhas de <H>.

Maximizamos m_p para m, somamos o valor $mp3^{p/3}$ para todos os subconjuntos H de V e aplicamos o teorema binomial, obtendo:

$$\sum_{p=0}^{n} \binom{n}{p} m\ p\ 3^{\ p/3} < m\ n \sum_{p=0}^{n} \binom{n}{p} 3^{p/3} = (1+\sqrt[3]{3})^n.$$

Logo, a complexidade de I4 é $O(mn(1+\sqrt[3]{3})^n)$. Esse resultado é aplicável a todos os algoritmos que fornecem coloridos independentes de G.

OBSERVAÇÕES SOBRE I4.

Entendemos que, ao sugerir esse algoritmo de coloração, Lawler não pretendia melhorar sobre os procedimentos já existentes, nem propor um procedimento novo.

O algoritmo I4 é do tipo dos algoritmos vistos nessa seção e pode atuar pior que os mesmos.

O objetivo era formular o processo, de uma forma que facilitasse o cálculo da complexidade para essa classe de algoritmos.

A complexidade de I4, conforme apresentamos acima, foi determinada por Lawler[31] no trabalho já mencionado e constitui a primeira estimativa de complexidade, para algoritmos de coloração, já publicada.

UM ALGORITMO DE COLORAÇÃO COM ESPAÇO POLINOMIAL.

O número de conjuntos independentes maximais de um grafo qualquer é de ordem exponencial, como já observamos ante

riormente. Decorre disso que qualquer algoritmo de coloração que obtenha e armazene simultaneamente os CIM's de G e de subgrafos de G apresentará, necessariamente, espaço exponencial.

Precisamos, portanto, de um procedimento que gere os CIM's de um grafo, de uma forma tal que a cada solicitação seja fornecido um único CIM, caso exista, que garantidamente não tenha sido fornecido anteriormente. O algoritmo dos quatro japoneses[44] a que já nos referimos é um procedimento do tipo desejado.

No quadro (VIII-8), apresentamos o algoritmo I5, desenvolvido por Szwarcfiter[42], em 1978, que consiste em uma adaptação do algoritmo de Christofides, para aproveitar a propriedade seqüencial de geração de CIM's conseguida pelos quatro japoneses.

Quadro VIII-8. Algoritmo exato I5.

```
begin                                                              L1
    procedure P(H) comment H é um subgrafo de G;                   L2
    begin set S, V ;                                               L3
        for S ε (conjunto dos CIM's de G)                          L4
        do                                                         L5
            if t < k then                                          L6
            begin if | S | = n then k:=t else                      L7
                begin                                              L8
                    t:=t+1;                                        L9
                    n:= n - | S | ;                                L10
                    V:=V-S; comment G = < V > ;                    L11
                            P (G);                                 L12
                    V:=V U S;comment G = < V > ;                   L13
                    N:=n+ |S|;                                     L14
                    t:=t-1;                                        L15
                end                                                L16
            end                                                    L17
    end P;                                                         L18
    t:=1; k:=n;                                                    L19
    P (G);                                                         L20
end                                                                L21
```

OBSERVAÇÕES SOBRE I5.

O algoritmo I5 gasta espaço polinomial, podendo ser implementado com uma complexidade de espaço de O(m+n). Os detalhes de uma tal implementação e outras considerações a respeito do próprio algoritmo serão dadas no capítulo (IX), que será inteiramente dedicado ao estudo do I5.

VIII-3. Coloridos por Enumeração.

O primeiro algoritmo que veremos nesse grupo foi apresentado por Brown[7], em 1972, como uma aplicação de técnicas de enumeração de soluções parciais, que evitam a obtenção de soluções redundantes.

No caso, uma solução parcial consiste em um colorido aproximado de G. Uma solução redundante é um colorido que apresenta o mesmo conjunto de classes de cor de algum colorido apresentado anteriormente.

O outro algoritmo a ser apresentado nesta seção consta da obra de Nijenhuis e Wilf[37], de 1975, e fornece como resultado todos os coloridos possíveis de G, usando k cores, para G e k dados.

MELHORANDO AS SOLUÇÕES PARCIAIS.

O algoritmo E1, mostrado no quadro (VIII-9), obtém, como primeira solução aproximada, um colorido seqüencial, para uma ordenação inicial de pontos, tal que um ponto v_i qualquer de G possui mais pontos adjacentes no conjunto $\{v_i, v_2, \ldots, v_{i-1}\}$ do que no conjunto $\{v_{i+1}, \ldots, v_n\}$. Empates são resolvidos, se possível, em favor do ponto com maior grau.

Suponhamos que nessa solução foram utilizadas k cores. A solução é armazenada e inicia-se a procura de uma outra solução que use menos do que k cores. Essa pesquisa utiliza a técnica de "backtracking", da seguinte forma. Seja v_p o primeiro ponto colocado na k-ésima classe de cor. Tenta-se colocar o ponto anterior a v_p (na ordenação inicial) numa classe de cor de número maior do que aquela em que está esse ponto e menor do que k. Se isso for possível, continua-se como no colorido seqüencial. Se não for possível, tenta-se a mesma coisa com o

ponto anterior ao considerado.

Se uma nova solução for determinada usando menos do que k cores, ajusta-se k para esse novo valor, armazena-se a solução e o processo é recomeçado.

Uma tentativa de usar a k-ésima cor durante o processo de recoloração também leva a uma reconsideração do ponto anterior, da forma já descrita.

O algoritmo termina quando o "backtracking" atinge o ponto 1, uma vez que colocá-lo em outra classe não pode melhorar em nada a solução.

No quadro (VIII-9), apresentamos o algoritmo E1, que é uma implementação recursiva da idéia acima exposta.

O conjunto D_i consiste do conjunto de cores que estão disponíveis para utilização no ponto v_i, dentre as cores já usadas até o momento. Tratando-se da primeira determinação de D_i, ele conterá também a próxima cor, seguinte às já usadas. A variável L_i é o número de cores usadas para colorir o subgrafo $<1,2,\ldots,i>$ e $\ell = L_i$ durante a iteração de B, para o ponto v_i.

Ao terminar o algoritmo, $k = \chi(G)$ e $g_1, g_2, \ldots, g_n$ são as cores atribuídas a $v_1, v_2, \ldots, v_n$.

Quadro VIII-9. Algoritmo exato E1.

```
begin  procedure  B(i);  comment  parte  de  v_i;                    L1
       begin  if  i:=1  then  stop                                    L2
              l:=L_i;                                                 L3
              for  i:=i  until  n  do                                 L4
              begin  if  D_i = φ  then  B(i-1);                       L5
                     escolha d, a menor cor em D_i;                   L6
                     if  d ≥ k  then  B(i-1);                         L7
                     D_i:=D_i - {d};                                  L8
                     c_i:=d;  comment  colore v_i com d;              L9
                     if  d > l  then  l:=l+1;                         L10
                     L_i:=l;                                          L11
                     determine  D_{i+1}                               L12
              end;                                                    L13
              for  i:=1 until n do g_i:=c_i; comment guarda            L14
                   a solução parcial obtida;                          L15
              k:=l; comment guarda a estimativa de χ(G);              L16
              i:=1; while  c_i ≠ k do  i:=i+1;                        L17
              B(i-1)                                                  L18
       end B;                                                         L19
       c_1:=1; L_1,L_2:=1; D_1:=φ; determine D_2;                     L20
       k:=n;                                                          L21
       B(2)                                                           L22
end                                                                   L23
```

OBSERVAÇÕES SOBRE E1.

Brown[7] mostra como diminuir o número de chamadas recursivas à procedure B, através de uma determinação mais elaborada de D_i.

Não conseguimos, ainda, obter uma estimativa da complexidade de E1. Observamos, apenas, que o pior caso deverá ocorrer para grafos que não dão bons coloridos seqüenciais. E sabemos que, para coloridos seqüenciais, existem grafos tão ruins quanto se queira.

PESQUISANDO EXAUSTIVAMENTE.

O algoritmo E2, que vemos no quadro (VIII-10), enumera todas as possibilidades de se colorir G com k, ou menos, cores, também aplicando "backtracking". E2 foi publicado em 1975, por Nijenhuis e Wilf.

Mais precisamente, dados um grafo G e um inteiro k, E2 fornece todos os vetores $c_1, c_2, \ldots, c_n$, onde $1 \leq c_i \leq k$, que representam coloridos válidos de G.

Quadro VIII-10. Algoritmo exato E2.

```
begin                                                              L1
    procedure B(i,cont); integer value (i,cont);                   L2
    begin                                                          L3
        for l:=1 until cont do                                     L4
        begin                                                      L5
            c_i ⇐ pilha; comment coloriu v_i;                      L6
            if i=n then output (c_1,c_2,...,c_n) else              L7
            begin comment acha os candidatos à                     L8
                cor do ponto v_{i+1};                              L9
                for j:=1 until k do cor(j):=true;                  L10
                for j:=1 until i do                                L11
                    if a_{j,i+1}=1 then cor(c_j):=false;           L12
                for j:=1 until k do                                L13
                    if cor(j) then                                 L14
                    begin                                          L15
                        cont:=cont+1;                              L16
                        pilha ⇐ j                                  L17
                    end                                            L18
                B(i+1,cont)                                        L19
            end                                                    L20
        end                                                        L21
    end B;                                                         L22
    pilha ⇐ 1;                                                     L23
    B(1,1)                                                         L24
end                                                                L25
```

COMPLEXIDADE DE E2.

Na linha L6, uma das k cores é atribuída ao ponto v_i, não havendo nenhum pnto v_j adjacente a v_i, com essa mesma cor, para j=1,2,...,i-1. Vemos que a procedure B é chamada uma vez para cada cor atribuída a cada ponto de G.

Se for aplicado a um grafo totalmente desconexo, E2 atribuirá ao ponto v_1 a cor 1 e, a cada um dos demais pontos, todas as cores 1,2,...,k. Nesse caso, B será chamada $O(k^{n-1})$ vezes. Se assumirmos que G é um grafo conexo, o número máximo de chamadas à procedure B será $O((k-1)^{n-1})$, ocorrendo para grafos que têm dois pontos com grau um e os demais pontos com grau dois.

DESEMPENHO DE E2.

Para um dado grafo G, se o valor de k for menor do que $\chi(G)$, o algoritmo E2 não fornecerá nenhum colorido de G. Por outro lado, se $k \geq \chi(G)$, os coloridos fornecidos por E2, utilizando o menor número de cores, serão os coloridos ótimos de G. Se o objetivo for determinar $\chi(G)$, é conveniente usar-se para k uma estimativa do número cromático de G, obtida através de um algoritmo de coloração aproximativo.

VIII-4. Coloridos por Redução.

Os dois algoritmos considerados a seguir baseiam-se em uma propriedade do número cromático de um grafo qualquer, estabelecida por Zikov, em 1949. Eles foram apresentados por Corneil e Graham[12], em 1973.

DEFINIÇÃO VIII-6.

Seja G um grafo não completo. Dados dois pontos não adjacentes x e y de G, os grafos reduzidos de G são os grafos G'_{xy} e G''_{xy}, assim definidos:

(i) G'_{xy} é obtido de G acrescentando-se a linha xy;

(ii) G''_{xy} é obtido de G tornado idênticos os pontos de x e y.

Vemos um exemplo na figura (VIII-3).

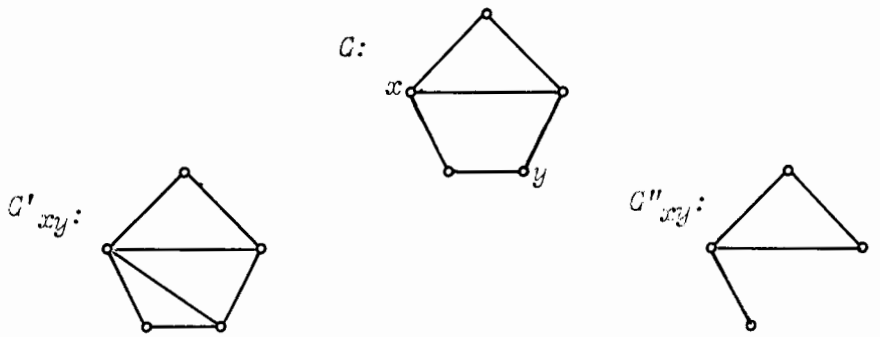

Figura VIII-3. Exemplo de grafos reduzidos de G.

DEFINIÇÃO VIII-4.

Uma árvore de Zikov para um grafo G é uma árvore binária Z, assim definida:

(i) G é a raiz de Z;

(ii) Seja H um nodo qualquer de Z. Se H é um grafo completo, então H é uma folha de Z. Caso contrário, H tem por filhos os grafos reduzidos de H, em relação a um par de pontos não adjacentes quaisquer de H.

CONSIDERAÇÕES SOBRE A ÁRVORE DE ZIKOV.

Na figura (VIII-4) apresentamos uma árvore de Zikov completa, para o grafo já visto na figura VIII-3.

A altura da árvore é igual ao número de linhas não presentes no grafo inicial, ou seja, $O(\bar{m})$, onde $\bar{m}$ é o número de linhas do grafo complementar de G. Como a árvore é binária, um limite superior para o número de nodos é dado por

$$\sum_{l=0}^{\bar{m}} 2^{l} = \left(\frac{\bar{m}+1}{2}\right) 2^{\bar{m}}.$$

Observemos que, se ao tornarmos dois pontos idênticos, guardarmos os números dos mesmos no ponto resultante, cada folha da árvore indicará um colorido de G em que cada ponto é uma classe de cor.

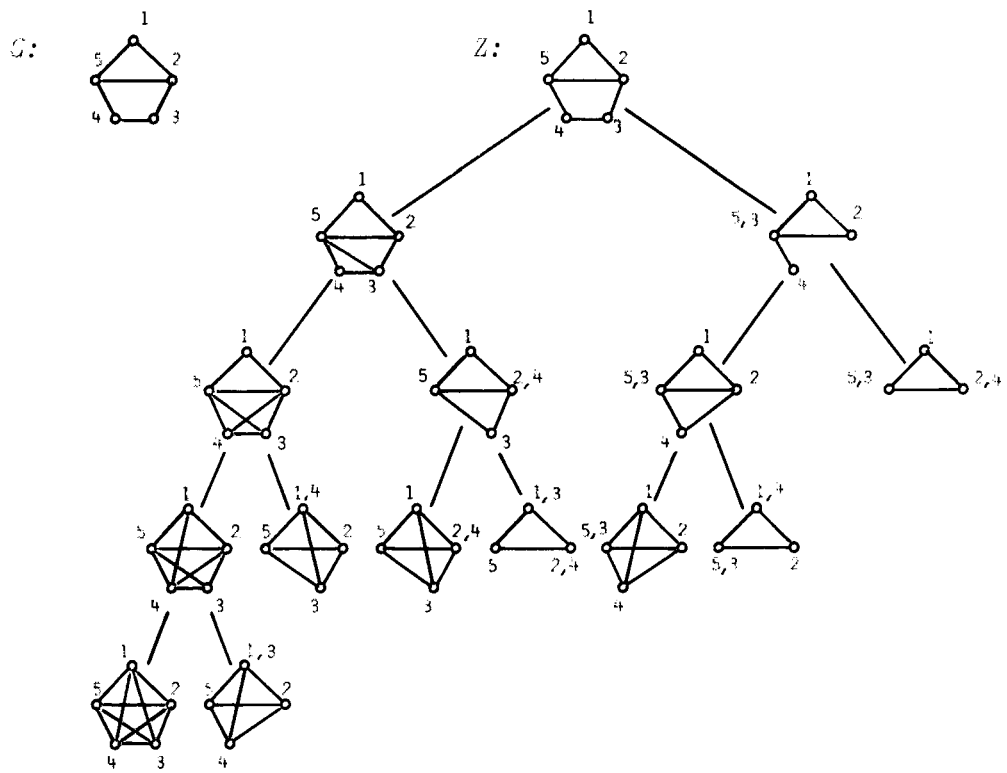

Figura VIII-4. Árvore de Zikov para G.

TEOREMA VIII-5.

Se x e y são dois pontos não adjacentes de G, então $\chi(G) = \min \{ \chi (G'_{xy}), \chi (G''_{xy}) \}$.

DEMONSTRAÇÃO.

Apresentada por Zikov e referenciada por Corneil e Graham[12].

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

O algoritmo R1, quadro (VIII-11), é uma aplicação imediata do teorema de Zikov.

Quadro VIII-11. Algoritmo exato R1.

```
begin                                                              L1
    procedure REDUZA (H,p)                                         L2
    begin                                                          L3
        if k > p then k:=p;                                        L4
        if (H não é um grafo completo) then                       L5
        begin                                                      L6
            sejam x,y dois pontos não adjacentes de H;             L7
            forme H'xy e H"xy;                                     L8
            REDUZA (H'xy,p);                                       L9
            REDUZA (H"xy,p-1)                                      L10
        end                                                        L11
    end  REDUZA;                                                   L12
    k:=limite superior para χ(G);                                  L13
    REDUZA (G,k)                                                   L14
end                                                                L15
```

COMPLEXIDADE DE Rl = $O(2^{\bar{m}})$.

Já vimos que $2^{\bar{m}}$ é um limite superior para o número de nodos da árvore de Zikov. Para construir toda a árvore de Zikov, Rl gasta um tempo proporcional ao seu número de nodos.

OBSERVAÇÃO SOBRE Rl.

Enquanto que nos algoritmos de coloração independente apenas um subconjunto das possíveis soluções era pesquisado, os algoritmos de coloração por redução podem produzir uma solução qualquer.

PODANDO A ÁRVORE DE ZIKOV.

Se em uma das reduções aplicadas nos nodos da árvore de Zikov chega-se a um grafo completo com k pontos, conclui-se que k é um limite superior para $\chi(G)$. Daí por diante, qualquer outro nodo da árvore que contenha uma k-clique não precisará mais ser reduzido, pois o número cromático do grafo desse nodo será maior ou igual a k, não interferindo no limite já encontrado para $\chi(G)$.

