Estudo Computacional de Algoritmos Exatos para Coloração de Grafos

Este exemplar corresponde à redação final da Dissertação devidamente corrigida e defendida por Yuri Abitbol Meneses Frota
`abitbol@lia.ufc.br` e aprovada pela Banca Examinadora.

Fortaleza, 29 de julho de 2003.

Ricardo Cordeiro Corrêa (MCC/UFC)

Dissertação apresentada ao Mestrado em Ciência da Computação, UFC, como requisito parcial para a obtenção do título de Mestre em Ciência da Computação.

Mestrado em Ciência da Computação

Universidade Federal do Ceará

Estudo Computacional de Algoritmos Exatos para Coloração de Grafos

Yuri Abitbol Meneses Frota

abitbol@lia.ufc.br

dezembro de 2001

Banca Examinadora:

- Ricardo Cordeiro Corrêa (MCC/UFC)
- Manoel Bezerra Campêlo Neto (DC/UFC)
- Marcos José Negreiros Gomes (DEC/UECE)
- Valmir Carneiro Barbosa (COPPE/UFRJ)

Agradecimentos

Agradeço primeiro a Deus. Agradeço ao meu pai e minha mãe pelo o que sou hoje, e a Debora por ter estado ao meu lado. Agradeço aos meus amigos por serem meus amigos. Agradeço aos meus professores pela orientação. Agradeço a Funcap pelo apoio durante a elaboração desta dissertação. Agradeço também a PELEJA e aos pelejantes, principalmente aos 4 cavaleiros do apocalipse, onde sem eles o percurso do mestrado teria sido inviável. E claro, agradeço a "meia horinha".

With great power comes great responsibility...
Uncle Ben †

Sumário

1

Introdução

Uma $k-$coloração de um grafo é uma atribuição de uma dentre k cores para cada um dos vértices do grafo de maneira que dois vértices adjacentes não possuam a mesma cor. O problema de coloração de um grafo pode ser definido como o problema de achar o menor número k de cores tal que exista uma k-coloração do grafo e este número mínimo de cores é denominado número cromático.

O problema de coloração de grafos é um dos problemas mais estudados na teoria dos grafos devido à sua relevância prática e teórica. O problema de coloração modela uma coleção de problemas como escalonamento [1], alocação de freqüências [2], alocação de registros [3] [4] e método de elementos finitos [5]. Do ponto de vista teórico, o problema de coloração é $\mathcal{NP}$-difícil [6] [7], logo é pouco provável que ele possa ser resolvido de maneira exata em tempo polinomial. Lund e Yannakakis [8] mostraram que é muito provável que não exista aproximação para o problema de coloração de grafos. Eles demonstraram que se existe uma constante c tal que para cada k existe um algoritmo em tempo polinomial (possivelmente dependente de k) para colorir cada grafo k-colorível com ck cores então $\mathcal{P} = \mathcal{NP}$.

Os métodos existentes hoje para resolver o problema de coloração de maneira exata se mostram impraticáveis, quando tratamos de grandes instâncias para o problema. Para

estas instâncias, geralmente são utilizadas estratégias de enumeração implícita por serem mais flexíveis e eficientes [9]. O primeiro algoritmo deste tipo é creditado a Brown em 1972 [10], que apresentou uma enumeração implícita de cores seguindo uma ordem de vértices. Baseado no modelo de Brown, vários outros algoritmos foram criados na tentativa de melhorar o desempenho destas enumerações [11] [12]. Infelizmente, estes algoritmos tendem a apresentar um grande custo computacional, que inviabiliza suas utilizações na prática. Outros algoritmos foram desenvolvidos com o passar dos anos [9] [13], e demonstraram pouca esperança perante a complexidade da solução deste problema. Recentemente, um esforço concentrado para solucionar o problema de coloração reuniu várias heurísticas sobre um conjunto de instâncias de diferentes tamanhos e características. Este esforço fez parte do segundo desafio de implementação de DIMACS [14], e tornou-se o projeto mais representativo de experimentos sistemáticos sobre o problema de coloração realizado até hoje.

Nesta dissertação nós apresentaremos um estudo computacional de novos algoritmos baseado em dois métodos. O primeiro método, proposto por Mehrotra e Trick [15], apresenta uma formulação baseada em conjuntos independentes maximais. Este método possui a capacidade de gerar limites inferiores muito bons para o problema de coloração, entretanto, a dificuldade de solucionar seu modelo de programação linear, desencoraja sua utilização em problemas de grande porte. O segundo método, apresentado por Herrmann e Hertz [16], propõe uma abordagem para encontrar o número cromático de um grafo através da resolução do problema de coloração para o seu subgrafo crítico, na esperança de que este possua menos vértices que o grafo original. O objetivo dos novos algoritmos é utilizar a força do método proposto por Mehrotra e Trick [15], que consegue alcançar limites inferiores muito próximos do valor do número cromático, com a tentativa de Herrmann e Hertz [16] de construir um grafo crítico para diminuir o tamanho da instância do problema e tornar mais rápida sua solução.

No Capítulo 2 veremos alguns conceitos fundamentais em teoria dos grafos e programação linear que serão necessários durante a leitura do texto. O Capítulo 3 é ded-

icado à apresentação do método genérico de busca adotado para solucionar problemas de otimização. O objetivo principal deste Capítulo é descrever os componentes da busca genérica (conjunto prioridade e operadores) e como eles interagem entre si. No Capítulo 4 veremos algumas formulações do problema de coloração assim como alguns resultados sobre sua complexidade. No Capítulo 5 serão apresentados alguns algoritmos exatos para o problema de coloração enquanto que no Capítulo 6 sugerimos uma modificação para um algoritmo existente e apresentaremos um novo algoritmo que foi desenvolvido durante a elaboração da dissertação. No Capítulo 7 apresentaremos alguns resultados computacionais que indicam que o novo algoritmo criado é competitivo quando comparado a outra técnica existente, utilizando a bateria de testes da DIMACS [14]. O Capítulo 8 contém algumas conclusões e direções para trabalhos futuros.

2

Notações e Definições

Durante a leitura do texto, iremos utilizar uma notação consistente com o padrão geralmente aceito pelos campos de teoria dos grafos, álgebra linear e programação linear. Este capítulo não tem a intenção de ser um texto de referência, e sim de definição das notações e terminologias adotados durante a dissertação. Na Seção 2.1 serão apresentados os conceitos fundamentais em grafos assim como a definição de alguns problemas clássicos da teoria dos grafos. Na Seção 2.2 apresentaremos notações em álgebra linear e algumas manipulações algébricas básicas. Nas Seções 2.3, 2.4, 2.5, 2.6 e 2.7 serão vistos conceitos de problema de otimização e algumas definições de programação linear. Livros de referência em teoria dos grafos [17] [18], álgebra linear [19] e programação linear [20] [21] [22] [23] contêm as definições apresentadas neste capítulo.