Dessa forma, executa-se uma poda na árvore de Zikov, tendo-se, no entanto, a certeza de que os ramos que estão sendo derrubados não abaixariam o limite superior estimado para $\chi(G)$.

O diagrama da figura (VIII-5) apresenta uma árvore de Zikov podada, em que não foi preciso calcular o valor de $\chi(A)$ e $\chi(B)$ para se descobrir o valor de $\chi(G)$.

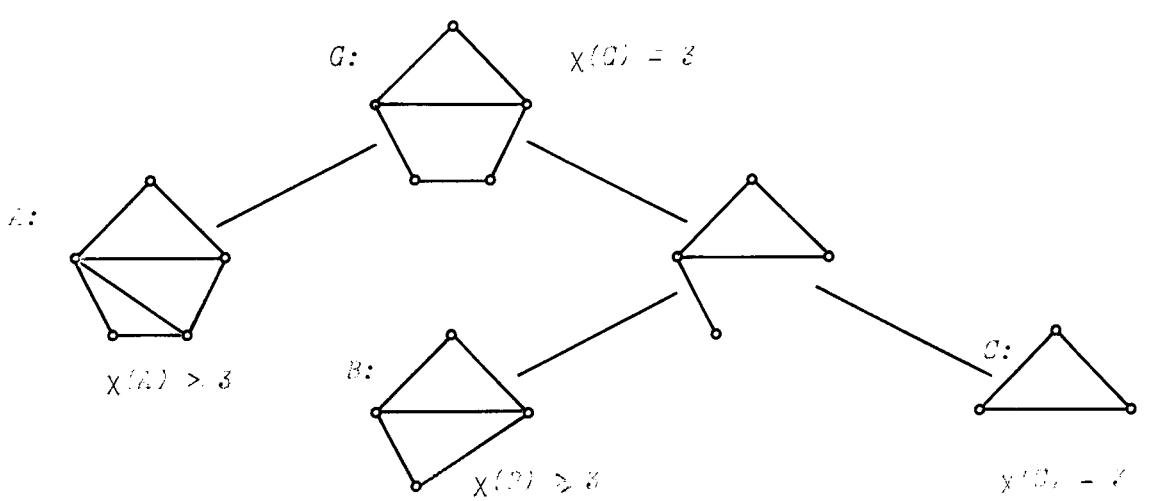

Figura VIII-5. Árvore de Zikov podada.

FORÇANDO O APARECIMENTO DE CLIQUES.

O algoritmo R2, quadro (VIII-12), que consideraremos a seguir, obtém o número cromático de um grafo qualquer G pelo processo de redução, construindo uma árvore de Zikov podada de G. Z2 é de autoria de Corneil e Graham[12], tendo sido publicado em 1973.

Antes de apresentarmos o algoritmo, resta ainda observar que para descobrir, a um dado ponto da redução, se um grafo contém uma k-clique, poderia ser gasto tanto tempo que não compensaria fazer a poda da árvore. Por isso, Corneil e Graham adotaram a estratégia de procurar um k-cluster ao invés de uma k-clique.

Enquanto que o problema de encontrar cliques é NP-completo (Karp[27]), o problema de encontrar clusters pode ser resolvido em tempo polinomial.

Encontrado um k-cluster, ao invés de se podar o ramo como se faria se fosse uma k-clique, prossegue-se reduzindo,mas utilizando-se para isso pontos não adjacentes do k-cluster. Com esse procedimento, força-se o aparecimento de uma clique com k pontos, caso em que o ramo é podado, ou com menos pontos, sendo feito o ajuste para o novo limite superior de $\chi(G)$.

Quadro VIII-12. Algoritmo exato R2.

```
begin                                                              L1
    procedure REDUZA (H,p)                                         L2
    comment H é um grafo e p seu número de pontos;                 L3
    begin                                                          L4
        if k > p then k:=p;                                        L5
        k':=k;                                                     L6
        if (H não é um grafo completo) then                        L7
        begin                                                      L8
            encontre H_k, um k-cluster                             L9
            while (H não é uma clique) do                          L10
            begin escolha x,y ε H_k não adjacentes;                L11
                forme H'_xy e H"_xy a partir de H;                 L12
                H:=H'_xy;                                          L13
                REDUZA (H"_xy,p-1);                                L14
                if k'≠ k then REDUZA (H,p);                        L15
            end                                                    L16
        end                                                        L17
    end REDUZA;                                                    L18
    k:= limite superior para χ (G);                                L19
    REDUZA (G,n);                                                  L20
end                                                                L21
```

COMPLEXIDADE DE R2 = $O(2^{\bar{m}})$.

Após encontrar um k-cluster de H, linha L10, o algoritmo R2 pesquisa o segundo filho de H, H''_{xy}, reduzindo-o até encontrar uma clique.O primeiro filho de H, H'_{xy}, só será pesquisado se a clique obtida acima tiver menos do que k pontos, como vemos na linha L16.

Existem situações em que R2 tem que construir a árvore completa de Zikov, tendo que pesquisar os dois filhos de cada nodo não-folha. Seja visto, por exemplo, o grafo com n pontos, todos de grau 2, compondo um ciclo de comprimento n.

Portanto, embora apresentando desempenho melhor que R1 na maioria dos casos, o algoritmo R2 é de mesma ordem que aquele no pior caso.

OBSERVAÇÕES SOBRE R2.

Um limite superior inicial é requerido para $\chi(G)$. O mais simples dos limites superiores, n, pode ser utilizado, ou qualquer outro mais justo, obtido, por exemplo, por um algoritmo aproximativo.

Segundo seus autores, utilizando-se alocação dinâmica de memória, foi possível implementar o algoritmo R2 em um espaço $O(n^3)$.

O método empregado para encontrar cluster tem complexidade $O(n^3)$.

OUTRAS SUGESTÕES.

Hedetniemi[25] sugere outros procedimentos para tornar mais rápido o algoritmo original de Zikov. Por exemplo, poder-se-ia retirar, de cada grafo intermediário, os pontos adjacentes a todos os demais, ou retirar um ponto cuja lista de adjacência está contida na lista de adjacência de algum outro ponto.

Boaventura[4] também descreve o algoritmo de Zikov, e mostra que, para determinadas k-cliques pode-se concluir que $\chi(G)=k$.

VIII-5. Colorido Seqüencial Exato.

Como fizemos para algoritmos aproximativos, vamos, agora, analisar um algoritmo exato de coloração, para uma classe especial de grafos. Neste caso, trata-se de grafos cordais.

DEFINIÇÃO VIII-5.

Dado um ciclo de G, cujo comprimento é maior do que três, chamamos de corda desse ciclo uma linha de G, que não pertence ao ciclo e que liga dois pontos do mesmo.

DEFINIÇÃO VIII-6.

G é um grafo cordal, se qualquer ciclo de G de comprimento maior do que três contém uma corda.

Gavril[22] apresenta um algoritmo de coloração exata, para grafos cordais, publicado em 1972. Para entendermos como o algoritmo funciona, precisamos ainda da seguinte definição.

DEFINIÇÃO VIII-7.

Uma R-orientação das linhas de um grafo é uma orientação dessas linhas, satisfazendo as duas condições seguintes:

(i) o grafo direcionado resultante não possui ciclos direcionados;

(ii) se $b \rightarrow a$ e $c \rightarrow a$, então $b \rightarrow c$ ou $c \rightarrow b$.

O algoritmo S6, apresentado no quadro (VIII-13), atribui um colorido seqüencial a um grafo cordal G qualquer, usando o menor número possível de cores, $k=\chi(G)$.

No início do procedimento, as linhas do grafo recebem uma R-orientação, o que é sempre possível, em se tratando de grafos cordais. Ao mesmo tempo, os pontos vão sendo numerados, de forma tal que todas as linhas estarão direcionadas do menor para o maior. Um procedimento simples, que efetua tal numeração em $O(n^4)$, é mostrado por Gavril[22].

Quadro VIII-13. Algoritmo exato S6.

```
begin                                                          L1
    ordene os pontos de G, de acordo com                       L2
    uma R-ordenação;                                           L3
    c_1:=1;                                                    L4
    for i:=2 until n do                                        L5
    begin                                                      L6
        r:= min (j | A_i ∧ C_j = φ)                            L7
        c_i:=r; comment colore v_i com a cor r;                L8
        k:= max (k,r)                                          L9
    end                                                        L10
end                                                            L11
```

COMPLEXIDADE DE S6.

Da linha L4 à L10, o que temos é exatamente o algoritmo S1, que vimos ser O(m+n), no capítulo anterior.

Como L2-L3 gasta um tempo $O(n^4)$, concluímos ser essa a complexidade de S6.

EXEMPLO DE APLICAÇÃO DE S6.

O grafo da figura (VIII-6) é um grafo cordal G que recebeu uma R-orientação e foi numerado de forma que cada linha vai do ponto menor para o maior.

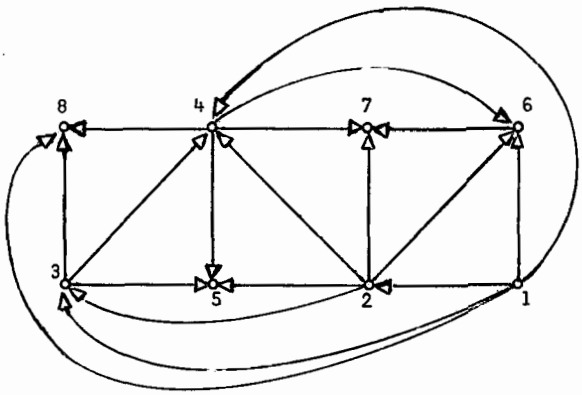

Figura VIII-6. Grafo cordal para aplicação de S6.

O algoritmo S6 nos fornece um 4-colorido de G com as seguintes classes de cor: {1,5,7}, {2,8}, {3,6}, {4}.

OBSERVAÇÃO SOBRE S6.

S6 fornece sempre um colorido exato. Seja v_p o primeiro ponto colocado na última classe C_k. Então, existiam pontos $v_1, v_2, \ldots, v_{k-1}$ adjacentes a v_p e menores que v_p, que foram colocados anteriormente nas classes $C_1, C_2, \ldots, C_{k-1}$ respectivamente. As linhas que ligam esses pontos a v_p são todas dirigidas para v_p, pois v_p é maior que tais pontos. Logo, pela definição de R-orientação, cada par de pontos v_i, v_j acima, $i \neq j$ e $1 \leq i,j \leq k-1$, é ligado por uma linha. Conclui-se que $<\{v_1, v_2, \ldots, v_{k-1}, v_p\}>$ é uma clique e $\chi(G) \geq k$.

IX. UM NOVO ALGORITMO DE COLORAÇÃO.

O algoritmo que descreveremos neste capítulo é uma modificação do algoritmo proposto originalmente por Christofides, classificando-se, portanto, no grupo dos coloridos independentes, definido no capítulo anterior.

Ele foi desenvolvido, no final de 1978, por Szwarcfiter[42], cuja motivação ao fazê-lo pode ser atribuída, em parte, à pesquisa que efetuamos para a confecção da presente tese.

Houve quem afirmasse que qualquer algoritmo de colorido independente deveria utilizar, necessariamente, um espaço de memória exponencial (Corneil e Graham[12]). Entretanto, é possível implementar o algoritmo I5 aqui exposto em espaço polinomial. A chave para isso consiste em se conseguir tratar seqüencialmente os conjuntos independentes maximais de um grafo, sem que haja necessidade de colocá-los ao mesmo tempo na memória.

Pelo algoritmo dos quatro japoneses[44] já se podia obter os CIM's de um grafo G qualquer, um de cada vez, por um procedimento que explicaremos detalhadamente na seção(IX-1). Faltava encontrar um esquema em que, à medida que cada CIM fosse produzido, ele pudesse ser de pronto utilizado no processo de construção da árvore de subgrafos de G.

Na seção (IX-2), faremos uma breve revisão de como é obtido um colorido independente de G e, na seção (IX-3), mostraremos de que forma o algoritmo I5 entrosa tal procedimento com o dos quatro japoneses, para conseguir o efeito desejado.

A apresentação do algoritmo será feita na seção (IX-4), de uma forma que julgamos mais apropriada para facilitar a compreensão inicial do mesmo. Em seguida, ao considerarmos a complexidade de espaço, alteraremos aos poucos a versão apresentada e sugeriremos as estruturas de dados a serem empregadas, chegando-se, assim, a uma definição mais precisa do algoritmo, que nos permitirá esclarecer como foi obtida a estimativa de espaço anunciada.

Até aqui, o algoritmo estará fornecendo apenas o número cromático de um grafo qualquer, utilizando, para isso, espaço

polinomial com o tamanho do grafo. Na última seção, serão acrescentados alguns comandos, para que I5 forneça também as classes de cor de um colorido ótimo de G, sem com isso prejudicar a estimativa de espaço estabelecida.

A forma de apresentação do algoritmo e o estudo da complexidade são devidos a Szwarcfiter[42].

IX-1. A Árvore dos Quatro Japoneses.

Todos os CIM's de um grafo G qualquer podem ser obtidos através do seguinte procedimento (Lawler[31]).

Seja I_j o conjunto de todos os CIM's do subgrafo induzido de G $<\{1,2,\ldots,j\}>$. Então, $I_1 = \{\{1\}\}$. A idéia é encontrar I_{j+1} a partir de I_j até chegar-se a I_n, que é o conjunto procurado.

Seja $c=|I_n|$. É fácil ver que $|I_1|<|I_2|<\ldots<|I_n|=c$.

Seja $S \varepsilon I_j$. Então, ou $SU\{j+1\} \varepsilon I_{j+1}$, no caso em que $S \wedge A_{j+1} = \{\}$, ou $S \varepsilon I_{j+1}$, no caso em que $S \wedge A_{j+1} \neq \{\}$.

Por outro lado, seja $S' \varepsilon I_{j+1}$. Se $j+1 \notin S'$, então $S' \varepsilon I_j$. Se $j+1 \varepsilon S'$, então para algum $S \varepsilon I_j$ temos que $S' - \{j+1\} \subseteq S$. O importante dessas relações é que elas nos mostram que um CIM qualquer S' em I_{j+1} ou já pertencia a I_j, ou pode ser derivado de algum CIM S em I_j pela fórmula: $S' = (S-A_{j+1}) \cup \{j+1\}$. Unindo todos os conjuntos S', obtidos dessa forma, com todos os conjuntos S pertencentes a I_j teremos uma família de conjuntos F que conterá todos os conjuntos de I_{j+1}, $F = \{S' \mid S'=(S-A_{j+1})U\{j+1\}, S \varepsilon I_j\} \cup I_j$.

Um jeito de extrair de F os elementos de I_{j+1} seria comparar todos eles, dois a dois, selecionando apenas os maximais e evitando repetições. Tal procedimento não nos seria conveniente, pois exigiria armazenar simultaneamente toda a família F.

O outro jeito de obter I_{j+1} a partir de F, que foi concebido pelos quatro japoneses, consiste :

(i) verificar cada $S'=(S-A_{j+1})U\{j+1\}$, $S \varepsilon I_j$, quanto à possibilidade de incluir um ponto $l<j+1$ não existente em S, de forma que $S'U\{l\}$ continue sendo um conjunto independente de I_{j+1}; S' é um CIM de I_{j+1}, se e somente se não existir um pon-

to ℓ nessas condições, isto é, se dado qualquer $\ell<j+1$, $\ell \notin I$, tivermos $A_\ell \wedge S' \neq \{\ \}$;

(ii) estabelecer uma ordenação dos conjuntos S', de forma a podermos identificar, no momento da sua geração, o primeiro de uma série de conjuntos iguais. Sejam $S'_1=(S_1-A_{j+1})\cup\{j+1\}$ e $S'_2=(S_2-A_{j+1})\cup\{j+1\}$ dois conjuntos iguais da família F, onde $S_1, S_2 \in I_j$. Disporemos S'_1 à esquerda de S'_2, na ordenação proposta, se e somente se o conjunto de pontos subtraídos em S_1-A_{j+1} for lexicograficamente menor que o conjunto subtraído em S_2-A_{j+1}. Estamos, portanto, comparando os conjuntos $S_1 \wedge A_{j+1}$ e $S_2 \wedge A_{j+1}$. Para sabermos se um dado $S'=(S-A_{j+1})\cup\{j+1\}$ é o primeiro de uma série de possíveis conjuntos iguais, deveremos descobrir se $S \wedge A_{j+1}$ é o menor de todos os conjuntos $S_i \wedge A_{j+1}$ onde $S_i \in I_j$, para os quais $S_i - A_{j+1}$ são iguais a $S - A_{j+1}$. O que o algoritmo faz, na realidade, ao invés de calcular S' e em seguida descobrir se ele deve ser selecionado por ser o primeiro, é tomar $S \in I_j$ e descobrir se o conjunto S', a ser gerado a partir dele, será selecionado. Isso ocorrerá, se e somente se não existir um ponto $\ell < j+1$ não pertencente a S, tal que

(1) $A_\ell \wedge (S-A_{j+1}) = \phi$ e

(2) $A_\ell \wedge (S \wedge A_{j+1}) \wedge \{1,2,\ldots,\ell-1\} = \phi$.

Observemos que, se existir um $\ell<j+1$, $\ell \notin S$, atendendo à condição (1) acima, significa que ℓ é adjacente a um ou mais pontos e que todos eles estão em S e também em A_{j+1}. Sejam $q_1,q_2,\ldots,q_t$ esses pontos, onde $t > 1$. Deduz-se daí, que existe um CIM Z em I_j contendo ℓ ao invés de $\{q_1,q_2,\ldots,q_t\}$ e contendo os demais pontos de S, tal que $Z-A_{j+1} = S-A_{j+1}$.