2.1. Teoria de grafos

Um *grafo* $G = (V, A)$ é uma dupla ordenada de conjuntos, onde V é o *conjunto de vértices* e A é um conjunto de pares não ordenados de V chamados de *arestas*. Alternativamente usaremos $V(G)$ e $A(G)$ para definir respectivamente o conjunto de vértices e o conjunto de arestas do grafo $G = (V, A)$. Denotaremos por n a cardinalidade do conjunto de vértices V, enquanto a cardinalidade do conjunto de arestas A será denotada por m. A aresta

definida pelos vértices u e v é denotada por uv, e os vértices u e v são chamados de *extremidades* da aresta uv. Todo grafo nesta dissertação é *simples*, ou seja, não possui *arestas múltiplas* ou *laços*, onde duas arestas são múltiplas se elas coincidem em ambas as extremidades e onde um laço é uma aresta cujas extremidades são iguais.

Dois vértices u e v são *adjacentes* em G se $uv \in A$. Denotaremos por $Adj_G(u) = \{v \mid uv \in A\}$ o *conjunto de adjacências* do vértice u em G. Definimos $Anti_G(u) = \{v \mid uv \notin A\}$ como sendo a *anti-vizinhança* do vértice u em G. Em ambos os casos, o índice G pode ser omitido quando o grafo em questão estiver claro pelo contexto. Para um vértice u, o *grau* de u, denotado por $d(u)$, é a cardinalidade de $Adj_G(u)$. O *grau máximo* de G, denotado por $\Delta(G)$, é igual ao $\max_{v \in V}\{d(v)\}$.

O grafo G é dito completo se $d(v) = n-1$ para todo $v \in V$. Um grafo H é denominado *subgrafo* de G, se $V(H) \subseteq V(G)$ e $E(H) \subseteq E(G)$. Um subgrafo $H \subseteq G$ é um *subgrafo induzido* de G se a aresta uv de $A(G)$ está em $A(H)$ sempre que u e v pertencer a $V(H)$. Para um vértice $v \in V$ será denotado por $G - \{v\}$ o subgrafo induzido em G pelo conjunto de vértices $V - \{v\}$. De forma análoga, dado um subgrafo induzido $G' = (V', A')$ de $G = (V, A)$, e dado um vértice $v \in V - V'$, será denotado $G' + \{v\}$ o subgrafo de G induzido por $V' \cup \{v\}$. De maneira geral, para um subgrafo induzido $G' = (V', A')$ de G, será denotado por $G - G'$ o subgrafo induzido em G pelo conjunto de vértices $V - V'$.

Uma *clique* em G é um subconjunto de vértices dois a dois adjacentes. Uma *clique maximal* é uma clique que não está estritamente incluída em nenhuma outra clique de G. A maior clique do grafo G, isto é, a clique de maior cardinalidade, é denominada de *clique máxima*. O dual de uma clique é um *conjunto independente*, o qual consiste em qualquer subconjunto de vértices $S \subseteq V$ tal que não há dois vértices nesse subconjunto adjacentes em G. Um conjunto independente S é dito *máximo* se G não possuir um outro conjunto independente $S^{'}$ com $\mid S^{'} \mid > \mid S \mid$. Um *conjunto independente maximal* é um conjunto independente que não está estritamente incluido em nenhum outro conjunto independente.

Um conjunto independente S de G é chamado *u-extensível*, para qualquer $u \in V - S$,

se $S \cup \{u\}$ é um conjunto independente de G. Além disso, S é uv-extensível, se S é u- ou v-extensível, para qualquer aresta $uv \in A$.

Algumas operações sobre grafos são úteis ao longo do texto. A primeira operação é denominada de *identificação* de dois vértices $u, v \in V$ e é definida como sendo o processo de fusão desses vértices em um novo vértice $\textsc{Ident}(u, v)$, tal que

$$Adj(\textsc{Ident}(u,v)) = Adj(u) \cup Adj(v) \qquad \text{se } uv \notin A$$
$$Adj(\textsc{Ident}(u,v)) = Adj(u) \cup Adj(v) - uv \quad \text{caso contrário}$$

Esta operação dá origem a um grafo $G' = (V', A')$ tal que $V' = (V \cup \textsc{Ident}(u,v)) - \{u, v\}$ e $A' = (A \cup Adj(\textsc{Ident}(u,v))) - (Adj(u) \cup Adj(v))$.

Uma segunda operação, denominada de *coloração* de um grafo, é definida como uma atribuição de uma ou mais cores a cada vértice deste grafo. Uma coloração de um grafo é dita *própria* se quaisquer dois vértices adjacentes recebem cores diferentes. Vemos na Figura 2.1 a ilustração de uma coloração atribuída a um grafo onde os números ao lado de cada vértice indicam os índices das cores a ele atribuídas.

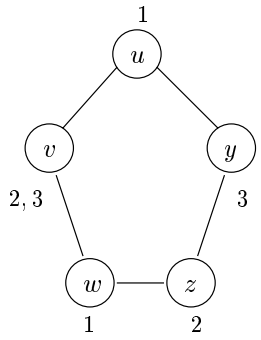


Figura 2.1. *Coloração própria de um grafo*

Um grafo G é *k-colorível* se existir uma coloração própria com k cores para G. Dada uma coloração própria de G, o *grau de saturação* de um vértice $v \in V$ denotado por $d_s(v)$ é definido como o número de cores diferentes atribuídas aos vértices adjacentes a v.

Chamamos de *número cromático* de G, denotado por $\chi(G)$, o menor número de cores necessárias para fazer uma coloração própria de G. O grafo G é dito *crítico* se, todos seus subgrafos induzidos têm o número cromático estritamente menor do que $\chi(G)$.

Uma terceira operação sobre grafos é definida a seguir. Dado uma coloração própria de G definida por uma partição do conjunto de vértices $(V_1, V_2, \ldots, V_k)$ onde cada partição V_i recebe a cor i, denotamos por *permutação(i,j) de cores* para $1 \leq i, j \leq k$ como uma $k'-$coloração de G onde

$$V_i^{'} = V_j;\ \ V_j^{'} = V_i;\ \ V_r^{'} = V_r \text{ para } r = 1, \ldots, k;\ r \neq i, j$$

Vemos que para todo vértice v e $u \in V_i$ temos que $uv \notin A$, logo os vértices de V_i formam um conjunto independente. Então vemos que qualquer coloração de G pode ser vista como uma família de conjuntos independentes de G, onde cada conjunto independente nesta família define uma cor.

2.2. Álgebra linear

Sejam $\mathbb{R}$, $\mathbb{Z}$ e $\mathbb{N}$ os conjuntos dos números reais, inteiros e naturais, respectivamente. Uma *matriz* é uma ordenação de elementos em linhas e colunas. Uma matriz tem *ordem* (m, n) quando ela tiver m linhas e n colunas. Um *vetor* é uma matriz de dimensão $(1, n)$ ou $(m, 1)$. O elemento na i-ésima linha e na j-ésima coluna de uma matriz A é denotado a_{ij}. Para n inteiro, denotaremos por $\mathbb{R}^n$, $\mathbb{Z}^n$ e $\mathbb{N}^n$ os conjuntos de todos os vetores com n componentes reais, inteiras e naturais, respectivamente. Similarmente denotaremos por $\mathbb{R}^{m\times n}$, $\mathbb{Z}^{m\times n}$ e $\mathbb{N}^{m\times n}$ os conjuntos das matrizes reais, inteiras e naturais com m linhas e n colunas.

Seja $A \in \mathbb{R}^{m\times n}$, a $j-$ésima coluna de A é um vetor com m elementos, denotado por a_{*j}.

$$a_{*j} = \begin{pmatrix} a_{1j} \\ \vdots \\ a_{mj} \end{pmatrix}$$

Analogamente, a $i-$ésima linha da matriz A é um vetor de comprimento n, denotado por a_{i*}.

$$a_{i*} = (a_{i1}, \ldots, a_{in})$$

Por convenção vamos utilizar letras maiúsculas quando nos referirmos a matrizes e letras minúsculas quando nos referirmos a vetores.

Uma matriz B é dita *submatriz* da matriz A se ela for obtida através da remoção de linhas ou colunas da matriz original A. Seja $A \in \mathbb{R}^{m\times n}$ e $B \in \mathbb{R}^{p\times q}$ matrizes. O *produto* AB é definido como uma matriz $C \in \mathbb{R}^{m\times q}$ tal que

$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj},$$

para todo $i = 1, \ldots, m$ e todo $j = 1, \ldots, q$ se somente se $n = p$.

Um conjunto de vetores $a_1, a_2, \ldots, a_k$ é denominado de *linearmente independente* se a seguinte condição é satisfeita:

$$\text{se } \sum_{j=1}^{k} \lambda_j a_j = 0 \text{ então } \lambda_j = 0 \text{ para todo } j = 1 \ldots k.$$

Um conjunto de vetores é dito *linearmente dependente* se eles não são linearmente independentes.

Dada uma matriz $A \in \mathbb{R}^{m\times n}$, denominamos de *posto-linha* da matriz A o número máximo de linhas linearmente independentes de A. Analogamente, denominamos de *posto-coluna* da matriz A o número máximo de colunas linearmente independentes de A. Pode ser mostrado que o posto linha de uma matriz é sempre igual ao seu posto coluna, logo, por simplicidade vamos denotar o posto da matriz A por $posto(A)$ que corresponde ao posto linha ou posto coluna de A.

A matriz $I \in \mathbb{R}^{n\times n}$ é denominada *matriz identidade* e é definida como sendo

$$i_{ij} = \begin{cases} 1, & \text{se } i = j \\ 0, & \text{caso contrário.} \end{cases}$$

Dada uma matriz $A \in \mathbb{R}^{m\times n}$, a *transposta* de A, denotada $A^T \in \mathbb{R}^{n\times m}$, é formada atribuindo a j-ésima coluna de A à j-ésima linha de A^T. Assim sendo, tem-se: $a^T_{ij} = a_{ji}$ onde $i = 1, \ldots, n$ e $j = 1, \ldots, m$.

Sejam as matrizes $A \in \mathbb{R}^{n\times n}$ e $B \in \mathbb{R}^{n\times n}$. Se $AB = I$ e $BA = I$, então B é denominada

a matriz *inversa* de A. A matriz inversa, se existir, é única e é denotada por A^{-1}. A matriz que possui inversa é denominada *inversível*.

Proposição 1. *[20] Uma matriz $A \in \mathbb{R}^{n \times n}$ é inversível se somente se suas colunas são linearmente independentes.*

2.3. Problemas de otimização

Um *problema de otimização* Π é um par (X, f), onde X é o conjunto de *soluções viáveis* de Π e $f : X \to \mathbb{R}$ é a sua *função custo*. Alternativamente, a notação $X(\Pi)$ indicará o conjunto de pontos viáveis de Π. O objetivo de um problema de otimização Π é encontrar uma solução viável que minimize ou maximize a função custo. No caso de Π ser um problema de minimização, deseja-se encontrar um ponto $x^* \in X$ tal que

$$f(x^*) \leq f(y), \forall y \in X.$$

Por outro lado, no caso de um problema de maximização, o objetivo é encontrar $x^* \in X$ tal que

$$f(x^*) \geq f(y), \forall y \in X.$$

Em ambos os casos, o ponto x^* é chamado de *ótimo global* ou *solução ótima* do problema Π.

Um exemplo de problema de otimização, é o problema de coloração de vértices que é definido como o problema de determinar o valor do número cromático $\chi(G)$ e achar uma $\chi(G)$-coloração para um grafo G. Em um problema de coloração, o conjunto de pontos viáveis corresponde às colorações próprias de um dado grafo G, enquanto que a função custo, para cada coloração, representa o número de cores distintas nessa coloração.

Um outro exemplo de problema de otimização é o problema de cobertura de vértices por conjuntos independentes maximais. Dado um grafo G, o problema é definido como a busca pelo menor número k de conjuntos independentes maximais $S_1, S_2, \ldots, S_k$ tais que, para todo vértice $v \in V$, temos que existe $i \in \{1, \ldots, k\}$ em $v \in S_i$.

Proposição 2. *O problema de cobertura de vértices por conjuntos independentes maximais é equivalente ao problema de coloração de vértices de um grafo.*

Prova: Seja $V_1, V_2, \ldots, V_k$ a partição em conjuntos independentes (não maximais) de V correspondendo a uma k-coloração própria de G, se existir vértices multicoloridos, considere apenas uma das multiplas cores atribuidas ao vértice. Agora considere o algoritmo polinomial Φ que recebe como entrada uma partição V_i e gera um novo conjunto $V_i^{'} = V_i \cup \{v \in V - V_i \mid V_i \text{ é } v\text{-extensível } \}$. Vemos que $V_i^{'}$ é um conjunto independente maximal. Logo, aplicando o algoritmo Φ para cada partição V_i, $i = 1 \ldots k$, iremos gerar uma família de conjuntos independentes maximais $P' = (V_1^{'}, V_2^{'}, \ldots, V_k^{'})$ que cobrem todos os vértices de G.