Resta-nos comparar $Z \wedge A_{j+1}$ com $S \wedge A_{j+1}$. Ora, $\ell \in Z \wedge A_{j+1}$, $\ell \notin S \wedge A_{j+1}$, $\{q_1,q_2,\ldots,q_t\} \in S \wedge A_{j+1}$ e $\{q_1,q_2,\ldots,q_t\} \notin Z \wedge A_{j+1}$. Logo, $Z \wedge A_{j+1}$ será lexicograficamente menor do que $S \wedge A_{j+1}$, se todos os pontos $q_1,q_2,\ldots,q_t$ forem maiores do que ℓ. Isso equivale a dizer que ℓ satisfaz a condição (2) acima.

Arrumando as equações (1) e (2) acima, resumimos dizendo que S satisfaz a condição lexicográfica, se e somente se para todo $\ell < j+1$, $\ell \notin S$,

$$A_\ell \wedge (S-(A_{j+1} \wedge \{\ell+1, \ell+2,\ldots,j\})) \neq \phi.$$

O processo de obtenção de I_n a partir de I_1, através do algoritmo dos quatro japoneses, pode ser visualizado como a construção de uma árvore binária, em que os CIM's de I_j correspondem aos nodos do nível j, considerando que a raiz está no nível 1.

DEFINIÇÃO IX-1.

Dado um grafo G qualquer, com conjunto de pontos - $\{1,2,\ldots,n\}$, a árvore dos quatro japoneses correspondente a G é uma árvore binária Q assim definida:

(i) $\{1\}$ é a raiz da árvore;

(ii) seja S um nodo qualquer de Q situado no nível j.Se j=n, então S é uma folha. Para $1 \leqslant j < n$, S tem um filho S' à esquerda, definido por $S'=(S-A_{j+1}) \cup \{j+1\}$, se o conjunto independente S' for maximal em $<\{1,2,\ldots,j+1\}>$ e S atender à condição lexicográfica; S tem um filho à direita S" = S, se $S \wedge A_{j+1} \neq \{\}$.

CONSIDERAÇÕES SOBRE A ÁRVORE DOS QUATRO JAPONESES.

Cada nodo não-folha da árvore dos quatro japoneses tem, pelo menos, 1 filho, visto que, para $S \in I_j$, ou S'=S pertence a I_{j+1} ou $S'=S \cup j+1$ pertence a I_{j+1}, para $1 \leqslant j \leqslant n$.

Além disso, existe uma correspondência um-a-um entre os CIM's de G e as folhas da árvore. Com isso, se construirmos a árvore dos quatro japoneses em pré-ordem, que é o "depth-first" para árvores binárias, no instante em que cada folha for visitada teremos obtido um novo CIM de G.

Na figura (IX-1), apresentamos um grafo G e a árvore dos quatro japoneses correspondente.

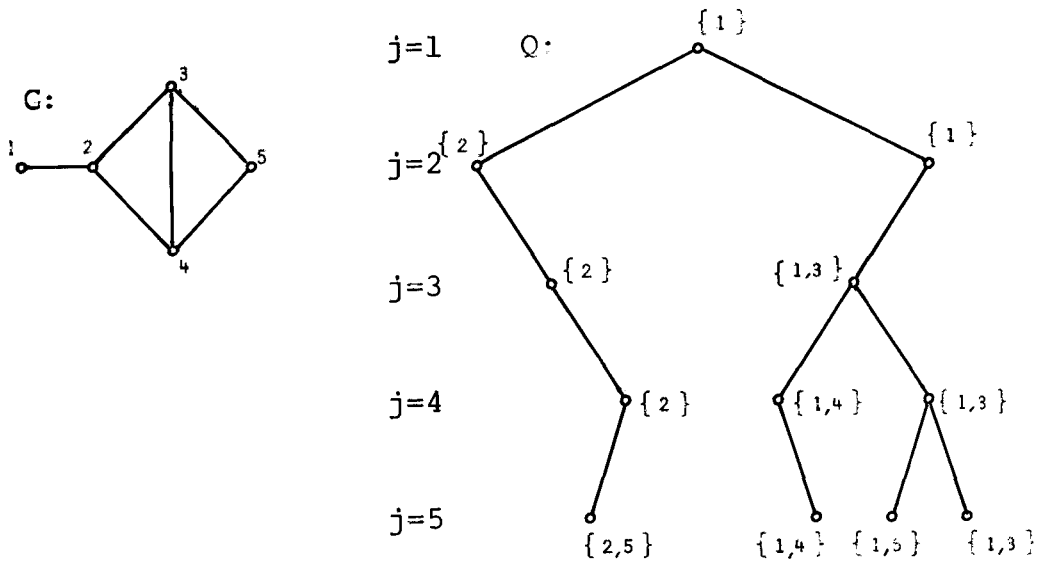

Figura IX-1. A árvore dos quatro japoneses para G.

IX-2. A Árvore de Christofides.

Vamos nos referir à árvore de subgrafos de um grafo G, definida no capítulo (VIII), por árvore de Christofides.

Vimos que cada nodo da árvore de Christofides corresponde a um subgrafo de G, encontrado durante o processo de obtenção de coloridos independentes para G. Um galho, ligando um nodo do nível j com um nodo do nível j+1, corresponde a um CIM do primeiro desses nodos. As classes de cor de um colorido independente de G são os galhos que ligam a raiz a uma folha qualquer da árvore de Christofides. Se G possui um k-colorido exato, G possui também um k-colorido independente exato, que fica determinado ao se encontrar uma folha que esteja no menor nível da árvore, em relação às demais.

Na figura (IX-2), mostramos a árvore de Christofides para o mesmo grafo utilizado na figura (IX-1). Convencionamos que cada nodo é o subgrafo maximal de G, induzido pelos pontos dispostos dentro do círculo correspondente.

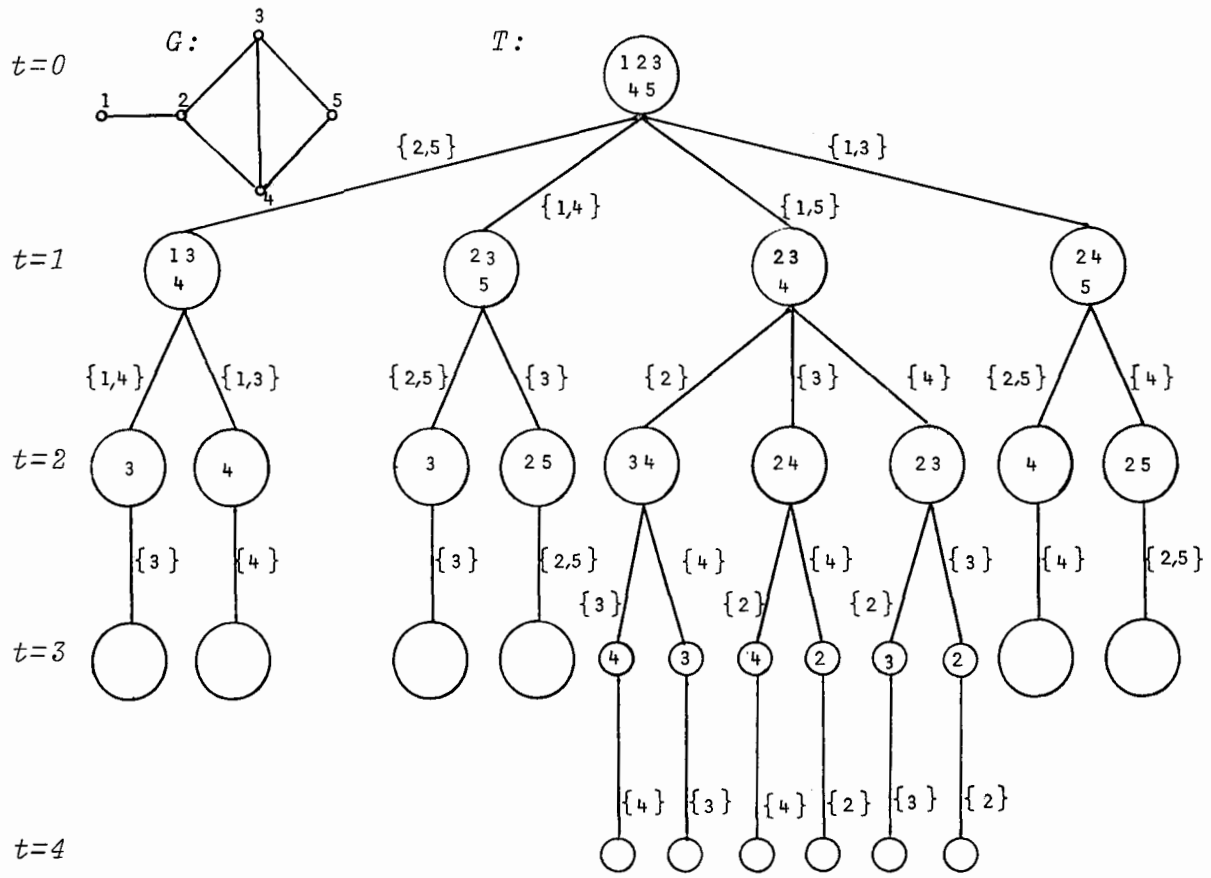

Figura IX-2. A árvore de Christofides para G.

O algoritmo I5, de Szwarcfiter, constrói a árvore de Christofides acima usando a técnica "depth-first", como o faz o algoritmo I3, de Wang. Ao contrário desse último, no entanto, ao invés de guardar para cada nível todos os nodos filhos de um dado nodo do nível anterior (um "feixe de balões" para cada nível, na figura (IX-2)), I5 guarda tão somente um nodo de cada nível, à medida que desce na árvore (ou melhor, que sobe para níveis maiores, rumo às folhas).

Finalizamos esta seção observando que a altura da árvore de Christofides é, no pior dos casos, igual a n. Um algoritmo para percorrê-la recursivamente por "depth-first" gastará, no mínimo, um espaço O(n), para implementação da pilha de nodos.

IX-3. A Árvore de Szwarcfiter.

Para mostrar como o algoritmo I5 efetua o devido entrosamento entre o algoritmo dos quatro japoneses e o algoritmo de

Christofides, exibiremos a árvore de Szwarcfiter para um grafo G, que consiste no entrelaçamento adequado da árvore de Christofides, T, para G, com várias árvores dos quatro japoneses, uma para cada nodo não-folha de T, e que substituem os galhos de T.

DEFINIÇÃO IX-2.

Dado um grafo G qualquer, uma árvore de Szwarcfiter para G é uma árvore W, contendo nodos do tipo T, identificados com subgrafos de G, e nodos do tipo Q, identificados com subconjuntos de pontos de G, da seguinte forma:

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

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

(iii) agora, seja H um nodo de W do tipo Q. Se H for uma folha na árvore dos quatro japoneses a que pertence, então H tem, exatamente, um filho H' na árvore W, sendo H', do tipo T, o subgrafo induzido pelos pontos que sobram ao tirarmos H do conjunto de pontos do mais próximo nodo do tipo T ascendente de H. Caso contrário, isto é, se H não for uma folha na árvore Q dos quatro japoneses, então H tem um ou dois filhos em W, que são os mesmos filhos de H em Q.

Na figura (IX-3), apresentamos a árvore de Szwarcfiter, para o grafo G que temos usado como exemplo.

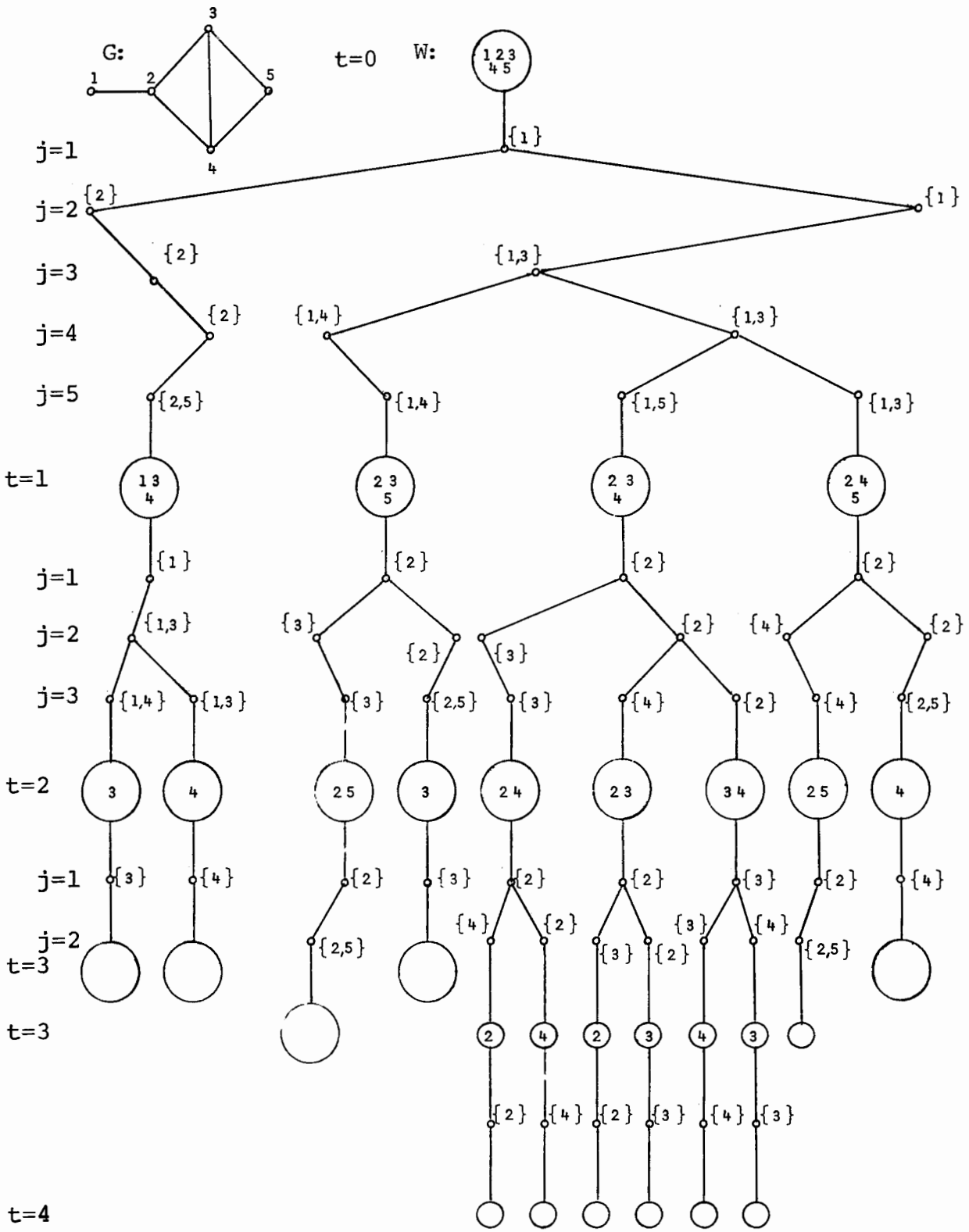

Figura IX-3. A árvore de Szwarcfiter para G.

OBSERVAÇÕES SOBRE A ÁRVORE DE SZWARCFITER.

Tivemos que alterar ligeiramente a definição da árvore dos quatro japoneses, para podermos construí-la a partir de um subgrafo H de G. Ao invés de usar {1} como raiz da árvore, ordenamos os pontos de H numa seqüência $v_1, v_2, \ldots, v_p$, tal que $v_1 < v_2 < \ldots < v_p$ e usamos $\{v_1\}$ como raiz.

Um limite superior para a altura da árvore de Szwarcfiter é dado por $n+(n-1)+(n-2)+\ldots+1 = \frac{n(n+1)}{2}$, ou seja $O(n^2)$.

IX-4. O Algoritmo e sua Complexidade.

Apresentamos o algoritmo I5 no quadro (IX-1).

Ele constrói a árvore de Szwarcfiter, para um grafo G qualquer, sendo t o nível da árvore de Christofides e j o nível da árvore dos quatro japoneses em que se está a cada instante.

Como de costume, k é a melhor aproximação já obtida para $\chi(G)$, a cada instante, sendo igual a $\chi(G)$, ao findar a execução de I5.

Quadro IX-1. Algoritmo exato I5.

```
begin  procedure P(S,j); value S,j; set S; integer j;                    L1
    comment S é um CIM de <{v1,v2,...,vj}>;                              L2
    begin set R;                                                         L3
        j:=j+1;                                                          L4
        if  j ≤ n  then                                                  L5
        begin  R:=(S-A(vi))U{vi};                                        L6
            if MAX (S,R,j) and SMALLEST(S,j) then P(R,j);                L7
            if  S Λ A(vj) ≠ φ then  P(S,j)                               L8
        end  else  comment S é um CIM de G;                              L9
            if  t < k  then                                              L10
            begin  if  | S |= n  then k:=t else                          L11
                begin                                                    L12
                    t:t+1;                                               L13
                    N:=n-|S|;                                            L14
                    G:=G - <S>;                                          L15
                       P ( { v1 },1);                                    L16
                    G:=G+<S>;                                            L17
                    n:=n+|S|;                                            L18
                    t:=t-1;                                              L19
                end                                                      L20
            end                                                          L21
    end  P;                                                              L22
    classifique cada lista de adjacência de G por                       L23
    valor crescente dos rótulos dos pontos;                              L24
    t:=1;  K:=n;                                                         L25
    P ( { v1 }, 1)                                                       L26
end                                                                      L27
```

```
logical procedure MAX (S,R,j);                                   L28
 value S,R,j:  set S,R; integer j;                               L29
begin  MAX := true;                                              L30
     for v_l ε V and v_l ∉ S and  l ≤ j  do                      L31
         if A(v_l) ∧ R = φ then  MAX := false                    L32
end  MAX;                                                        L33
logical  procedure SMALLEST (S,j);                               L34
value  S,j;  set S; integer j;                                   L35
begin SMALLEST:=  true                                           L36
     for v_l ε V and v_l ∉ S and l ≤ j do                        L37
     begin  R:= { v_i ε V | l<i≤j} ;                             L38
          if  A(v_l) ∧ (S-(A(v_j)∧R))= φ                         L39
          then SMALLEST := false                                 L40
     end                                                         L41
end  SMALLEST;                                                   L42
```

OBSERVAÇÕES SOBRE I5.