Agora suponha uma cobertura de conjuntos independentes maximais de G definida por $C = (s_1, s_2, \ldots, s_k)$. Vemos que os vértices contidos num conjunto s_i, para todo $i = 1 \ldots k$, não são adjacentes entre si, logo todos podem receber uma mesma cor, digamos i. Então, podemos gerar uma coloração para G onde alguns vértices podem estar multicoloridos por pertencerem a mais de um conjunto independente maximal. □

Será denominado de *coloração independente maximal* uma cobertura de V formada por conjuntos independentes maximais.

Proposição 3. $\chi(G)$ = *número de cores de uma coloração independente maximal de* G *de cardinalidade mínima.*

Prova: Considere uma $\chi(G)-$coloração de G qualquer. Por contradição, seja a uma cor e seja s_a um conjunto independente não maximal definido pelos vértices com a cor a. Então, uma nova $\chi(G)-$coloração própria de G pode ser obtida atribuindo a cor a para os vértices em $s - s_a$, onde s é um conjunto independente maximal qualquer que contém s_a. □

2.4. Programação linear

Uma função $f : \mathbb{R}^n \to \mathbb{R}$ é uma *função linear* se somente se $f(\alpha x + \beta y) = \alpha f(x) + \beta f(y)$ para todos $x, y \in \mathbb{R}^n$ e $\alpha, \beta \in \mathbb{R}$. Para qualquer função linear $f(x)$ e qualquer número real b, a equação $f(x) = b$ e as inequações $f(x) \leq b$ e $f(x) \geq b$ são lineares. Um subconjunto $P \subseteq \mathbb{R}^n$ é chamado *poliedro* se

$$P = \{x \in \mathbb{R}^n \mid g_i(x) \leq b_i, i = 1 \ldots m\},$$

onde $g_i : \mathbb{R}^n \to \mathbb{R}$ são funções lineares e $b_i \in \mathbb{R}$. Uma função linear $f : \mathbb{R}^n \to \mathbb{R}$ pode ser escrita na forma $f(x) = c^T x$ onde $c \in \mathbb{R}^n$. Analogamente, um poliedro $P \subseteq \mathbb{R}^n$ é da forma $P = \{x \in \mathbb{R}^n \mid Ax \leq b\}$, onde $A \in \mathbb{R}^{m \times n}$ e $b \in \mathbb{R}^m$. Durante o texto o poliedro P será representado por (A, b).

Seja $P \subseteq \mathbb{R}^n$ um poliedro, $a \in \mathbb{R}^n$ e $\alpha \in \mathbb{R}$. Uma inequação $a^T x \leq \alpha$ é *válida para* P se $P \subseteq \{x \in \mathbb{R}^n \mid a^T x \leq \alpha\}$. Quando o poliedro em questão estiver claro pelo contexto, dizemos simplesmente que a inequação é válida.

Um *problema de programação linear* (PPL) é um problema de otimização (P, f) onde P é um poliedro e f é uma função linear. Um PPL pode sempre ser expresso sob a forma:

$$\begin{array}{lllll} \text{(PPL)} & \text{Minimizar} & c^T x & & \\ & \text{Sujeito a} & Ax & = & b \\ & & x & \geq & 0 \end{array}$$

onde $c \in \mathbb{R}^n$, $A \in \mathbb{R}^{m \times n}$ e $b \in \mathbb{R}^m$. Vamos representar este PPL por (A, b, c) e assumir que $posto(A) = m$. O vetor c é denominado *vetor de coeficientes de custo* e o vetor x é o *vetor de variáveis* a serem determinadas no problema. As equações do tipo $Ax = b$ são chamadas *restrições do problema*. As restrições do tipo $x \geq 0$ são denominadas de *restrições de não negatividade*. A matriz A é denominada *matriz de restrições*.

Uma submatriz $B \in \mathbb{R}^{m \times m}$ de A é denominada de *matriz base* (ou simplesmente *base*) de A se B é inversível. A matriz N composta pelas colunas de A que não pertencem a B será denominada de *matriz não básica*. Sem perda de generalidade, vamos supor

que uma base B é composta pelas primeiras m colunas de A, isto é $B = (a_1, a_2, \ldots a_m)$. Conseqüentemente, a matriz não básica N será composta pelas $n - m$ demais colunas de A, isto é $N = (a_{m+1}, a_{m+2}, \ldots, a_n)$.

Chamamos de *variáveis básicas* as primeiras m variáveis $x_1, x_2 \ldots x_m$ do vetor x correspondentes às colunas de B e que formam o vetor x_B. Denotamos também por x_N o subvetor de *variáveis não básicas* definido pelas $n-m$ variáveis restantes $x_{m+1}, x_{m+2}, \ldots, x_n$, correspondentes às colunas de N.

Dada uma base B de A, dizemos que a solução $x = \begin{bmatrix} x_B \\ x_N \end{bmatrix}$ onde

$$x_B = B^{-1}b$$
$$x_N = 0$$

é uma solução básica do PPL (A, b, c).

Se $x_B \geq 0$, isto é, a solução básica x respeita as restrições de não negatividade, então x é denominada de *solução básica viável*. Duas soluções básicas distintas são ditas *adjacentes* se seus conjuntos de variáveis básicas possuem $m - 1$ variáveis em comum. O teorema a seguir reduz o PPL ao problema de obter a melhor solução básica viável.

Teorema 1. *[20] Assuma que o* PPL (A, b, c) *tem solução ótima. Então, existe uma solução ótima do* PPL (A, b, c) *que é solução básica viável.*

Seja $x = \begin{pmatrix} B^{-1}b \\ 0 \end{pmatrix}$ uma solução básica viável do PPL (A, b, c). O *valor da função objetivo* em x é dado por

$$c^T \begin{pmatrix} B^{-1}b \\ 0 \end{pmatrix} = (c_B^T, c_N^T) \begin{pmatrix} B^{-1}b \\ 0 \end{pmatrix} = c_B^T B^{-1} b,$$

onde c_B e c_N são os coeficentes de custos relativos às variáveis básicas e não básicas, respectivamente. O vetor de *multiplicadores duais* relativos a x é

$$\pi^T = c_B^T B^{-1},$$

onde cada valor π_i $(i = 1, \ldots, m)$ associa-se à $i-$ésima restrição do PPL. Além disso, o *vetor de custos reduzidos* associados à solução x é dado por

$$c_N^T - c_B^T B^{-1} N = c_N^T - \pi^T N.$$

Teorema 2. *[20] Se $c_N^T - c_B^T B^{-1} N \geq 0$ então a solução x é uma solução básica viável ótima para o* PPL (A, b, c).

2.5. Método SIMPLEX e geração de colunas

Um dos primeiros algoritmos criados para solucionar problemas de programação linear foi o SIMPLEX. Este método percorre as soluções básicas viáveis do problema com o objetivo de encontrar sua solução ótima (Teorema 1). A idéia básica do método é apresendada a seguir.

O algoritmo começa com uma solução básica viável e testa sua otimalidade. Se não existir nenhum valor de custo reduzido negativo, então o algoritmo pára, pois a solução ótima terá sido atingida (Teorema 2). Caso contrário, o algoritmo identifica uma solução básica viável adjacente com valor de função objetivo melhor. A base desta nova solução básica viável é obtida através da troca de uma coluna da base anterior por uma coluna não básica. A otimalidade desta nova solução é novamente testada, e o esquema é repetido, até que uma solução ótima básica viável seja encontrada.

Vemos que no PPL (A, b, c) com m restrições e n variáveis, o número de soluções básicas é limitado pelo número de bases. Então, um conjunto de $n - m$ variáveis não básicas podem ser escolhidas de

$$\begin{pmatrix} n \\ m \end{pmatrix} = \frac{n!}{(n-m)!\, m!}$$

maneiras diferentes. Logo vemos que o SIMPLEX não é um método polinomial no pior caso. Apesar dos problemas de programação linear possuírem solução polinomial [24], o

método SIMPLEX é bastante difundido pois, na prática, funciona bem em instâncias de grande porte.

Em alguns problemas específicos, as colunas da matriz A, por algum motivo, não estão todas disponíveis simultaneamente. Isto ocorre quando o número de variáveis do problema é exponencial ao tamanho da entrada e não pode ser abordado diretamente. Geralmente nestes casos é tentado solucionar o problema para um número restrito de variáveis (colunas). Sabemos que as colunas de uma base são suficientes para expressar uma solução básica viável do problema, logo podemos gerar as demais colunas da matriz A equivalentes aos menores custo reduzidos iterativamente até provar-se a otimalidade de uma solução. Este método é conhecido como *geração de colunas*.

2.6. Dualidade

Associado a um problema de programação linear existe um outro problema de programação linear denominado *problema dual*. O problema dual do PPL (A, b, c) (também chamado primal) é definido por:

$$\begin{array}{lllll} \text{(D)} & \text{Maximizar} & b^T\pi & & \\ & \text{Sujeito a} & A^T\pi & \leq & c \\ & & \pi & livre & \end{array}$$

Vemos que o conjunto de variáveis do problema dual é composto pelos multiplicadores duais, que associarão cada variável dual a uma restrição do problema primal.

Uma importante relação entre o problema PPL e seu problema dual D é a seguinte.

Proposição 4. *[20] Se x é uma solução viável para* PPL *e π uma solução viável para seu problema dual* D*, então $c^T x \geq b^T \pi$.*

O critério de otimalidade apresentado pelo Teorema 2 pode ser equivalentemente expresso em termos do problema dual. Uma maneira é apresentada a seguir:

Proposição 5. *[20] Seja* $x = \begin{bmatrix} B^{-1}b \\ 0 \end{bmatrix}$ *uma solução básica viável de* PPL. *Se o vetor de multiplicadores duais* $\pi^T = c_B^T B^{-1}$ *é uma solução viável para* D *então* x *é uma solução básica viável ótima de* PPL.

Pelas Proposições 4 e 5 vemos que se um PPL tem solução ótima, então seu problema dual também terá, e suas soluções ótimas terão o mesmo valor.

2.7. Programação inteira

Um *problema de programação linear inteira* (PPI) é uma problema de programação linear em que as variáveis devem ser números inteiros. Em termos matemáticos temos

$$\begin{array}{lllll} \text{(PPI)} & \text{Minimizar} & c^T x & & \\ & \text{Sujeito a} & Ax & = & b \\ & & x & \geq & 0 \\ & & x & \in & \mathbb{Z}^n \end{array}$$

As restrições do tipo $x \in \mathbb{Z}^n$ são denominadas de *restrições de integralidade*. Um PPI onde todas as variáveis devem ser binárias (0 ou 1) é denominado de *problema de programação linear inteira 0-1*.

O PPL obtido do problema de programação inteira pela retirada de todas as restrições de integralidade é chamado de *relaxação linear* do PPI. Como o conjunto viável do PPI está contido no conjunto viável de sua relaxação linear, obtemos diretamente a seguinte proprosição:

Proposição 6. *Se* x^* *é uma solução ótima do* PPI *e* x *é uma solução ótima de sua relaxação linear, então* $c^T x \leq c^T x^*$.

Problemas de programação inteira, similarmente aos problemas de programação linear, possuem algoritmos para solucioná-los em um número finito de passos. Mas a

semelhança pára neste ponto: O problema de programação linear inteira pertencente à classe $\mathcal{NP}-$Difícil.

3

Busca Genérica

Sabemos que a procura por soluções em um espaço de solução combinatorial grande se constitui em um difícil problema da otimização combinatória. Muitos problemas da literatura podem ser resolvidos com pesquisas combinatoriais. Buscas exaustivas, nas quais geralmente temos que percorrer todo o espaço de solução, são muito caras para problemas de grande porte. Por isso, estudos estão sendo realizados para aperfeiçoar as técnicas de busca.

Este capítulo é dedicado à descrição de um método genérico de busca sugerido por Corrêa, que pode ser usado para descrever qualquer tipo de busca em problemas de otimização combinatória. Na Seção 3.1 dissertaremos sobre métodos de busca para problemas de otimização combinatória. Na Seção 3.2 será apresentada a definição de subproblema de otimização combinatória. A definição dos componentes que formam a busca genérica, conjunto prioridade e operadores, será apresentada nas Seções 3.3 e 3.4 respectivamente. Na Seção 3.5 será apresentado o algoritmo que detalhará a busca genérica. Na Seção 3.6 veremos como obter um método *branch-and-bound* a partir da busca genérica.

3.1. Algoritimos de busca para otimização combinatória

Soluções ótimas para problemas de otimização são muito desejadas em várias situações do mundo moderno. Por esta razão, muitos métodos de soluções exatos eficientes foram desenvolvidos através de pesquisas nesta área. A maioria dos métodos exatos usa propriedades estruturais do espaço de solução para guiar a busca pela solução ótima do problema em um esquema *branch-and-bound*.

O método denominado *branch-and-bound* é um algoritmo de árvore de busca que usa um método de enumeração implícita do espaço de solução para problemas de otimização combinatória. Seu princípio baseia-se em sucessivas decomposições do problema original em problemas menores até ser encontrada uma solução viável para o problema, e de provar sua otimalidade. Esta enumeração é implícita pois alguns subproblemas que não contêm nenhuma solução ótima não serão analisados. *Backtracking*, programação dinâmica e A* podem ser vistos como variações do algoritmo *branch-and-bound* [25] [26].