A operação G-<S> indicada na linha L15 consiste em se obter o subgrafo de G induzido pelo conjunto de pontos V-S; a partir de então designados por $v_1, v_2, \ldots, v_p$, onde $v_1 < v_2 < \ldots < v_p$.

A operação G+<S> indicada na linha L17 consiste em se recompor o grafo existente antes da execução de L15, preservando a sua rotulação inicial.

COMPLEXIDADE DE I5 = $O(mn\ (1+\sqrt[3]{3})^n)$.

Esse é o limite superior calculado por Lawler[30] e que se aplica a qualquer dos algoritmos do grupo dos coloridos independentes.

IX-5. Complexidade de Espaço.

A profundidade máxima de recursão efetuada por I5 é igual a $O(n^2)$, pois essa é, conforme vimos, a altura da árvore

de Szwarcfiter. Isso nos dá a primeira aproximação para a estimativa do espaço necessário.

1^{a} APROXIMAÇÃO = $O((n+m)n^2)$ espaço.

Isso corresponde a se guardar, para cada nível da recursão, o grafo considerado.

Não há necessidade de se guardar, para cada nível da recursão, o grafo considerado. A idéia consiste, como indica o algoritmo, em se calcular G - <S> e, posteriormente, efetuar a recuperação do grafo original, pela operação G+<S>. Como S é preservado entre essas duas operações, não há, inclusive, necessidade de espaço adicional para S.

Basta se retirar de A(w) as linhas (v_i,w) tais que $v_i \in S$ e $w \notin S$, deixando intactas as listas A(v). Além disso, deve-se criar um elo de v_{i-1} para v_{i+1}, a fim de se compor o novo conjunto de pontos de G.

A operação G + <S> deve executar o procedimento inverso, tomando-se apenas o cuidado de efetuar as inserções das linhas, de forma a manter as listas de adjacência ordenadas, do menor ponto para o maior. Isso é suficiente para se preservar a rotulação original.

Chegamos, assim, a uma aproximação mais justa para a estimativa de espaço.

2^{a} APROXIMAÇÃO = $O(n^3)$ espaço.

Para cada nível da recursão, reserva-se apenas O(n) espaço, para armazenar os conjuntos de pontos S e R.

É possível implementar-se o algoritmo, de tal modo que em cada nível da recursão o espaço requerido seja tão somente O(c), onde c é uma constante. Se não, vejamos.

A variável R, aparentemente local à "procedure" P pode ser transformada em global, uma vez que, entre a sua geração e a sua utilização, linhas L6 e L7 do algoritmo, respectivamente, não é efetuada nenhuma chamada recursiva.

O parâmetro S também pode ser transformado em variável global, sendo recalculado, quando necessário, sem aumentar a

complexidade de tempo do algoritmo.

Assim, no início da "procedure" P, antes do comando - j:=j+1, calcula-se S. Se j=1, S é simplesmente $\{v_1\}$. Caso contrário, necessitaremos ter conhecimento se P foi invocada por P(R,j), linha L7, ou por P(S,j), linha L8. Isso pode ser resolvido facilmente pela adição de uma variável booleana global. Se P foi invocado por P(S,j), então S permanece inalterado. Se P foi invocado por P(R,j), então obtém-se S da equação - $S=(S-A(v_j))\ U\ \{v_j\}$.

Há outro ponto em que S deve ser recalculado. No ponto de retorno, imediatamente após P(R,j), na linha L7, S é recuperado pela equação $S=(S-\{v_j\})\ U\ (\ (S-\{v_j\})\ \Lambda\ A\ (v_j))$, que é operação inversa à que foi efetuada na linha L6. É evidente que no ponto de retorno após P(S,j), na linha L8, não há necessidade de recalcular S, uma vez que S não foi alterado por essa chamada.

Resta-nos considerar apenas a chamada $P(\{V_1\},1)$ da linha L16. Precisaremos guardar S numa pilha imediatamente antes de $P(\{v_1\},\ 1)$ e recuperá-lo imediatamente após a $P(\{v_1\},1)$. Considerando que os CIM's que compõem a configuração dessa pilha num certo instante qualquer do processo são necessariamente disjuntos, tal pilha tem tamanho máximo O(n).

Obtivemos, assim, a terceira aproximação para a complexidade de espaço de I5.

3^{a} APROXIMAÇÃO = $O(n^2)$ espaço.

Aqui, apenas a profundidade de recursão é considerada, pois no esquema descrito a "procedure" teria apenas o parâmetro j. As diversas referências feitas a P(R,j), P(S,j) e $P(\{v_1\},1)$, tanto no texto acima como no algoritmo, deveriam ser substituídas por P(j), P(j) e P(1), respectivamente. Entretanto, continuaremos fazendo uso da notação anterior para faciliar a localização do ponto correspondente no algoritmo.

ELIMINANDO A RECURSÃO.

É possível implementar o algoritmo I5 com complexidade de espaço O(n+m).

Obviamente, como a profundidade da recursão é $O(n^2)$, o

espaço mínimo requerido é $O(n^2)$, enquanto a recursão for mantida. Para reduzi-lo ainda mais, teria que ser considerada a possibilidade de traduzir-se a recursão para iteração, nos moldes de Knuth e Szwarcfiter[28] e Knuth[29]. Nessa tradução seria preciso evitar a construção de pilhas para armazenar o parâmetro da "procedure", j, e os pontos de retorno, a fim de conseguirmos um O(n+m) espaço. Observemos que não há outras variáveis locais. Vejamos como isso pode ser feito.

PARÂMETRO J.

Ao parâmetro j faz-se corresponder uma variável simples j, assim calculada:

(i) nos pontos da versão iterativa correspondentes aos pontos imediatamente anteriores às chamadas $P(\{v_1\},1)$, linhas L16 e L26 do algoritmo, faz-se j:=1;

(ii) no ponto correspondente ao início de P, linha L4, mantêm-se j:=j+1;

(iii) no ponto correspondente a end P, linha L22, faz-se j:=j-1;

(iv) imediatamente após o ponto correspondente ao comando n:=n+|S|, na linha L18, recupera-se o valor de j anterior à chamada da linha L16, ou seja, faz-se j:=n+1.

PONTOS DE RETORNO.

Existem quatro possíveis pontos de retorno, para quais a versão iterativa desviaria, após a execução da declaração correspondente a end P. Esses pontos seriam os imediatamente seguintes a: (1) $P(\{v_1\},1)$, na linha L26;
(2) $P(\{v_1\},1)$, na linha L16;
(3) P(R,j), na linha L7 e
(4) P(S,j), na linha L8.

O primeiro pode ser identificado pelas condições j=1 e n= número de pontos do grafo dado.

O segundo ponto pode ser identificado pelas condições j=1 e n<número de pontos do grafo dado.

Não sendo satisfeitas as condições acima, o seguinte procedimento pode ser adotado, para se decidir entre o terceiro e o quarto pontos de retorno. Durante a execução do processo, consideramos o conjunto S no ponto correspondente a end P , linha

L27. Seja S' o valor de S correspondente ao mesmo ponto end P, na iteração imediatamente anterior à em consideração. Se S' $\neq$ S, o desvio em questão seria para o terceiro ponto e, caso contrário, para o quarto. O custo adicional de se armazenar S' é, naturalmente, apenas O(n).

IX-6. Composição das Classes de cor.

Para que o algoritmo I5 forneça um colorido de G correspondente ao seu número cromático, procede-se da seguinte forma.

A cada vez que o valor de k for atualizado, por ter sido encontrada uma folha num nível menor que as anteriores, esse fato é memorizado, para que, ao se seguir de volta em direção à raiz, os conjuntos S sejam identificados com as classes de cor do k-colorido correspondente de G.

A memorização a que nos referimos pode ser implementada por meio de um vetor booleano estado, de comprimento O(n) , sendo um elemento do vetor para cada nível da árvore de Christofides. Ao passar por um nível t em direção às folhas, faz-se estado (t):=false. Devido ao teste da linha L10, uma folha só é atingida se ela estiver num nível menor que as folhas atingidas anteriormente, ou se for a primeira folha mais à esquerda da árvore. Seja qual for o caso, tendo-se atingido uma folha, faz-se estado (t):= true. Ao passar por um nível t em direção à raiz, se para algum dos filhos do nodo considerado tiver causado um estado (t+1):=true, então faz-se estado (t):=true e associa-se ao conjunto S, em vigor naquele instante, a classe de cor t.

No quadro (IX-2), mostramos o algoritmo I6, que é uma versão alterada de I5, ocupando espaço $O(n^2)$ e fornecendo, além do número cromático, um colorido independente exato de G. As rotinas MAX e SMALLEST são como em I5.

Naturalmente, não exibiremos a versão iterativa do algoritmo I5, que mostramos ter complexidade de espaço igual a O(n+m).

Quadro IX-2. Algoritmo exato I6.

```
begin global R,S,b; set R,S; boolean b;                                  L1
    procedure P(j); integer value j;                                     L2
    begin                                                                L3
        if j=1 then S:={v1}                                              L4
        else if b then S:=(S-A(vj))U{vj});                               L5
        comment S é um CIM de <{v1,v2,...,vj}>;                          L6
        j:=j+1;                                                          L7
        if j ⩽ n then                                                    L8
        begin                                                            L9
            R:=(S-A(vj))U{vj};                                           L10
            if MAX (S,R,j) and SMALLEST (S,j) then                       L11
            begin                                                        L12
                b:= true                                                 L13
                P(j);                                                    L14
                S:=(S-{v})U((S-{vj})∩A(vj)),                             L15
            end                                                          L16
            if S∩A(vj) ≠ ϕ then                                          L17
            begin                                                        L18
                b:= false;                                               L19
                P(j);                                                    L20
            end                                                          L21
        end else comment S é um CIM de G;                                L22
            if t ⩽ k then                                                L23
            begin if |S|=n then                                          L24
                begin comment o grafo dado                               L25
                    é k-colorável;                                       L26
                    k:=t;                                                L27
```

```
                    estado (t) := true;                          L28
                end  else                                        L29
                begin                                            L30
                    estado (t) := false;                         L31
                    t:=t+1;                                      L32
                    n:= n- |S|;                                  L33
                    G:=G - <S>;                                  L34
                    coloque S na pilha;                          L35
                        P(1);                                    L36
                    retire  S  da  pilha;                        L37
                    G:=G+ <S>;                                   L38
                    n:=n + |S|;                                  L39
                    t:=t-1;                                      L40
                    estado (t) := estado (t)                     L41
                        or  estado (t+1);                        L42
                    if estado (t) then atribua                   L43
                    aos pontos de S a cor t;                     L44
                end                                              L45
            end                                                  L46
        end  P;                                                  L47
        classifique cada lista de adjacência de                  L48
        G por valor crescente dos rótulos dos pontos             L49
        t:=1;  k:=n;                                             L50
        P(1);                                                    L51
end                                                              L52
```

X. CONSIDERAÇÕES FINAIS.

"Podemos" resolver o problema de coloração de grafos, assim como qualquer outro que envolva objetos combinatórios, pelo método da força bruta. Aqui, o método da força bruta pode ser implementado da seguinte forma.

Seja G um grafo com n pontos.

Começando com k=1 e somando 1 a cada etapa, verifique, para cada valor de k, as k^n formas possíveis de atribuir as k cores aos n pontos, até encontrar uma que seja um k-colorido de G.

É verdade que tal trabalho pode dar uma mão-de-obra imensurável a priori , constituindo o único consolo saber-se que o processo terminará em um tempo finito, pois, para k=n, existe um k-colorido de G.

O algoritmo da força-bruta que acabamos de delinear tem complexidade $O(n^n)$. É importante determinar a complexidade dos algoritmos exatos de coloração, para se conhecer o quanto cada algoritmo é menos brutal que um método qualquer, não trabalhado.

Não é fácil, no entanto, obter estimativas dessa complexidade, que sejam o bastante justaspara caracterizar adeqüadamente cada método. Nesse sentido, o trabalho de Lawler[30] é pioneiro.

Por outro lado, estudos que consistem em um levantamento dos algoritmos já propostos e na tentativa de determinação de suas complexidades, como é o caso desta tese, têm o mérito de estimular o surgimento de novos algoritmos.

Algoritmos novos para coloração exata podem surgir dentro de uma das seguintes categorias:

(i) algoritmos para colorir um grafo qualquer, que se enquadram em algum dos grupos já existentes; têm complexidade exponencial e apresentam, como inovação, ou uma função exponencial inferior para o tempo, ou a complexidade de espaço menor do que a determinada para o grupo em questão;

(ii) algoritmos para colorir classes especiais de gra-

fos, a exemplo de S6; vimos que esses algoritmos podem ser muito bons, com complexidade polinomial; espera-se que surja um algoritmo eficiente para colorir grafos planares;

(iii) algoritmos para colorir um grafo qualquer, que apresentem uma abordagem do problema, totalmente nova; têm, certamente, complexidade exponencial.

Acreditamos que algoritmos novos, para coloração de grafos quaisquer, devam aparecer mais na categoria (i) do que na (iii), dentro de grupos que vêm apurando sua linhagem, e cujo representante mais significativo é o dos coloridos independentes.

Quanto a procedimentos aproximados de coloração, as heurísticas, não vemos como algum trabalho novo poderia ser desenvolvido. Ao que tudo parece, o que havia para ser feito já o foi. Em vista do perfil traçado para os algoritmos aproximativos, acreditamos, como Williams, que não valha a pena mexer mais neles.

Encerramos, apresentando, na figura (X-1) um quadro cronológico de todos os algoritmos vistos.

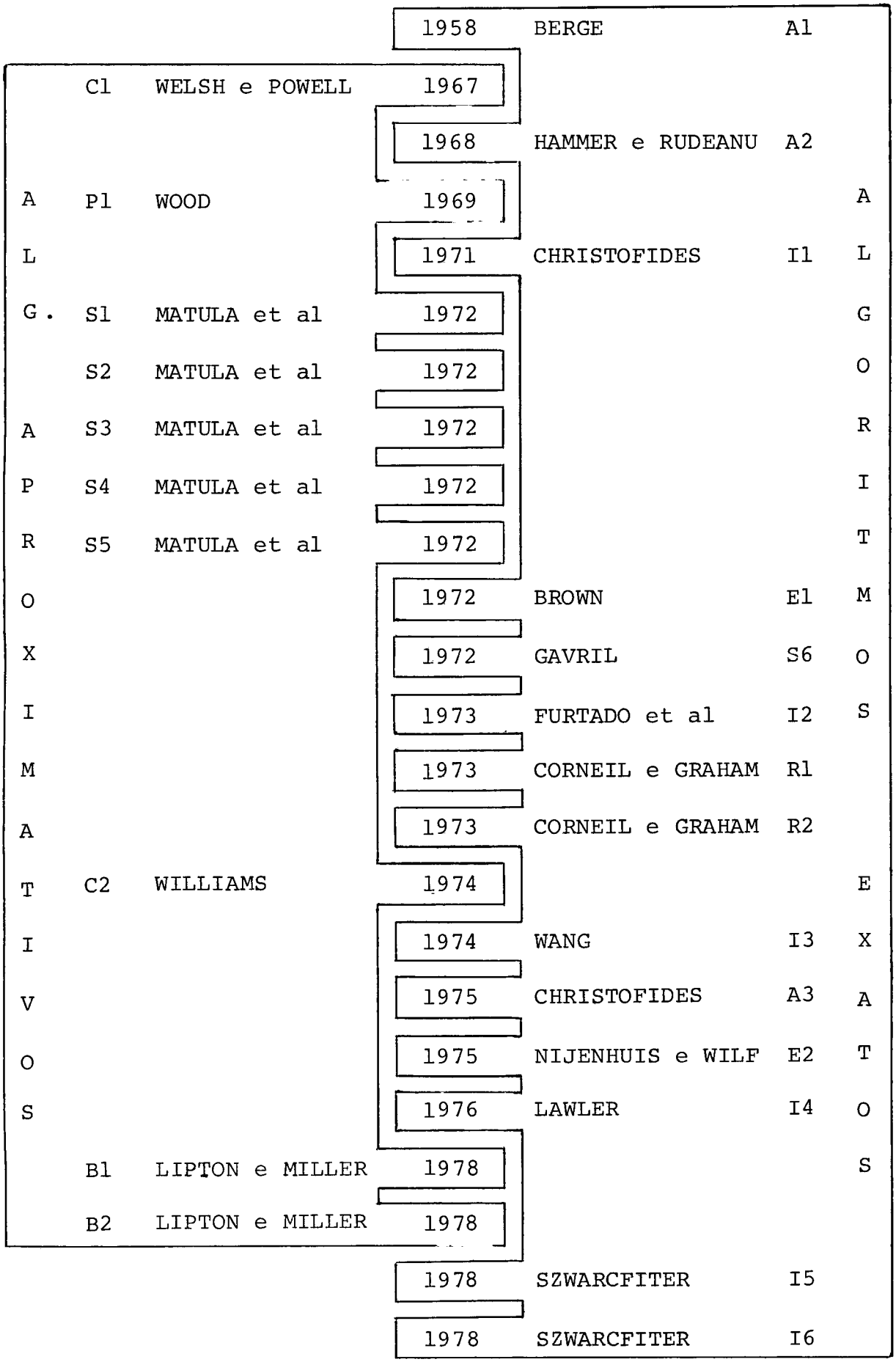

Figura X-1. Relação cronológica dos algoritmos vistos.

REFERÊNCIAS BIBLIOGRÁFICAS.