Nas próximas seções apresentaremos um método de enumeração implícita geral que pode ser configurado para qualquer tipo de busca sobre problemas de otimização combinatória. Este método possui uma grande flexibilidade e se constitui numa ferramenta prática e eficaz na resolução destes problemas.

3.2. Subproblema de otimização combinatória

Dado um problema de otimização $\Pi = (X, f)$, definimos um *subproblema* de Π como sendo um par (X_1, f) tal que $(X_1 \subseteq X)$, ou seja, um subproblema constitui-se em um problema de otimização combinatória formulado sobre um subconjunto do problema original. Vemos que um subproblema é obtido de um dado problema quando adicionamos restrições ao conjunto de soluções viáveis de forma que somente as soluções viáveis em X, que também satisfazem a essas novas restrições, pertençam a X_1.

Como exemplo, considere o problema de coloração de vértices de um grafo. Conforme

ilustrado na Figura 3.1, a adição de mais uma aresta uw ao grafo resultaria na resdução do conjunto de soluções viáveis do problema, pois obrigaria os vértices u e w a terem cores distintas. Notemos que uma coloração própria para o subproblema implica numa coloração própria para o problema original.

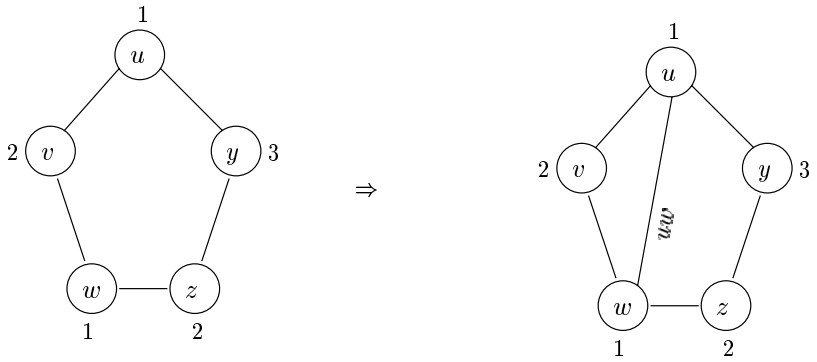


Figura 3.1. *Subproblema de G*

Dado um problema de otimização $\Pi = (X, f)$, definimos um *subproblema estendido* σ de Π como sendo uma quádrupla $\sigma = (p_\sigma, h_\sigma, l_\sigma, u_\sigma)$, onde p_σ é um subproblema de (X, f), $h_\sigma \in \mathbb{R}$ é um valor chamado de *prioridade* do subproblema, cujo objetivo será explicado na Seção 3.5, $l_\sigma \in \mathbb{R}$ é um valor denominado *limite inferior* do subproblema, e é tal que:

- $l_\sigma \leq f(x^*_{p_\sigma})$
- se $\mid X(p_\sigma) \mid = 1$ então $l_\sigma = f(x^*_{p_\sigma})$

e, finalmente, $u_\sigma \in \mathbb{R}$ é um *limite superior* do subproblema, definido como sendo um valor tal que

$$u_\sigma \geq f(x^*_{p_\sigma})$$

sendo $x^*_{p_\sigma}$ uma solução ótima de p_σ e $X(p_\sigma)$ o conjunto de pontos viáveis de p_σ.

Com essas definições, podemos agora descrever o método da busca genérica como um algoritmo de partição que realiza decomposições do problema em subproblemas disjuntos, até que uma solução viável seja encontrada e seja demonstrada a otimalidade do subproblema, ou até que seja provado a inviabilidade daquele subproblema. Sabemos que

percorrer todo o espaço de solução é impraticável, logo devemos tentar construir uma prova de que a solução encontrada é ótima, baseada nas sucessivas partições do espaço de solução, evitando percorrer soluções desnecessárias.

Durante essa busca recursiva no espaço de solução, a variável x_u ($x_u \subseteq X$) é constantemente atualizada com a melhor solução viável encontrada do problema $\Pi = (X, f)$. Na variável U é armazenado o valor desta solução, ou seja, $U = f(x_u)$.

3.3. Conjunto prioridade

O algoritmo da busca genérica contém *conjuntos prioridades* que são responsáveis por armazenar os subproblemas estendidos que foram originados de decomposicões previamente realizadas. Dado um problema (X, f), um conjunto prioridade, denotado por H, é um conjunto que contém subproblemas estendidos de (X, f) e sobre o qual são definidas as seguintes três funções :

SELEÇÃO : Esta função retorna um subproblema estendido σ tal que :

- $p_\sigma = (\emptyset, f)$ se $H = \emptyset$
- $(h_\sigma \leq h_{\sigma'})$, $\forall \sigma' \in H$, se $H \neq \emptyset$

O subproblema estendido σ é extraído do conjunto prioridade no fim da operação ($H = H - \{\sigma\}$). Vemos que o operador de SELEÇÃO irá retornar sempre um subproblema estendido de maior prioridade do conjunto. Esta prioridade governa a ordem em que os subproblemas são selecionados para serem decompostos, e isto influi decisivamente no comportamento da busca.

INCLUSÃO : Esta função tem como entrada um subproblema estendido σ onde este será inserido no conjunto prioridade H se $l_\sigma < U$, caso contrário o subproblema será descartado. Após esta função, o conjunto prioridade será acrescido do subproblema estendido σ ($H = H \cup \{\sigma\}$).

PODA : Esta função tem como entrada um valor de poda $r \in \mathbb{R}$ e irá retirar do conjunto prioridade H todo subproblema estendido σ tal que $l_\sigma \geq r$. Após a poda, o conjunto prioridade passa a ser $H - \{\sigma \in H \mid l_\sigma \geq r\}$.

3.4. Operadores

Os operadores são funções responsáveis por modificar os subproblemas estendidos que estão armazenados nos conjuntos prioridades. O algoritmo da busca genérica contém operadores que são responsáveis, dentre outras coisas, por realizar as decomposições dos subproblemas.

Um *operador* OP é uma função que age sobre um subproblema estendido σ, e retorna um conjunto de subproblemas estendidos $\text{OP}(\sigma)$ tal que:

1. se $\sigma' \in \text{OP}(\sigma)$ então $p_{\sigma'}$ é um subproblema de p_σ
2. $\bigcap_{\sigma' \in \text{OP}(\sigma)} X(p_{\sigma'}) = \emptyset$, ou seja, os subproblemas são disjuntos

Dentre os operadores, existem aqueles responsáveis por encontrar soluções viáveis no espaço de solução X. Toda vez que uma nova solução viável é encontrada por um operador, deve-se realizar a função de *atualização* sobre o par (U, x_u) do problema. Esta função de atualização será descrita em detalhes na Seção 3.5.