1. AHO, A. V., HOPCROFT, J.E. e ULLMAN, J.D., The Design and Analysis of Computer Algorithms, Addison-Wesley, Reading, Massachusetts, 1974.

2. BERGE, C., Théorie des Graphes et ses Applications. Dunod, Paris, 1958.

3. BERNHART, F. R., "A digest of the four color theorem", Journal of Graph Theory, volume 1, págs.207-225, 1977.

4. BOAVENTURA NETTO, P.O., Teoria dos Grafos, COPPE/UFRJ, Rio de Janeiro, 1977.

5. BONDY, J. A., "Bounds for the chromatic number of a graph", Journal of Combinatorial Theory, volume 7, pags. 96-98, 1969.

6. BROOKS, R. L. "On colouring the nodes of a network , Proceedings of the Cambridge Philosophical Society, volume 37, págs.194-197, 1941.

7. BROWN, J. R., "Chromatic scheduling and the chromatic number problem", Management Science, volume 19, págs. 456-463, 1972. APÊNDICE J.

8. CHRISTOFIDES, N., "An algorithm for the chromatic number of a graph", The Computer Journal, volume 14, págs. 38-39, 1971.

9. CHRISTOFIDES, N., Graph Theory, an Algorithmic Approach, Academic Press, Londres, 1975. APÊNDICE F.

10. COLE, A. J., "The preparation of examination timetables using a small-store computer", The Computer Journal, volume 7, págs. 117-121, 1964

11. COOK, S. A., "The Complexity of theorem-proving procedures", Procedings of the Third ACM Symposium on Theory of Computing, págs. 151-158, 1971.

12. CORNEIL, D.G. e GRAHAM, B., "An algorithm for determining the chromatic number of a graph", SIAM Journal on Computing, volume 2, págs. 311-318, 1973. APÊNDICE L.

13. CSIMA, J. e GOTLIEB, C. C., "Tests on a computer method for constructing school timetables", Communications of the ACM, volume 7, págs. 160-163, 1964.

14. DEMPSTER, M. A. H., "Two algorithms for the time-table problem", Combinatorial Mathematics and its Applications,
Editor: D.J.A.Welsh, Academic Press, New York, págs. 63-85, 1969.

15. DEO, N., Graph Theory with Applications to Engineering and Computer Science,
Prentice Hall, Englewood Cliffs,
New Jersey, 1974.

16. EVEN, S., Algorithmic Combinatorics,
Macmillan, New York, 1973.

17. FLOYD, R. W., "Nondeterministic algorithms", Journal of the ACM, volume 14, págs. 636-644, 1967.

18. FOXLEY, E. e LOCKYER, K., "The construction of examination timetables by computer", The Computer Journal, volume 11, págs. 264-268, 1968.

19. FURTADO, A. L., Teoria dos Grafos - Algoritmos, Livros Técnicos e Científicos, Rio de Janeiro, 1973.

20. GAREY, M. R., JOHNSON, D. S. e STOCKMEYER, L., "Some simplified NP-complete graph problems" Proceedings of the Sixth ACM Symposium on Theory of Computing, págs. 47-63, 1974.

21. GAREY, M. R. e JOHNSON, D. S., "The complexity of near-optimal graph coloring", Journal of the ACM, volume 23, págs. 43-49, 1976.

22. GAVRIL, F., "Algorithms for minimum coloring, maximum clique, minimum covering by cliques, and maximum independent set of a chordal graph", SIAM Journal on Computing, volume 1, págs.180-187, 1972. APÊNDICE M.

23. HAMMER (IVANESCU), P.L. e RUDEANU, S., Boolean Methods in Operations Research, Springer-Verlag, 1968.

24. HARARY, F., Graph Theory, Addison Wesley, Reading, Massachusetts, 1972.

25. HEDETNIEMI, S.T., Review 22063, Computing Reviews, volume 12, págs. 446-447, 1971.

26. KARP, R.M., "Reducibility among combinatorial problems", Complexity of Computer Computations, editores: R.E. Miller e J.W. Thatcher, Plenum Press, New York, págs. 85-104, 1972.

27. KARP, R. M., "On the computational complexity of combinatorial problems", Networks, volume 5, págs. 45-68, 1975.

28. KNUTH, D. E. e SZWARCFITER, J. L., "A structured program to generate all topological sorting arrangements", Information Processing Letters, volume 2, págs. 153-157, 1974.

29. KNUTH, D. E., "Structured Programming with go to statements", ACM Computing Surveys, volume 6, págs. 261 - 301, 1974.

30. LAWLER, E. L., "A note on the complexity of the chromatic number problem", Information Processing Letters, volume 5, págs. 66-67, 1976. APÊNDICE I.

31. LAWLER, E. L., "Graphical algorithms and their complexity", Foundations of Computer Science II, editores: K.R. Apt e J.W. de Bakker, Mathematical Centre, Amsterdam, págs. 3-32, 1976.

32. LIPTON, R. J. e Miller, R.E., "A batching method for coloring planar graphs", Information Processing Letters, volume 7, págs.185-188, 1978. APÊNDICE E.

33. MATULA, D. W., "A min-max theorem for graphs with application to graph coloring", SIAM Review, volume 10, págs. 481-482, 1968

34. MATULA, D. W., MARBLE, G. e ISAACSON, J. D., "Graph coloring algorithms", Graph Theory and Computing, editor: R.C. Read, Academic Press, New York, págs. 109-122, 1972. APÊNDICE A.

35. MITCHEM, J., "On various algorithms for estimating the chromatic number of a graph", The Computer Journal, volume 19, págs. 182-183, 1976.

36. MOON, J. W. e MOSER, L., "On cliques in graphs", Israel Journal of Mathematics, volume 3, págs. 23-28, 1965.

37. NIJENHUIS, A. e WILF, H. S., Combinatorial Algorithms, Academic Press, New York, 1975. APÊNDICE K.

38. PECK, J.E.L. e WILLIAMS, M. R., "Examination Scheduling", algoritmo 286, Communications of the ACM, volume 9, págs. 133-134, 1966.

39. ROSCHKE, S.I. e FURTADO, A.L., "An algorithm for obtaining the chromatic number and an optimal coloring of a graph", Information Processing Letters, volume 2, págs. 34-38, 1973. APÊNDICE G.

40. ROUSE-BALL, W. W., Mathematical Recreations & Essays, The Macmillan Company, New York, 1947.

41. SZEKERES, G. e WILF, H.S., "An inequality for the chromatic number of a graph", Journal of Combinatorial Theory, volume 4, págs. 1-3, 1968.

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

43. TARJAN, R.E., "Complexity of combinatorial algorithms", Technical Report, STAN-CS-77-609, Stanford University, 1977.

44. TSUKIYAMA, S., IDE, M., ARIYOSHI,H. e SHIRAKAWA, I., "A new algorithm for generating all the maximal independent sets",SIAM Journal on Computing, volume 6, págs. 505-517, 1977.

45. WANG, C. C., "An algorithm for the chromatic number of a graph", Journal of the ACM, volume 21, págs. 385-391, 1974. APÊNDICE H.

46. WELSH, D. J. A. e POWEL, M. B., "An upper bound for the chromatic number of a graph and its aplication to timetabling problems", The Computer Journal, volume 10, págs.85-86, 1967. APÊNDICE B.

47. WILLIAMS, M.R., "The colouring of very large graphs", Combinatorial Structures and Their Applications, editor: R. Guy, Gordon and Breach, New York, Págs. 477-478.

48. WILLIAMS, M.R., "Heuristic procedures (if they work,leave them alone)", Software - Practice and Experience, volume 4, págs. 237-240, 1974. APÊNDICE C.

49. WOOD, D.C., "A technique for colouring a graph applicable to large scale timetabling problems", The Computer Journal, volume 12, págs. 317-319, 1969. APÊNDICE D.

Apêndice A. ALGORITMOS S1, S2, S3, S4 e S5;
Matula, Marble e Isaacson.

Sequential Vertex Colorings.

A graph G = (V,E) with vertex set V and edge set E will herein be assumed to have no loops or multiple edges. For $A \subset V$, the induced subgraph $< A >$ = (A,E') of G will be the subgraph of G, where E' contains all edges of E, both end points of which are in A. Also $< v_1, v_2, \ldots, v_j >$ will denote $< \{v_1, v_2, \ldots v_j\} >$. A k coloring of G is an assignment of colors to the vertices of G using no more than k colors and such that adjacent vertices have different colors. For an ardering $v_1, v_2, \ldots, v_n$ of the vertices V of G, a sequencial coloring of G corresponding to this order is a k coloring of G utilizing each of the colors 1,2,..., k determined recursively as follows:

(1) v_1 is assigned color 1, thus 1 coloring $< v_1 >$;

(2) if $< v_1, v_2, \ldots, v_{i-1} >$ has been j colored, then $v_1, \ldots, v_{i-1}$ are assigned the same colors in $< v_1, \ldots, v_i >$, and v_i is assigned color m, where $m \leq j + 1$ is the minimum positive integer not occurring on adjacent vertices in $< v_1, \ldots v_i >$. Thus, $< v_1, \ldots v_i >$ is j colored for $m \leq j$, and (j + 1), and (j + 1) colored otherwise.

(...) Let the degree of v in the graph G, $\deg_G(v)$ (deg (v) when G is understood), be the number of adjacent vertices of v in G.

(...) An ordering of the vertices of a graph G such that $\deg(v_i) > \deg(v_{i+1})$ for $1 \leq i < |V(G)|$ will be called a largest-first (LF) ordering of the vertices. Determination of a sequencial coloring corresponding to such an ordering will be termed the largest-first algorithm (LF algorithm). The sequential coloring corresponding to a given LF ordering will effect the same coloring as described by the algorithm of Welsh and Powell |10| and will utilize no more than $\max_i \min \{i, 1+\deg(v_i)\}$ colors. Note that since the LF ordering of the vertices is not necessarily unique, the number of colors utilized in the coloring provided by the LF algorithm can vary depending on the particular LF

ordering chosen. Application of the LF algorithm will mean its application for a particular largest-first ordering.

(...) A closer inspection of the sequential coloring procedure shows that for a given ordering $v_1, v_2, \ldots, v_n$ of the vertices of a graph G, the corresponding sequential coloring algorithm could never require more than k colors where

$$(3)\quad k = \max_{1 \leq i \leq n} \{1 + \deg_{<v_1, v_2, \ldots, v_i>}(v_i)\} .$$

The determination of a vertex ordering minimizing k in (3) was derived earlier by Matula [5], and can be found by the following procedure:

(1) for $n = |V(G)|$, let v_n be chosen to have minimum degree in G;

(2) for $i = n - 1, n-2, \ldots, 2, 1$, let v_i be chosen to have minimum degree in $< V(G) - \{v_n, v_{n-1}, \ldots, v_{i+1}\} >$. For any vertex ordering $v_1, \ldots, v_n$ determined in this manner, we must have

$$(4)\quad \deg_{<v_1, v_2, \ldots, v_i>}(v_i) = \min_{1 \leq j \leq i} \deg_{<v_1, v_2, \ldots, v_i>}(v_j),$$

$$1 \leq i \leq n,$$

so that such an ordering will be termed a smallest-last (SL) vertex ordering. The fact that any smallest-last vertex ordering minimizes k in (3) over the n! possible orderings is shown by Matula [6].

Note that the determination of a smallest-last vertex ordering has a feature of recursiveness not shared by the largest-first ordering procedure. The degrees of vertices computed in determining a smallest-last ordering are over subgraphs, whereas determination of the largest-first ordering utilizes only the degrees of vertices in the whole graph. Thus, the orderings are not necessarily equivalent.

The procedure of determining a smallest-last ordering of the vertices and then determining the corresponding sequential coloring will be termed the smallest-last coloring algorithm (SL algorithm). It is evident from the construction that the SL algorithm will always determine a coloring requiring no more than $1 + \max_H \min_{v \in V(H)} \{\deg_H(v) \mid H \text{ is a subgraph of } G\}$ colors.

(...) The SL algorithm does not always effect a $\chi(G)$ coloring of G. (...) In attempting to improve the sequential coloring procedure previously described note that when the vertex v_i is adjoined to the (k-1) colored subgraph$<v_1,v_2,\ldots,v_{i-1}>$, a kth color is needed only if v_i is adjacent to vertices with colors 1,2,...,k-1. Now if a complete graph on k-1 vertices exists among the neighbors of v_i, then $\chi(<v_1,v_2,\ldots,v_i>) = k$ and the new color is necessary. Otherwise, it may be possible to change the colors on some neighbors of v_i so as to preserve the (k-1) coloring of $<v_1,v_2,\ldots,v_{i-1}>$ while leaving at most k-2 colors on neighbors of v_i, thus freeing a color for v_i.

Given a graph G with a k coloring in the colors 1,2,...,k, let A_i be the set of vertices of G colored i. For $i \neq j$, the i,j bichromatic subgraph of G is the subgraph $<A_i \cup A_j>$, and a component of $<A_i \cup A_j>$ is an i,j component. If the distinct vertex colors i and j are interchanged on an i,j component of the k colored graph G, then another k coloring of G is obtained. This procedure is termed an $i \leftrightarrow j$ interchange on the k colored graph G. A bichromatic interchange on the k colored graph G is an $i \leftrightarrow j$ interchange on G for some $i \neq j$.

A search for bichromatic interchanges at critical points in the coloring process is introduced in the following coloring algorithm.

For an ordering $v_1,v_2,\ldots,v_n$ of the vertices V of G, a sequential-with-interchange coloring of G corresponding to this ordering is a k coloring of G utilizing each of the colors 1,2,...,k determined recursively as follows:

(1) v_1 is assigned color 1, thus 1 coloring $<v_1>$;

(2) if $<v_1,v_2,\ldots,v_{i-1}>$ has been j colored using each of the colors 1,2,...,j, and if m is the minimum positive integer not ocurring on vertices of $<v_1,v_2,\ldots,v_{i-1}>$ adjacent to v_i in G, then

(a) for $m \leq j$, we assign each vertex of $<v_1,\ldots,v_{i-1}>$ the same color in $<v_1,\ldots,v_i>$, and v_i is assigned color m, thus j coloring $< v_1,\ldots,v_i >$;

(b) for m = j+1, let $k \subset \{1,2,\ldots,j\}$ be the set of color values such that $\alpha \in K$ implies exactly one vertex adjacent to v_i in $< v_1,\ldots,v_i >$ has color α in $< v_1,\ldots,v_{i-1} >$. It for

some $\alpha,\beta \in K$, $\alpha \neq \beta$ an α,β component of $< v_1,\ldots,v_{i-1} >$ has only one vertex adjacent to v_i in $< v_1,\ldots,v_i >$, then perform one α,β interchange on one such α,β component of $< v_1,\ldots,v_{i-1} >$. Now color the vertices $v_1,\ldots,v_{i-1}$ the same as in this new coloring of $< v_1,\ldots v_{i-1} >$, and v_i with the available color, either α or β and a j coloring of $< v_1,\ldots,v_i >$ is obtained; otherwise if no such interchange is possible, color $v_1,\ldots v_{i-1}$, the same as in $< v_1,\ldots,v_{i-1} >$, and color v_i with color j+1, thus (j+1) coloring $< v_1,v_2,\ldots,v_i >$.

The largest-first-with-interchange coloring algorithm (LFI algorithm) will refer to the sequential-with-interchange coloring algorithm applied to a vertex sequence in largest-first order. The recursive-smallest-vertex-degree-last-with-interchange coloring algorithm (SLI algorithm) will refer to the sequential-with-interchange coloring algorithm applied to a vertex sequence in smallest-last order. It should be noted that both of these algorithms depend on the particular LF or SL ordering used and on the particular bichromatic interchange made in Step 2b when more than one suitable interchange is available.

References.

5. Matula, D.W., A min-max theorem for graphs with application to graph coloring, SIAM Rev. 10, 481-482(1968).

6. Matula, D.W., k-components, clusters and slicing in graphs, SIAM J. Appl. Math. 22, 1972 (in press).

10. Welsh D.J.A., and Powell, M.B., An upper bound for the chromatic number of a graph and its application to timetabling problems, Comput.J. 10, 85-86(1967).

Texto extraído de:

GRAPH COLORING ALGORITHMS.
D.W.Matula, G.Marble e J.D. Isaason.
Graph Theory and Computing , editor: R.C. Read.
Academic Press, New York, 1972.
Páginas 110-117, 122.

Apêndice B. ALGORITMO C1;
Welsh e Powell.

The algorithm.

Let V(G) denote the vertex set of G. Let T_1 be a subset of V(G) defined recursively as follows:

$$A_1 \in T_1.$$

If $\{ A_{i_k} \}_{k=1}^{m} \subset T_1$, where $i_1 = 1$ and

$$i_1 \leq i_2 \leq \cdots \leq i_m$$

then $A_j \in T_1$ if and only if

$$j \geq i_m$$

and also A_j is not adjacent to any member $\{ A_{i_k} \}_{k=1}^{m}$.

Clearly T_1 is a finite, non-null subset of V(G). Similarly we define T_2 to be a subset of $V(G) - T_1$ constructed recursively by:
If i is the least integer for which $A_i \notin T_1$, then $A_i \in T_2$.
If $\{A_{j_k}\}_{k=1}^{p} \in T_2$ where $j_1 < j_2 < \ldots < j_p$ then $A_\ell \in T_2$ if and only if

$$\ell > j_p$$

and A_ℓ is not linked in G to any of the vertices $\{A_{j_k}\}_{k=1}^{p}$.
Notice that T_2 may be null (when G is completely disconnected). Similarly we define T_3 to be a subset of $V(G) - (T_1 \cup T_2)$ such that: If i is the least integer for which $A_i \notin T_1 \cup T_2$, then $A_i \in T_3$, and then continuing as before.

In this way we define a sequence $\{T_i\}$ of disjoint subsets of V(G) such that $V(G) = \bigcup_{i=1}^{\infty} T_i$ and there exists a finite integer $\alpha(G)$ such that

$$T_r = \phi \quad r > \alpha(G).$$

From the nature of the construction no two vertices in a given set T_i are adjacent in G and hence by assigning to each vertex in T_i the colour C_i a colouring of G is achieved in $\alpha(G)$ colours.