Um exemplo de operador é a RAMIFICAÇÃO (ou RAM), o qual realiza a decomposição de um subproblema. Este operador recebe como entrada um subproblema estendido σ, e terá como retorno um conjunto de k subproblemas estendidos $\{\sigma^1, \sigma^2, \ldots, \sigma^{k-1}, \sigma^k\}$ (onde k é a cardinalidade da ramificação) derivados do problema original, tais que

- $\bigcup_{\sigma^k \in \text{RAM}(\sigma)} X(p_{\sigma^k}) = X(p_\sigma)$
- $\forall \sigma^k \in \text{RAM}(\sigma)$, o valor de h_{σ^k} dependerá da política de prioridade da estrutura.
- $l_{\sigma^k} = l_\sigma$, $\forall \sigma^k \in \text{RAM}(\sigma)$

- $u_{\sigma^k} = \infty$ (limite superior indefinido)

Um exemplo para o operador RAMIFICAÇÃO é apresentado a seguir. Dado um problema de coloração de vértices σ, definido em $G = (V, A)$, podemos decompor o problema original em dois subproblemas disjuntos σ' e σ'' da seguinte forma. Sejam $a, b \in V(G)$ e $ab \notin A(G)$. Então σ' e σ'' são definidos em $G_{\sigma'}$ e $G_{\sigma''}$ onde:

- $G_{\sigma'} = (V_{\sigma'}, A_{\sigma'})$, onde $V_{\sigma'} = V - \{a, b\} + \text{IDENT}(a, b)$ e $A_{\sigma'} = A \cup Adj(\text{IDENT}(a, b)) - \{Adj(a) \cup Adj(b)\}$
- $G_{\sigma''} = (V_{\sigma''}, A_{\sigma''})$, onde $V_{\sigma''} = V$ e $A_{\sigma''} = A \cup \{ab\}$

A Figura 3.2 mostra a decomposição do problema inicial em dois subproblemas disjuntos: o primeiro pela inserção de uma aresta entre os vértices $a, b \in V(G)$, o que induz uma coloração distinta entre eles, e o segundo pela identificação de a e b, o que obriga a atribuição de cores idênticas para a e b.

Outro exemplo de operador é o LIMITE_SUPERIOR. Se um subproblema estendido σ, recebido como entrada, for "simples" o suficiente para ser solucionado, o operador encontra a solução ótima x^*_σ de σ, que é também uma solução viável de (X, f). Então o operador retorna $\sigma = nil$ onde nil representa um problema nulo. Caso σ não seja "simples" o suficiente para ser solucionado, o operador retorna $\{\sigma'\}$ tal que:

- $p_\sigma = p_{\sigma'}$
- $u_{\sigma'} \leq u_\sigma$
- O valor de $h_{\sigma'}$ dependerá da política de prioridade da estrutura.
- $l_{\sigma'} = l_\sigma$

Vemos que se uma solução ótima x^* do subproblema σ' for encontrada tal que $f(x^*) < u_\sigma$ ou $u'_\sigma \leq u_\sigma$, então o operador LIMITE_SUPERIOR deve atualizar (U, x_u) através da função de atualização.

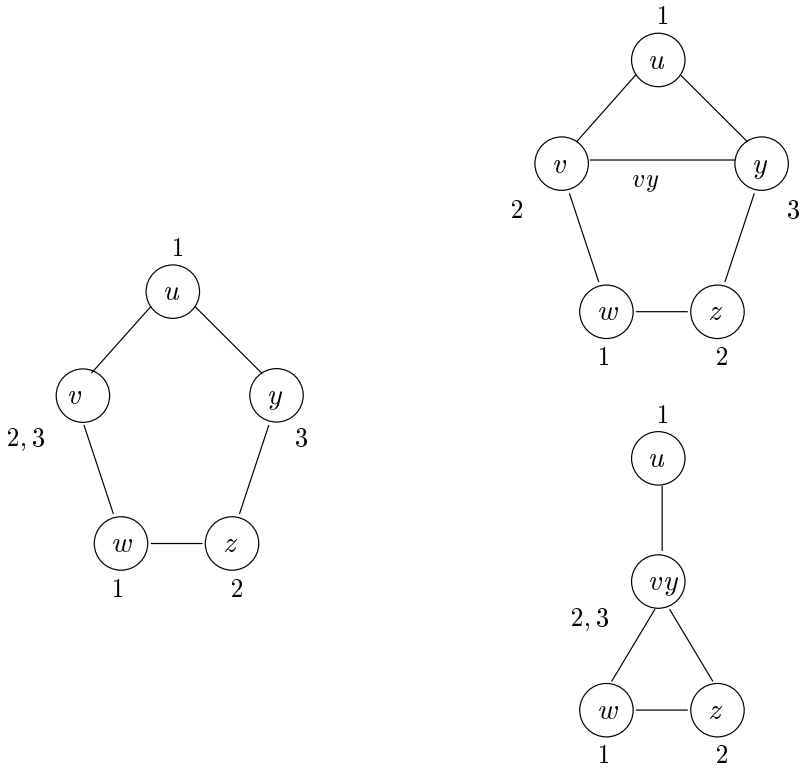


Figura 3.2. *Ramificação de G.*

Um exemplo de limite superior aplicado ao problema de coloração de vértices seria o valor $\Delta(G) + 1$. A construção desta prova repousa na idéia de atribuir uma cor diferente para cada vizinho do vértice $v \in V(G)$, logo, precisariamos de mais uma cor para colorir v.

$$\chi(G) \leq \Delta(G) + 1.$$

Pela Figura 3.3 vemos que o número cromático do grafo é menor do que $\Delta(G) + 1$.

Vamos considerar neste exemplo que um grafo é "simples" de colorir se ele for uma clique, pois o número cromático de uma clique é igual ao número de vértices do grafo. Logo, se um grafo G for uma clique então encontra-se a coloração ótima de G, senão seu limite superior será $\Delta(G) + 1$. O modelo da busca genérica necessita que pelo menos um operador possa encontrar soluções viáveis para o problema. Essa checagem por possíveis soluções pode estar anexada a outros operadores.