Texto extraído de:

AN UPPER BOUND FOR THE CHROMATIC NUMBER OF A GRAPH AN ITS APPLICATION TO TIMETABLING PROBLEMS.
D.J. Welsh e M.B. Powell.
The Computer Journal, Volume 10, 1967.
Páginas 85-86.

Apêndice C. ALGORITMO C2;
Williams.

(...) One of the earliest descriptions of a graph colouring heuristic was published by Peck and Williams[1].

(...) The basic Peck-Williams heuristic is based upon the premise that if a vertex i is connected to p other vertices, then it will be more difficult to colour than vertex j which is only connected to q vertices (p > q).
This basic assumption leads to the following heuristic procedure for colouring graphs:

(1) Set c to the value 1.

(2) Select the vertex, v, with the largest degree which:
   (a) has not yet been assigned a colour, and
   (b) is not connected to any other vertex which has already been given colour c.

(3) If no vertex can be found in step (2), go to step(5).

(4) Give vertex v the colour c and go to step (2).

(5) If all vertices have been coloured, go to step (7).

(6) Add one to the value of c and go to step (2).

(7) Print out the colours assigned to the vertices.

This procedure has two distinct advantages:
(i) It has intuitive appeal.
(ii) It is simple to implement even on very small computers.

(...) This heuristic will ensure that the first vertex to be assigned a colour will be the vertex of the highest degree. It is obvious that the selection of vertex i for inclusion in colour T may produce a significantly different colouring from the one produced when vertex j has been selected instead.

(...) A great deal of time and effort was put into searching for an heuristic procedure which, while still being relatively simple, would eliminate the random choice of vertices which bedevilled the first procedure. A rather fortuitous bit of reading (more fully explained in Williams[2]) led to the realization that the vector d ($d_i = \Sigma_j a_{ij}$) was the first step

towards the production of the dominant eigenvector of the matrix A by the standard iterative technique

$$d_i^{(n+1)} = \sum_j a_{ij} d_j^{(n)}$$

(d^{∞} being the eigenvector corresponding to the largest eigenvalue of A). If the vector of degrees, d, is used to start the iterative scheme (i.e. d is used as $d^{(1)}$) then it only takes a few iterations before the magnitude of the elements of the vector $d^{(n)}$ cease to change with respect to each other.

If a vector, $d^{(k)}$ ($2 \leq k \leq 10$) is generated and used in the heuristic, in place of the vector of degrees d, then the procedure appears to perform much better.

References.

1. J.E.L. Peck and M.R.Williams, 'Examination Scheduling', Algorithm 286, Comm. ACM, 9, No. 6 (1966).

2. M.R. Williams, 'Colouring the vertices of large graphs', Proc. Calgary (N.A.T.O.) Conf. Graph Theory and its Applications (Ed. R. Guy), Gordon and Breach, Calgary, Alberta, 1969.

Texto extraído de:

HEURISTIC PROCEDURES (IF THEY WORK - LEAVE THEM ALONE).
M.R. Williams.
Software, Practice and Experience, volume 4, 1974.
Páginas 237 - 240

Apêndice D. ALGORITMO P1;
Wood

Colouring by forming similarity matrix.

The problem is analogous to the classification of objects by attributes, where a collection of objects has to be divided into clusters. A technique used in taxonomy is to form a similarity matrix, to indicate those pairs of objects of greatest similarity which should be put into the same group. In the same way a similarity matrix $S = \{s_{ij}\}$ is formed to determine which vertices should be the same colour. The similarity matrix found most effective is defined as (C é a matriz de adjacência)

$$s_{ij} = 0 \text{ if } c_{ij} = 1$$

$$s_{ij} = \sum_{k} (c_{ik} \,\&\, c_{jk}) \text{ if } c_{ij} = 0$$

where & is the logical operation 'and' (Table 1), and k is summed over all vertices other than i and j. In other words, if i and j are not connected, the similarity is the number of other vertices k which are connected to both i and j.

and

	0	1
0	0	0
1	0	1

Table 1

The similarity matrix is scanned to find the greatest similarity. The first colouring group is started by a pair of vertices with the maximum similarity. Each pair of vertices with this similarity is then coloured according to the following algorithm. The similarity level is reduced by one, and again each pair of vertices with this similarity is coloured. The similarity matrix is scanned repeatedly, reducing the similarity level by one each time, until all vertices have been coloured. Vertices of degree less than the number of groups are left uncoloured, since they can always be added to one of the existing groups.

Colouring algorithm.

The following procedure is adopted to colour a pair of vertices i,j, according to whether both, one or neither of the vertices are already coloured.

(a) Both i and j are coloured.
    1. Go to next pair.

(b) One vertex, say i, is in colouring group G, and the other vertex, j, is uncolored.
    1. If the degree of j is less than the number of groups, then j can always be coloured and is ignored; go to next pair.
    2. Try to add j to group G, i.e., if $c_{jk} = 0$ for each vertex k in group G, then add j to G; go to next pair.
    3. Go to next pair if j cannot be added to group G.

(c) Neither i nor j is coloured.
    1. If the degree of both i and j is less than the number of groups, they are ignored.
    2. Find the first group G to which i and j can be added, i.e., $c_{ik} = 0$ and $c_{jk} = 0$ for each vertex k in group G.
    3. If i and j cannot be added to an existing group, they become the first members of a new group.

Implementation.

Since the store requirements for the conflict matrix (matriz de adjacência) for large sets of data are prohibitive, and the matrix consists mostly of zeros, a list is stored for each subject of those subjects with which it clashes. It would also be pointless to store the full similarity matrix since the elements are required in descending order of magnitude, and the majority of them are zero. Each similarity level is therefore stored as a chain list of the pairs of subjects with that similarity, thus avoiding the need to scan the matrix repeatedly.

With large examples, the lists for small similarities, e.g. less than five, would be enormous and of little value, so they are not recorded. Any subject still uncoloured when the similarity is reduced to this level can easily be coloured singly.

Texto extraído de:

A TECHNIQUE FOR COLOURING A GRAPH APPLICABLE TO LARGE SCALE TIMETABLING PROBLEMS.
D.C. Wood.
The Computer Journal, Volume 12, 1969.
Página 318.

Apêndice E. ALGORITMOS B1 e B2;
Lipton e Miller.

Basic results and terminology.

(...) For a graph $G = (V,E)$ with $|V| = n$ we let $T_K^A(n)$ denote the number of steps (or time) needed to k-color G with respect to algorithm A. When A is obvious, we use just $T_k(n)$.

(...) We will describe an algorithm to 5-color any planar graph for which $T_5(n) = O(n\log n)$. Previous algorithms for 5-coloring planar graphs had $T_5(n) \geq \Omega(n^2)$.

Since vertices of degree ≤ 2 cause no trouble in 5-coloring, from here on we consider G to contain only vertices of degree ≥ 3.

Lemma 1. Let G be a planar graph with n vertices and $D_6(G)$ be the number of vertices in G of degree 3,4,5 or 6. Then $D_6(G) > \frac{1}{4} n$.

Proof. Assume $D_6(G) \leq \frac{1}{4} n$. Then the total degree $\geq 7 \times \frac{3}{4} n + 3 \times \frac{1}{4} n$. By Euler's formula [5] it is known that the total degree of any planar n vertex graph is $\leq 6n - 14$. Thus we have a contradiction and the lemma is proved. □

Lemma 2. Given a planar graph G with n vertices, one can in $O(n)$ time find an independent set of at least $\frac{1}{28} n$ vertices, with vertices each of degree d, $3 \leq d \leq 6$.

Proof. By Lemma 1 there must be vertices $v_1,\ldots v_m$, $m > \frac{1}{4} n$, that have degree d with $3 \leq d \leq 6$. Clearly such a set of vertices can be found in $O(n)$ time simply by inspecting each vertex in turn. Now we use a "greedy" algorithm to, in linear time, find an independent set $\{v_{i_1}, v_{i_2}, \ldots, v_{i_k}\}$ with $k \geq \frac{1}{7} m$. The greedy algorithm proceeds as follows:

Let $v_{i_1} = v_1$. If $v_{i_1},\ldots,v_{i_j}$ have been selected, let $v_{i_{j+1}}$ be the first v_r that is adjacent to no vertex already

selected. Now, since the degree of each of these vertices is 6 or less $k \geq \frac{1}{7} m$ and since $m > \frac{1}{4} n$ it follows that $k > \frac{1}{28} n$. Also, it should be clear that the selection can be done in O(n) time.□

The main results.

So far we have shown that an independent set of vertices of size $> \frac{1}{28} n$ in which each vertex has small degree ($d \leq 6$) can be found in O(n) time for any planar graph.

We now turn to the question of coloring the graph.

Lemma 3. (1) $T_7(n) \leq T_7(\lambda n) + O(n)$ for some constant $0 < \lambda < \frac{27}{28}$.

(2) $T_7(n) = O(n)$

Proof. Clearly (2) follows from (1) by a simple induction, so we need only prove (1). Let G = (V,E) be a planar graph with n vertices. By Lemma 2 we can find $\{v_{i_1}, v_{i_2}, \ldots, v_{i_k}\}$ an independent set of vertices of G with $k > \frac{1}{28} n$. Now let H be the graph induced by the set of vertices $V - \{v_{i_1}, v_{i_2}, \ldots, v_{i_k}\}$. That is, H is G with $\{v_{i_1}, v_{i_2}, \ldots v_{i_k}\}$ removed and all edges touching any vertex in $\{v_{i_1}, v_{i_2}, \ldots, v_{i_k}\}$ removed. Let N_{i_j} be the set of vertices of G that are adjacent to v_{i_j}, $j = 1,2,\ldots,k$. We call N_{i_j} the neighborhood set of v_{i_j}. Since for all i_j, the degree of v_{i_j} is ≤ 6, we have $|N_{i_j}| \leq 6$. Also, since $\{v_{i_1}, \ldots, v_{i_k}\}$ is an independent set H contains all vertices in the N_{i_j} sets, $j = 1, 2, \ldots, k$. Now we wish to prove (1). Clearly (1) is true for all planar graphs having 7 or fewer vertices, providing a basis for induction on n. Now assume (1) is true for all planar graphs having 7 or fewer than n vertices. Then, we can 7-color H in time $T_7(\lambda n)$. Note here that for H $0 < \lambda < (1-\frac{27}{28}) = \frac{27}{28}$ since H has at most $\frac{27}{28} n$ vertices. Then we can extend the coloring of H to 7-color G in the obvious way: For each v_{i_j}, $j = 1,2,\ldots,k$ color v_{i_j} any color not used by vertices of N_{i_j}. Since $|N_{i_j}| \leq 6$ this is always possible in a

7-coloring, and since $\{v_{i_j}, \ldots, v_{i_k}\}$ is independent no interaction between colorings occur over the set. Therefore (1) immediately follows and the lemma is proved.□ (...)

Theorem. (1) $T_5(n) \leq T_5(\lambda n) + O(n\log n)$ for some $0 < \lambda < \frac{27}{28}$.

(2) $T_5(n) = O(n\log n)$.

Proof. As in Lemma 3 it is sufficient to prove (1). Let $G=(V,E)$ be a planar graph of n vertices. Let $\{v_{i_1}, \ldots, v_{i_k}\}$, N_{i_j} for $j = 1, \ldots, k$, and H be defined as in proof of Lemma 3. We now proceed to prove (1) in a manner similar to that of Lemma 3. Clearly (1) is true for all graphs of 5 or fewer vertices. Now assume (1) is true for all planar graphs having fewer than n vertices. Then we can 5-color H in time $T_5(\lambda n)$, where as before $0 < \lambda < \frac{27}{28}$. The extension of the 5-coloring from H to G is, however, more complex than the 7-coloring extension in Lemma 3. For any v_{i_j} for which fewer than 5 colors are used to color the nodes of N_{i_j} the extension is immediate. Checking each v_{i_j} for this condition and extending the coloring in this way, when possible, clearly can be done in time $O(k)$. This leaves a subset of $\{v_{i_1}, v_{i_2}, \ldots, v_{i_k}\}$ for which each neighborhood set N_{i_j} required exactly 5 colors. Let this set of vertices be designated by $\{x_1, \ldots, x_m\}$, with neighborhood sets $N_1, \ldots, N_m$. We have 5-colored the graph H', the graph induced by $V - \{x_1, \ldots, x_m\}$ in time $T_5(\lambda n) + O(k)$. All that remains is to extend the coloring to $\{x_1, \ldots, x_m\}$. Since the neighborhood set of each x_i uses 5-colors, the extension must do some changing of colors. The interchange techniques for 5-coloring [4] are called into play. Let x and y be vertices of H'. We say that $x \equiv_{\alpha\beta} y$ where $\alpha, \beta \in \{0,1,2,3,4\}$ (the colors used) provided there is a path of vertices $x = z_1, \ldots, z_\ell = y$ from x to y each of which is colored either α or β. Clearly, this is an equivalence relation. Obviously, since this is a coloring, α and β alternate along the path. The following is a key fact:

(*) $\forall\ 1 \leq i \leq m\ \exists\ r,s,\alpha,\beta$ such that $y_{i_r} \in N_i$, $y_{i_s} \in N_i$, y_{i_s} is colored α, y_{i_s} is colored β, $y_{i_r} \not\equiv_{\alpha\beta} y_{i_s}$,

and neither α nor β is used by any other neighbor of x_i. Fact (*) is proved like the reduction results in Ore[5] for 5-coloring.

Now consider just the vertices of H' that are colored 0 or 1. Let $C_1,\ldots,C_t$ be the connected components formed by these vertices. We next claim, by renaming if necessary, since there are only 10 choices for α,β that (**) holds:

(**) $\forall\ 1 \leq i \leq m_0$, such that y_{i_0} is colored 0, y_{i_1} is colored 1, y_{i_0} and y_{i_1}, neighbors of x_i, $y_{i_0} \neq {}_{01} y_{i_1}$ and no other neighbor of x_i is colored 0 or 1; where $m_0 \geq \varepsilon_0\, m$ for some constant $\varepsilon_0 > 0$. Now (**) essentially states first that y_{i_0} and y_{i_1} are never in the same component, and second that we can find a "batch" of m_0 such vertices from $\{x_i,\ldots, x_m\}$.

Next form a bipartite graph B as follows. The input vertices of B are $\{x_1,\ldots,x_m\}$; the output vertices of B are $\{C_1,\ldots,C_t\}$. There is an edge from x_j to C_k if and only if y_{i_0} or y_{i_1} is in C_k. Clearly this is a bipartite graph and each x_j has degree exactly 2. We now claim that B is planar. This can be seen by using the contraction operations in Ore[5]. Essentially we are using the simple fact here that we can collapse or contract any part of a planar graph to a point and still retain a planar graph.

Now let B' be the planar graph obtained from B by, for each x_j, replacing a path C_i, x_j, C_k by an edge from C_i to C_k and deleting x_j. The vertices of B' are $C_1,\ldots,C_t$ with $t \leq n$. Clearly B' has m edges. Now in linear time we can 7-color B'. If deg(v) = the degree of vertex v, and $V_1,V_2,\ldots,V_7$ is the partition induced on the vertices of B' by the 7-coloring we see that:

$$2m = \sum_{i=1}^{t} \deg(C_i) = \sum_{i=1}^{7} \sum_{C_j \in V_i} \deg(C_j).$$

So that for some V_i we have:

$$\sum_{C_j \in V_i} \deg(C_j) \geq \frac{2m}{7}.$$

Assume for convenience that V_1 is this block. Now consider $C_j \in V_1$ in the bipartite graph B. They satisfy:

(1) no y_{i_0} or y_{i_1} is in a C_j and a C_k with $j \neq k$.

(2) at least $\frac{2m}{7}$ y_{i_0} 's are in some C_j.

The first follows since V_1 is independent in B', the second by the way V_1 was selected.

Now we can interchange the colors in each $C_j \in V_1$. By (1) and (2) this causes at least $\frac{2}{7}$ m x_i's to be surrounded by only 4 colors. Thus we obtain.

$T_5(n) \leq T_5(\lambda n) + O(k) + \Gamma$,

where Γ is the time required to replace the deleted vertices $v_{i_1}, \ldots, v_{i_k}$. Now we have described above a procedure to do this that runs in several stages. At each stage a positive fraction, say , of all the deleted nodes are replaced. Thus there are at most O(log n) such stages. Moreover, the cost per stage is clearly at most the time required to discover certain connectivity questions. Now this can be done in O(n) time and so is bounded by O(nlong). Now since $k \leq n$ the proof of (1) is complete.□

References.

[4] David W. Matula, George Marble and Joel D. Isaacson, Graph coloring algorithms, Graph Theory and Computing, R.C. Read, ed. (Academic Press, New York, 1972) 109-122.

[5] Oystein Ore, The Four Color Problem (Academic Press, 1967).

Texto extraído de:

A BATCHING METHOD FOR COLORING PLANAR GRAPHS.
R.J. Lipton e R. E. Miller.
Information Processing Letters, volume 7, 1978.
Páginas 185-188.

Apêndice F. ALGORITMO I1;
Christofides.

Maximal subgraphs.

The maximal internally stable set of graph $G = (X,\Gamma)$ - where X is the set of vertices and Γ is a relation - is defined (...) as the set $S[G]$ where.

$$S[G] \cap \Gamma(S(G)) = \phi \qquad (1)$$

and there is no set $S'(G) \supset S(G)$ which satisfies (1).

Let us now define a maximal r-subgraph of a graph G as a graph $(S_r[G], \Gamma)$ which is r-chromatic (i.e. it can be coloured with r colours with no two adjacent vertices having the same colour) and there is no $S'_r[G] \supset S_r[G]$ which is r-chromatic. A subgraph having a maximal internally stable set as its vertices can then be called a maximal 1-subgraph, i.e. $S[G] = S_1[G]$.

The chromatic number of G is then obviously given by the smallest value of r so that any of the $S_r[G] = X$.

The following observation can now be made.

Theorem:
If a graph is r-chromatic it can be coloured with r colours colouring first with one colour a maximal internally stable set $S_1[G]$, next colouring with another colour a set $S_1[(X-S_1(G)), \Gamma]$ and so on, until all the vertices are coloured. (...)

Colourings of the type indicated in the theorem will be called optimal independent colourings.

A maximal r-subgraph with the vertex sets $S_r^i[G]$, say, can be obtained from a maximal (r-1)-subgraph according to the recurrence relation:

$$S_r^i[G] = S_{r-1}^j[G] \cup S_1^k[(X-S_{r-1}^j[G], \Gamma)] \qquad (2)$$

where $S_{r-1}^j[G]$ is any one of the family $\Sigma_{r-1}[G]$ of the vertex sets of the maximal (r-1)-subgraphs of G, and $S_1^k[(X - S_{r-1}^j[G], \Gamma)]$ is any one of the vertex sets of the maximal 1-subgraphs (internally

stable sets) of the subgraph formed by the vertices of G not included in the (r-1)-subgraph $S^j_{r-1}[G]$.

To each $S^j_{r-1}[G]$, there correspond a number of such 1-subgraphs and the family of the vertex sets of the maximal r-subgraphs $\Sigma_r[G]$ is obtained by considering every union of any one $S^j_{r-1}[G]$ for $j = 1,\ldots, q_{r-1}$ (where q_{r-1} is the total number of maximal r-1 subgraphs of G) with every maximal 1-subgraph corresponding to the $S^j_{r-1}[G]$. If a set $S^{j1}_r[G] \supset S_r{}^{j2}[G]$, then $S^{j2}_r[G]$ must be removed from the family of set = $\Sigma_r[G]$ before continuing to the next stage. (...)

Description of the algorithm.

The following is a simplified description of an algorithm to find the chromatic number of a graph and the colouring realising this number.

1. Set r = 1, i = 0. Find the vertex sets $S^j_r[G]$, $(j = 1,2,\ldots,q_r)$, of the maximal r-subgraphs of G(say there are q_r such sets). Set j = 1.

2. Find the vertex sets of a maximal 1-subgraph of the graph $G_j = (X - S^j_r[G], \Gamma)$. If one exists go to step 4. If all of them have already been found go to step 3.

3. If $j \neq q_r$ set j = j + 1. If $j = q_r$ set r = r + 1, i = 0 and j = 1. Go to step 2.

4. Set i = i + 1.

Calculate $S^i_{r+1}[G] = S^j_r[G] \cup S_1[(X-S^j_r(G), \Gamma)]$as indicated by equation (2).

5. If $S^i_{r+1}[G] = X$ stop. The number (r+1) is the chromatic number. The subsets that were introduced into the set $S^i_{r+1}[G]$ according to (2) give the actual colouring. (These subsets can be kept separate with markers as they are introduced.) If $S^i_{r+1}[G] \neq X$, go to step 6.

6. If $S^i_{r+1}[G] \subseteq S^k_{r+1}[G]$, for any k = 1,2,..., (i-1); set $S^i_{r+1}[G] = \phi$ and i = i-1. If $S^i_{r+1}[G] \supseteq S^k_{r+1}[G]$set $S^k_{r+1}[G] = S^i_{r+1}[G]$ and i = i-1. If neither, set $q_{r+1} = i$. Go to step 2.

If the algorithm is not stopped when the first $S^i_{r+1}[G] = X$,

it will continue to produce an alternative colouring with r+1 colours, if such a colouring exists. One should note however that the algorithm will not give a complete enumeration of all possible colourings with r+1 colours but will only produce the optimal independent colours. Such colourings may be only a small fraction of the total possible number of colourings using r+1 colours.

Texto extraído de:

AN ALGORITHM FOR THE CHROMATIC NUMBER OF A GRAPH.
N. Christofides.
The Computer Journal, volume 14, 1971.
Páginas 38-39.

Apêndice G. ALGORITMO I2;
Roschke e Furtado.

Branch and bound criteria.

In this section we propose branch and bound criteria to be used in connection with Christofides' algorithm. A tree of "states" is grown in breadth-first order. Each state indicates the possible selection of one MISS [Maximal Internally Stable Set] (except for the initial state), what vertices have still to be colored, and what MISS are candidates for future selection. When the process terminates at a given state, the MISS in a minimal spanning set are found by tracing back to the initial state.

Selection for generating new states from a state s is made by looking in s for an uncolored vertex v that is present in the smallest number t of candidate MISS. Consider the sequence $(m_1^*, m_2^*, \ldots m_t^*)$ of the MISS that cover v, ordered by decreasing cardinality. Since any optimal coloring must include at least one m_i^*, it suffices to branch from s to create t new states, one for each m_i^*, according to the sequence.

However, the branching is discontinued and the process terminates as soon as in a new state there remains only one candidate MISS.

The candidate MISS at s_i are the MISS of the subgraph of G spanned by the uncolored vertices at s_i; however, instead of calling again the MISS finding algorithm as suggested by Christofides' theorem, we can obtain the candidate MISS at s_i from the candidate MISS at s, by the rules:

a. if $m_p^{(s)} \cap m_i^* = \phi$, $m_p^{(s)}$ is a candidate at s_i;

b. if $m_p^{(s)} \cap m_i^* \neq \phi$, store $m'_p = m_p^{(s)} - m_i^*$ in a temporary set.

After examining all $m_p^{(s)}$ we form the set S as the union of the set of candidates (rule a) with the temporary set (rule b). Then every m'_p that is not a proper subset of some MISS in S is

also accepted as candidate.

In order to prevent certain sequences of MISS to be considered more than once, another rule is applied:

c. the m_j^*, for $j \leq i$ are not candidates.

Texto extraído de:

AN ALGORITHM FOR OBTAINING THE CHROMATIC NUMBER AND AN OPTIMAL COLORING OF A GRAPH.
S.I. Roschke e A.L. Furtado.
Information Processing Letters, volume 2, 1973
Página 35.

Apêndice H. ALGORITMO I3;
Wang.

A Depth-First Search Algorithm.

A shortest path in a reduced subgraph tree can be found by using one of the two search methods, i.e., breadth-first search and depth-first search. In the breadth-first search, the algorithm scans through all possible paths in a given level for a terminal node; if a terminal node is encountered, then a shortest path is found and the algorithm stops; otherwise the algorithm continues to scan all nodes in the next level for a terminal node. On the other hand, in a depth-first search algorithm, we use the following rule to select the next node for scanning: select a node which is not yet scanned before and is branched from a node most recently being scanned. The first path from the root to a terminal node scanned by the algorithm serves as an initial guide line when other paths are scanned. Whenever a shorter path is encountered, this shorter path will be used as a guide line when other paths are scanned. The algorithm scans down a path only up to the level which is equal to the lenght of the path used as a guide line. The path, which is last used as a guide line when all paths have been scanned, is one of the shortest paths of a subgraph tree desired. (...)

It is assumed here that the maximal independent sets of a given graph G are computed before the algorithm begins. For example, one may use Bierstone's algorithm for this purpose. The following notation is used in the algorithm:

V the set of vertices of graph G,

M_j the maximal independent set (MIS) of G, where $1 \leq j \leq m$ and m is the total number of MIS's of G,

g the would-be chromatic number of G,

c_j the would-be jth color class of a chromatic coloring of G,

n the current search level of the subgraph tree of G,

T the set of vertices of G not in the current subgraph at

level n,

b_n the MIS of $< V - T >$ at level n in processing,

e_n the number of unprocessed MIS's of $< V - T >$ at level n,

STACK store all unprocessed MIS's of $< V - T >$ at level 1 up to level n.

Step 1. Set $T = \phi$, $n = 0$, $b_0 = \phi$, and $c_i = \{i\}$ for $i = 1,2,\ldots,g$. ($g = |V(G)|$.)

Step 2. If $n \geq g$ then set $T = T - b_n$, go to step 6; otherwise continue to step 3.

Step 3. If $V - T = \phi$ then set $g = n$, $T = T - b_n$, $c_i = b_i$ for $i = 1,2,\ldots,g$ and go to step 6; otherwise continue to step 4.

Step 4. Compute the maximal independent sets of $< V - T >$ from $M_1 - T$, $M_2 - T$,..., $M_m - T$ and let them be denoted as S_1, S_2,..., S_t.

Step 5. Choose $u \in V - T$ such that u is contained in the least number of maximal independent sets of $< V - T >$. Let them be denoted as S_{u_1}, S_{u_2},..., S_{u_r}.
Put S_{u_i} on STACK for $i = 1,2,\ldots,r$.
Set $n = n+1$, $e_n = r$.

Step 6. If $n = 0$ then stop; otherwise continue to step 7.

Step 7. If $e_n = 0$ then set $n = n-1$, $T = T - b_n$ and go to step 6; otherwise continue to step 8.

Step 8. Move the top item from STACK to b_n. Set $e_n = e_n - 1$, $T = T \cup b_n$, and go to step 2.

Texto extraído de:

AN ALGORITHM FOR THE CHROMATIC NUMBER OF A GRAPH .
C.C. Wang.
Journal of the ACM, volume 21, 1974.
Páginas 388-390.

Apêndice I. ALGORITMO I4;
Lawler.

A number of algorithms have been proposed (to solve the chromatic number problem) (...), yet there appears to be no information in the literature concerning nontrivial upper bounds on the complexity of the problem. In this note we show, by very simple arguments, that the problem can be solved by an algorithm with worst-case running time of $O(mn(1+\sqrt[3]{3})^n)$, where m is the numbers of arcs in the graph and n is the number of nodes. (Note: $1 + \sqrt[3]{3} \simeq 2.445$.)

Definitions.

Let G(N,A) be a graph, with node set N and arc set A. A stable subset of G is a subset $S \subseteq N$ such that no two nodes in S are adjacent in G. A k-coloring of G is a partition of N into k stable subsets. If there exists such a partition, G is said to be k-colorable. The chromatic number of G is the least value of k such that G is k-colorable.

Let N' be an arbitrary subset of the nodes of G. The subgraph of G induced on N' has N' as its node set and A' as its arc set, where A' contains all arcs of A, both ends of which are incident to nodes in N'. we let $\chi(N')$ denote the chromatic number of the subgraph induced on N'.

A recursive computation.

A stable subset S is maximal if S is not a proper subset of any other stable subset S'. It is asserted that if a graph is k-colorable, there is a partition of its node set into k stable subsets, where at least one of the stable subsets is maximal. It follows that if N' is nonempty there exists a maximal stable subset S of the subgraph induced on N', such that

$\chi(N') = 1 + \chi(N' - S)$.

There are a finite number of maximal stable subsets S of the induced subgraph. By minimizing over them we obtain

$$\chi(N') = 1 + \min_{S \subseteq N'} \{\chi(N' - S)\}, \quad N' \neq \phi, \quad \chi(\phi) = 0. \qquad (1)$$

These equations, in fact, represent the essential ideas behing various computational procedures which have been proposed for the chromatic number problem.
See especially [3, 10].

Complexity estimate.

We shall estimate the running time required to solve eqs. (1) for all $N' \subseteq N$, in the worst case. Suppose, for fixed N', we have already found $\chi(N'')$, for all proper subsets $N'' \subset N'$. The time required to compute $\chi(N')$ is then proportional to the number of maximal stable subsets of the subgraph induced on N', plus the time required to generate them. Let $|N'| = r$. We make use of two results:

1. The number of maximal stable subsets of a graph on r nodes does not exceed $3^{r/3}$,
2. There exists an algorithm for generating all maximal stable subsets of an r-node graph in time $O(mrK)$, where K is the number of maximal stable subsets [9]. (...).

It follows that the time required to compute $\chi(N')$ is bounded by a function of $O(mr3^{r/3})$. Summing over all $N' \subseteq N$ and invoking the binomial theorem, we find that the overall running time required to solve eqs. (1) is bounded by a function of order

$$\sum_{r=0}^{n} \binom{n}{r} mr3^{r/3} < mn \sum_{r=0}^{n} \binom{n}{r} 3^{r/3} = mn \left(1 + \sqrt[3]{3}\right)^n.$$

This yields the desired result.

References.

[3] N. Christofides, An algorithm for the Chromatic Number of a Graph, Computer J., 14 (1971) 38-39

[9] S. Tsukiyama, M. Ide, H. Arujoshi and H. Ozaki, A New Algorithm for Generating all the Maximal Independent Sets, unpublished manuscript.

[10] C.C. Wang, An Algorithm for the Chromatic Number of a Graph, J. ACM 21 (1974) 385-391.

Texto extraído de:

A NOTE ON THE COMPLEXITY OF THE CHROMATIC NUMBER.
E.L. Lawler.
Information Processing Letters, volume 5, 1976.
Páginas 66-67.

Apêndice J. ALGORITMO El;
Brown.

Two algorithms to find the chromatic number q of an arbitrary graph will be developed. If s represents a solution, the problem may be reformulated. For variable set $X = \{x_1, x_2, \ldots, x_n\}$ and attribute set $A = \{c_1, c_2, \ldots, c_m\}$, minimize f(s), the number of attributes used in solutions, subject to the undirected graph $G = (X, \Gamma)$ such that x and Γx cannot be assigned the same attribute c_i. All the attributes are indistinguishable. In partial enumeration, large blocks of possible solutions are eliminated by screening techniques which restrict the attributes that can be used to produce new partial solutions from partial solution p. Initially, all attributes may be assigned to the chosen variable x_k. Let U_k represent the attributes that can be assigned to x_k after the attributes have been screened.

The screening process consists of three independent parts. The first part of the screening process is due to all the attributes being indistinguishable (...). The second part is due to the graph and prohibits the chosen variable from being assigned any attribute that has been assigned to any variable in the partial solution that is connected to the chosen variable. The third part is due to the objective function, so that when a solution is found using j attributes, no more than j-1 attributes may be used in the following enumeration. These three parts produce a set of attributes U_k that can be assigned to the chosen variable x_k.

Basic Algorithm.

The enumeration technique that will be used is backtrack programming. Before the algorithm is applied, the variables are ordered such that variable x_k is connected to more of the variables $x_1, x_2, \ldots, x_{k-1}$ than any of the variables $x_{k+1}, x_{k+2}, \ldots, x_n$. Ties are broken by choosing the variable with the most edges. This preordering specifies which variable is to be chosen to augment the partial solution.

Figure 1 is a flow diagram of the basic algorithm.

The index k refers to the variable being considered, and U_k is the set of unused attributes for variable x_k. The number of attributes used in the best solution found so far is q, and ℓ is the number of attributes used in the current partial solution. L_k is ℓ for every variable x_k.

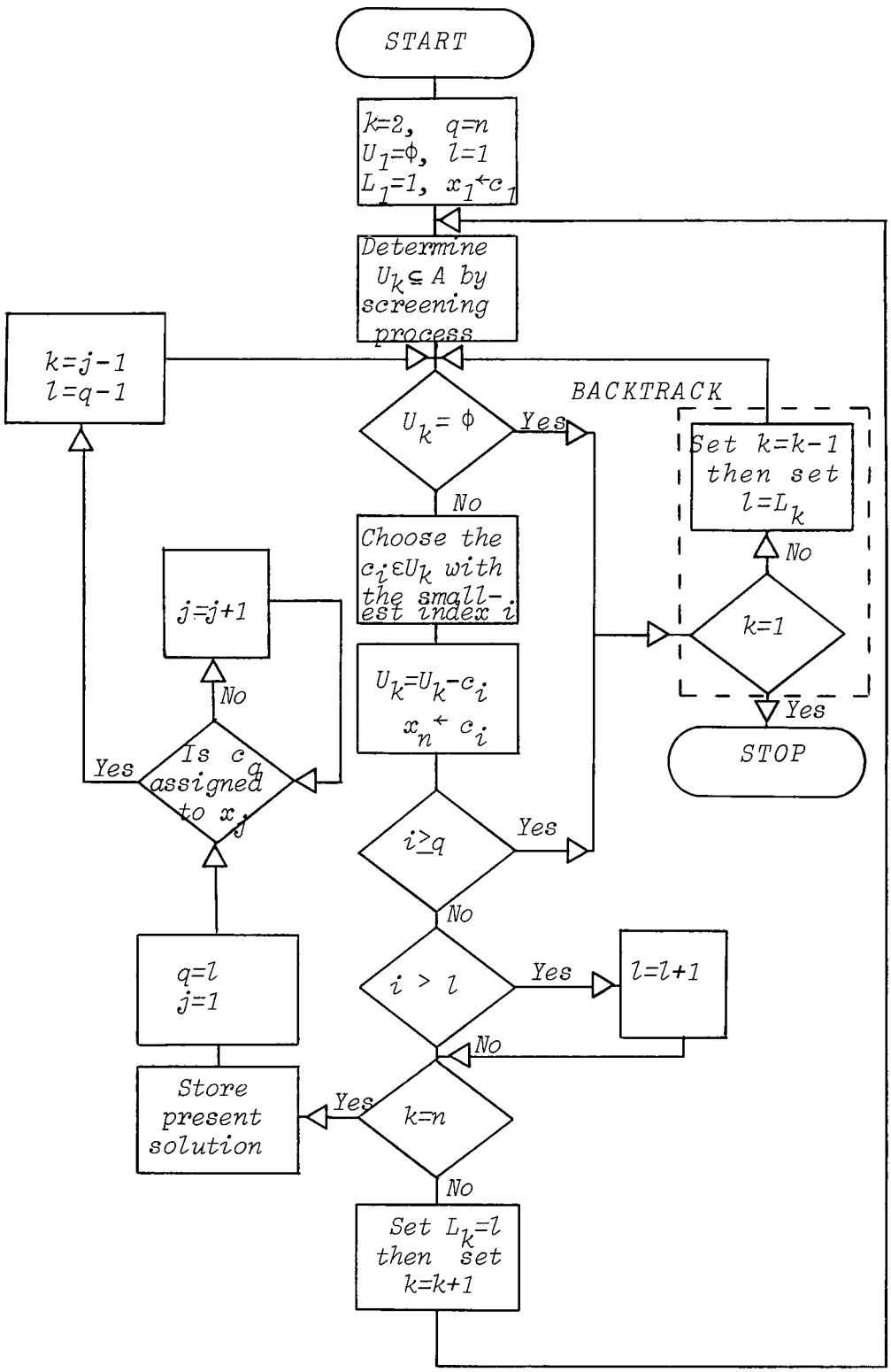

FIGURE 1

Look - Ahead Algorithm.

A natural question to ask at this juncture is whether the basic algorithm can be improved. One strategy that might yield improvement is to try and eliminate more attributes during the screening process and thus reduce the number of backtracks required. Of course, this would increase the time required for each iteration. The unanswered question then is whether the time saved by decreasing the number of backtracks will be significantly greater than the time spent per iteration.

The following procedure is an attempt to reduce the number of backtracks required by looking ahead and identifying the branches of the tree which cannot possibly yield a solution and, alternatively, identifying those branches which appear most promising. Thus, this procedure is called the look-ahead algorithm. During the screening process when the subset $U_k \subseteq A$ of atributes that may be assigned to x_k is being determined, the algorithm looks ahead and determines whether a candidate c_i for U_k would, at some later time, cause an increase in the number of attributes needed. The information gained by looking ahead is also used to determine which attribute $c_i \in U_k$ to assign to x_k.

More specifically, a record is kept for each x_k indicating the number of times that a variable x_i, where $i < k$ and x_i is connected to x_k, is assigned a particular attribute c_j. This record tells what attributes are available for U_k when the process iterates to x_k. For each of these candidate attributes c_j, the process is to find what effect assigning c_j to x_k would have on each x_ℓ, where $\ell > k$ and x_ℓ is connected to x_k. Two statistics are gathered for each c_j: the number of x_ℓ's which in the future could possibly be assigned c_j unless c_j is assigned to x_k (hereafter called the number of preventions), and wether this assignment would increase the current upper bound in the chromatic number (hereafter called an increase of the chromatic number bound). If the assignment of an attribute c_j would cause an increase of the chromatic number bound to equal the current upper limit on the chromatic number (the upper limit is equal to the chromatic number of the last solution found), this attribute c_j is not included in U_k. The attributes $c_j \in U_k$ are ordered first,

by whether they would increase the chromatic number bound, and, secondly, by the number of preventions. The first attribute c_j in this ordering is then assigned to x_k.

Texto extraído de:

CHROMATIC SCHEDULING AND THE CHROMATIC NUMBER PROBLEM.
J.R. Brown.
Management Science, volume 19, 1972.
Páginas 458-460.

Apêndice K. ALGORITMO E2;
Nijenhuis e Wilf.

(A) General (BACKTR)

(...) The method we have adopted involves the following principles (...):

(1) The complete calculation is carried out by three programs: MAIN, BACKTR, CANDTE, of which BACKTR is universal (and appears below) and the other two are prepared by the user.

(2) Communication between the programs is as shown in Fig. 23.1. Note that BACKTR and CANDTE do not speak to each other directly.

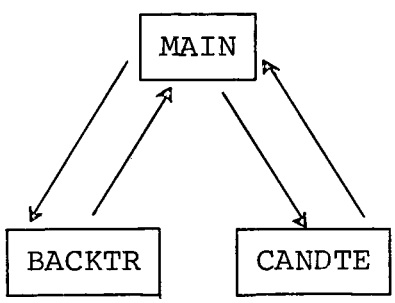

Figure 23.1

(3) MAIN receives input data from the "outside world", and asks BACKTR to inaugurate the search for complete vectors by calling BACKTR with INDEX = 0.

(4) When BACKTR needs a list of all candidates for the Kth component of the output vector, having already found A(1),..., A(K-1), it asks for this list by RETURN-ing to MAIN with INDEX = 2.

(5) MAIN responds to this request by a call to CANDTE, telling CANDTE the value of K, the predecessors A(1),...,A(K-1), and whatever auxiliary arrays are needed for the construction.

(6) CANDTE finds the list of candidates and places them at the end of a STACK, i.e., a linear array containing all candidates for all positions up to the Kth, along with a count of the candidates, which becomes the last word in the stack.

(7) MAIN tells BACKTR this information, and the search continues. When K = L, so that the search is successfully completed, BACKTR returns to MAIN with INDEX = 1. If there are no more vectors of the type sought, the search terminates with a return to MAIN with INDEX = 3.

The above description is a general one. Specific recipes for writing the two routines MAIN and CANDTE will now be given.

The structure of MAIN is as follows:

```
        DIMENSION  A(100),  STACK(1000),...
        Obtain input data
        INDEX = 0
1       CALL BACKTR (L,A, INDEX, K, M, STACK, NSTK)
        GO TO  (10,20,30), INDEX
10      Process output vector  A(1),..., A(L) but do not
        change it!
        GO TO 1
20      CALL CANDTE (A, K, M, STACK,...)
        GO TO 1
30      ...
```

The variables and arrays mentioned play very precise roles:

L is the desired length of a complete output vector.

A is the output vector.

INDEX is explained above.

K is the length of a partially constructed vertex: A call to CANDTE is a request for position A(K); K is set by BACKTR.

M is the location of the last item on the stack; it is changed by both BACKTR and CANDTE.

STACK is a linear array, of maximal length NSTK, whose appearance at a typical intermediate stage in the calculation is shown in Fig. 23.2.

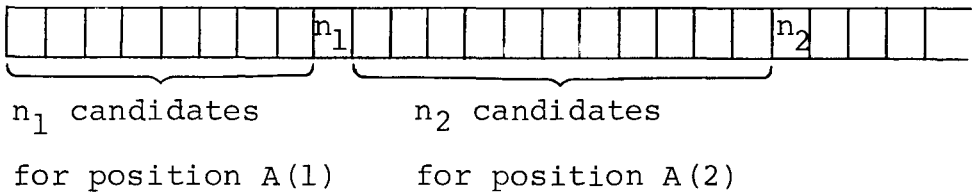

Figure 23.2

More precisely, let the lists of candidates for A(1),...,A(K-1), A(K) be stored in STACK, each list followed by its length. Let NC = STACK (M) be the last item on STACK. Then the items STACK(M-1),..., STACK(M-NC) are the candidates for A(K), given the current values of A(1),...,A(K-1). When one candidate is needed, NC = STACK(M) is examined; if it is zero, we set M ← M-1, K ← K-1, and repeat. Otherwise, we set M ← M-1, A(K) ← STACK(M), STACK(M) ← NC-1. If K = L, we return A. Otherwise we set K ← K+1 and ask CANDTE to place the candidates for A(K) in locations M+1,...,M+Q of STACK, to enter Q into STACK(M+Q+1), and to set M ← M+Q+1. Then BACKTR takes over again.

From this discussion, the precision mission of CANDTE emerges, which we state as follows: Given K, A(1),...,A(K-1),M. Find all candidates for A(K), insert them in locations M+1,...,M+Q of STACK, insert Q into STACK(M+Q+1), set M ← M+Q+1, and return to MAIN. (...)

```
      SUBROUTINE BACKTR(L,A,INDEX,K,M,STACK,NSTK)
      IMPLICIT INTEGER (A-Z)
      DIMENSION A(L), STACK(NSTK)
10    IF (INDEX.NE.0) GO TO 50
20    K = 1
      M = 0
30    INDEX = 2
      RETURN
50    NC = STACK(M)
      M = M-1
60    IF(NC.NE.0) GO TO 100
70    K = K-1
```

```
80        IF(K.NE.0) GO TO 50
90        INDEX = 3
          RETURN
100       A(K) = STACK(M)
          STACK(M) = NC-1
110       IF(K.NE.L) GO TO 120
          INDEX = 1
          RETURN
120       K = K+1
          GO TO 30
          END
```

(B) Coloring the Vertices of a graph (COLVRT)

As our first application of backtracking, let G be a graph of n vertices, and let λ be a given positive integer. A proper coloring of the vertices of G in λ colors is an assignment of a color a_i $(1 \leq a_i \leq \lambda)$ to each vertex i = i,n in such a way that for each edge e of G, the two endpoints of e have different colors.

The vector $(a_1, a_2, \ldots, a_n)$ will be the output of our backtrack program, and we will prepare, in this section, the subroutine CANDTE, in this case called COLVRT, which will cause all possible proper colorings of G in λ colors to be delivered sequentially.

We observed in the discussion of Section (A) that the key question for the user of BACKTR is the determination of all candidates for position K of the output vector if a partially constructed vector (A(1),...,A(K-1)) is given.

In the present case, our answer is as follows: If K = 1, A(1) = 1 is the only candidate (for normalization). If K > 1, the list of candidates is the set of those integers J such that

(1) $1 \leq J \leq \lambda$

and

(2) there is no $I \leq K - 1$, such that A(I) = J and vertex I is connected to vertex K in the graph G.

Suppose that the graph G is specified by means of its vertex-adjacent matrix in LOGICAL form, i.e.,

ADJ(I,J) = $\begin{cases} \text{.TRUE.} & \text{if } I < J, \text{ and vertex I connected to J} \\ \text{.FALSE.} & \text{if } I < J, \text{ otherwise } (1 \leq I, J \leq N) \end{cases}$

Then the set of candidates for position K, if K > 1, is precisely

$\{1,2,\ldots,\lambda\} - \{A(I) \mid I \leq K-1 \text{ and ADJ}(I,K) = \text{.TRUE.}\}$

The actual program adheres exactly to the format described in Section(A). (...)

```
      SUBROUTINE COLVRT(N,A,K,M,STACK,NSTK,LAMBDA,ADJ,COL)
      IMPLICIT INTEGER(A-Z)
      LOGICAL ADJ,COL
      DIMENSION A(N),STACK(NSTK),ADJ(N,N), COL(N)
      IF (K.GT.1)  GO TO 10
      STACK(1) = 1
      STACK(2) = 1
      M = 2
      RETURN
10    K1 = K-1
      DO 20  I = 1,LAMBDA
20    COL(I) = .TRUE.
      DO 30  I = 1,K1
30    IF(ADJ(I,K)) COL(A(I)) = .FALSE.
      M1 = M
      DO 40  I = 1,LAMBDA
      IF (.NOT.COL(I)) GO TO 40
      M1 = M1 + 1
      STACK(M1) = I
40    CONTINUE
      STACK(M1+1) = M1-M
      M = M1+1
      RETURN
      END
```

Texto extraído de:

THE BACKTRACK METHOD (BACKTR).
A. Nijenhuis e H.S. Wilf.
Combinatorial Algorithms.
Academic Press, New York, 1975.
Páginas 175-183

Apêndice L. ALGORITMOS R1 e R2;
Corneil e Graham.

The Zykov algorithm.

(...) Our algorithm is a modification and extension of the Zykov algorithm. Before describing Zykov's method, we introduce the concept of reduction of a graph.

DEFINITION. Let $G(V,E)$ be a graph with nonadjacent vertices x and y. The reduced graphs of G are G'_{xy} and G''_{xy}, where

(i) $G'_{xy} = G'_{xy}(V',E')$; $V' = V$; $E' = E + (x,y)$

(i.e., G'_{xy} is obtained from G by adding the edge (x,y)).

(ii) $G''_{xy} = G''_{xy}(V'',E'')$; $V'' = V - y$;

$(i,j) \in E''$ iff $(i,j) \in E$ for $i,j \neq x$,

$(i,x) \in E''$ iff $(i,x) \in E$ or $i,y \in E$,

(i.e., G''_{xy} is obtained from G by coalescing (identifying the vertices x and y). (...)

Obviously any graph which is not complete can be reduced; a complete graph is irreducible, and the chromatic number of the irreducible graph K_α is α. If either G'_{xy} or G''_{xy} is reducible, the process can be continued until all graphs obtained are irreducible; at this stage G is said to be completely reduced, and (1) $R(G) = \{G_1, G_2, \ldots G_s\}$ is the set of irreducible graphs thus obtained. Also let (2) $R'(G) = \{G'_1, G'_2, \ldots G'_s\}$ denote a set of graphs produced at some intermediate stage of the reduction process. For example, $R'(G) = \{G'_{xy}, G''_{xy}\}$ after just one reduction step.

Theorem 1 below is the corollary of Zikov's theorem [13] on which the Zykov algorithm is based.

THEOREM 1. If x and y are nonadjacent vertices in G, then $\chi(G) = \min[\chi(G'xy, G''xy)]$.

Proof. For a proof of this theorem, see [3].

Theorem 1 can be applied recursively, so when G is completely reduced to R(G) (cf.(1)), we have

(3) $\chi(G) = \min [|G_1|, |G_2|, \ldots, |G_s|]$.

Similarly when G is partially reduced to R'(G) (cf.(2)), we have

(4) $(G) = \min [\chi(G'_1), \chi(G'_2), \ldots, \chi(G'_s)]$.

The improved algorithm.

We will now show how the number of reduction steps required by the Zykov algorithm may be reduced by using a branch and bound approach [7]. In our case the branching is the reduction of G to G'_{xy} and G''_{xy}. We now show how the branching is bounded.(...) In general whenever α has been established as an upper bound for $\chi(G)$, then a graph encountered in the reduction of G which is known to contain an α-clique need not be reduced further. This pruning of the Zykov tree is the essence of bounding and is a major step in the development of our algorithm.

The basic bounding strategy, once α(an upper bound for $\chi(G)$) has been obtained, is now obvious. It would, however, be poor practice to determine whether a reducible graph H contains an α-clique since the clique finding problem is in the Cook-Karp class mentioned in the Introduction. Instead the strategy is to find an α-cluster in H, where an α-cluster is a set of vertices which has a high density of edges; let H_α denote the subgraph of H determined by this α-cluster. Next H is reduced to H' and H" such that H' is formed by adding an edge to H_α. This process is continued with H' until the α-cluster becomes an α-clique, whereupon this branch is terminated. Graphs formed by coalescence (e.g. H") is treated similarly.

If α is the exact value of $\chi(G)$, then the above procedure will always terminate succesfully by building α-cliques. However, if $\alpha > \chi(G)$, then at some stage a graph introduced by coalescence will have only α-1 vertices, implying that $\chi(G) \leq \alpha$-1. Whenever this occurs, the value α-1 is substituted for α and the execution of the algorithm continues uninterrupted (i.e., a decrease of α does not require a repetition of any of the previous steps). It terminates when every branch of the Zykov tree introduced by

reduction has been forced to contain an α-clique. Upon termination, the value of $\chi(G)$ will be exactly α.

We now present a concise recursive definition of the algorithm. Its most important feature is the procedure Reduce (which might be more aptly labeled "Reduce-if-necessary") which has two parameters: H, a graph, and N, the order of H. Note that the algorithm essentially performs a preorder traversal [6, p.316] of a pruned Zykov tree. (...)

Recursive definition of our algorithm.

```
Main Program:  (α is a global variable)
               Find Initial α, an upper bound for χ(G);
               Reduce (G,n);
               Stop (now χ(G) = α).

Procedure Reduce (H,N);
               if α > N then α: = N.
               β: = α; (β is a local variable)
               if H is complete then GO TO EXIT;
               Find Hα, a cluster on α-vertices;
          A:   if Hα is an α-clique then GO TO EXIT;
               choose nonadjacent vertices x and y in Hα;
               Form H'xy and H"xy by reduction of H;
               H: = H'xy;
               Reduce (H"xy, N-1);
               If β = α  then go to A
                         else REDUCE (H,N);
               (the else is necessary in case α was changed
               during the previous step)
       EXIT:   END;
```

References.

[3] B. Graham, An algorithm to determine the chromatic number of a graph, M.Sc. thesis, Tech. Rep., no. 47, Dept. of Computer Sci., Univ. of Toronto, Toronto, Canada, 1972

[7] E.L. Lawler and D.E. Wood, Branch and Bound methods: A survey,

Operation Res., 14(1966), pp. 699-619.

[13] A.A. Zykov, On some properties of linear complexes, Mat.Sb., 24(1949), pp. 163-188; English transl. Amer. Math. Soc. Translation no. 79. 1952.

Texto extraído de:

AN ALGORITHM FOR DETERMINING THE CHROMATIC NUMBER OF A GRAPH.
D.G. Corneil e B.Graham.
SIAM Journal on Computing, volume 2, 1973.
Páginas 311-315,318.

Apêndice M. ALGORITMO S6;
Gavril.

The algorithm runs as follows:

In the kth stage we generate a minimum coloration of the vertex subgraph G^k. (This is sometimes called the section subgraph; its set of vertices is $\{1,2,\ldots,k\}$, and its edges are those edges of G which connect vertices i and j of this set.) Let this minimum coloration be $D_k = (A_1^k, A_2^k, \ldots, A_{m_k}^k)$, where $A_i^k \cap A_j^k = \phi$ if $i \neq j$ and $\bigcup_{i=1}^{m_k} A_i^k = \{1,2,\ldots,k\}$. Clearly, $D_1 = (\{1\})$. We add vertex k+1 to this coloration by the following rule: Find the first A_i^k to which k+1 can be added without destroying its independence. If one is found, add k+1 to it to form D_{k+1}; if none is found, add a new set, $A_{m_{k+1}}^{k+1} = \{k+1\}$ to form D_{k+1} (clearly $m_{k+1} = m_k+1$). D_n is a minimum coloration of G.

Texto extraído de:

ALGORITHMS FOR MINIMUM COLORING, MAXIMUM CLIQUE, MINIMUM COVERING BY CLIQUES AND MAXIMUM INDEPENDENT SET OF A CHORDAL GRAPH.
F. Gavril.
SIAM Journal on Computing, volume 1, 1972
Página 184.