Mais um exemplo de operador é o LIMITE_INFERIOR que recebe também como entrada

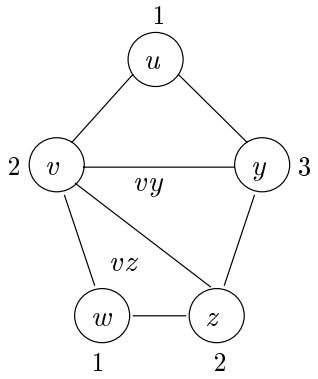


Figura 3.3. $\chi(G) \leq (\Delta(G) + 1)$

um subproblema estendido σ e retorna $\{\sigma'\}$ tal que

- $p_\sigma = p_{\sigma'}$
- $u_{\sigma'} = u_\sigma$
- O valor de $h_{\sigma'}$ dependerá da política de prioridade da estrutura.
- $l_{\sigma'} \geq l_\sigma$

O tamanho da maior clique do grafo ($w(G)$) é um exemplo de limite inferior para o problema de coloração de vértices, pois como está ilustrado na Figura 3.4, o número de cores necessárias para colorir G é, no mínimo, a cardinalidade da maior clique $w(G)$. Assim, o operador de limite inferior receberia como entrada um grafo G e retornaria como limite inferior o valor $w(G)$.

3.5. Método genérico de busca

Depois de termos definido conjunto prioridade e operadores, somos agora capazes de detalhar a interação entre os componentes de um algoritmo de busca genérica. O algoritmo é composto por:

1. Uma família de conjuntos prioridades, denotado por $\mathcal{H}$. Vemos que $\mid \mathcal{H} \mid \geq 1$ é

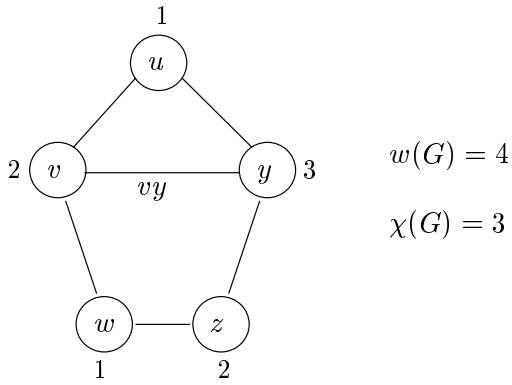


Figura 3.4. $\chi(G) \geq w(G)$

uma condição necessária para o algoritmo pois é preciso pelo menos um conjunto prioridade para percorrer o espaço de solução X do problema tratado.

2. Uma heurística PRIO que irá estabelecer a política de prioridade dos conjuntos prioridades onde

$$h_\sigma = \text{PRIO}(\sigma), \quad \forall \sigma \in H, \; onde \; H \in \mathcal{H}$$

A heurística governa a ordem em que os subproblemas estendidos são selecionados de um conjunto prioridade. Deve-se procurar estabelecer uma ordem de visitação dos subproblemas estendidos de tal forma que os "melhores problemas" (aqueles que possuem alta probabilidade de levar a uma solução ótima) possam ser selecionados primeiro.

3. Um conjunto de operadores, denotado por $\mathcal{O}$. Dentre os operadores do conjunto, é nescessário haver uma função de RAMIFICAÇÃO para que os subproblemas possam ser decompostos em subproblemas mais restritos e possamos percorrer o espaço de solução. É necessária também, uma função que possa encontrar soluções viáveis para o problema (EX: LIMITE_SUPERIOR, LIMITE_INFERIOR). O método pode ter apenas um operador desde que ele englobe os elementos necessários para a busca.

4. Um par de variáveis (U, x_u) que irão guardar a melhor solução corrente. Sempre que uma nova solução viável for encontrada durante o processo de decomposição do problema, a função de atualização será realizada. Se a nova solução for melhor que

U, a variável será atualizada e um processo de PODA será realizado sobre a família dos conjuntos prioridades.

Notemos que, dado um subproblema estendido σ tal que $l_\sigma \geq U$, os problemas $\sigma' \in \text{RAM}(\sigma)$ só podem ter limites inferiores maiores do que l_σ. Logo, nenhuma solução melhor do que x_u poderá ser encontrada a partir de σ. Sempre que uma nova solução viável para o problema for encontrada, o procedimento da atualização, que está descrito no algoritmo da Figura 3.5 deve ser invocado.

```
1.  Algoritmo: ATUALIZAÇÃO
2.  Entrada subproblema σ, valor da solução do subproblema f(x*_σ)
3.
4.    Se f(p_σ) < U Então
5.      U ← f(x*_σ);
6.      x_u ← p_σ;
7.      H_i.PODA(U); ∀H_i ∈ ℋ
```

Figura 3.5. *Algoritmo de atualização*

5. Um grafo orientado $\mathcal{GR}$ de relacionamento (operadores/conjuntos prioridades). Este grafo é responsável pela representação do relacionamento entre operadores e conjuntos prioridades. Os vértices de $\mathcal{GR}$ podem ser descritos como do tipo operadores ou conjunto prioridade, e todo arco $uv \in A(\mathcal{GR})$ tem uma extremidade em um operador e a outra num conjunto prioridade. Dado um arco $uv \in A(\mathcal{GR})$, se u é um vértice do tipo conjunto prioridade e v é um vértice do tipo operador então o conjunto prioridade associado a u é chamado de *conjunto prioridade de entrada* do operador associado a v. Caso u seja um vértice do tipo operador e v seja um vértice do tipo conjunto prioridade então o conjunto prioridade associado a v é denominado de *conjunto prioridade de saída* do operador associado a u.

   O grafo $\mathcal{GR}$ deve ser tal que todo operador possua um conjunto prioridade de entrada e um conjunto prioridade de saída. Além disto, todo conjunto prioridade deve estar relacionado a pelo menos um operador. Em Corrêa [27] são apresentadas algumas

propriedades que $\mathcal{GR}$ deve satisfazer para garantir a convergência do método.

6. Um escalonador denotado por Esc. O escalonador recebe um conjunto de operadores $\mathcal{O}$ como entrada e retorna um operador $Op \in \mathcal{O}$. Algumas condições devem ser observadas para o escalonador, pois nem todo operador pode ser escolhido a todo momento. Por exemplo, operadores que tiverem conjunto prioridade de entrada vazios não podem ser escolhidos.

Vemos que os elementos descritos nessa enumeração permitem a implementação de diversas abordagens de busca pois podemos configurar o modelo de acordo com os objetivos a serem alcançados. O modelo da busca genérica está implementado na forma de biblioteca de funções que podem ser configuradas através da passagem de parâmetros como operadores e escalonadores (Apêndice A). Esta biblioteca de funções, denominada PARADISE, possui uma versão paralela que pode ser utilizada futuramente para explorar o potencial paralelo dos problemas aqui apresentados.

O algoritmo da busca genérica, ilustrado pela Figura 3.6, termina quando todo subproblema estendido presente nos conjuntos prioridades ($\forall \sigma \in H \mid H \in \mathcal{H}$) é solucionado ou é provada a inviabilidade de encontrar a solução ótima através daquele subproblema estendido.

```
1.  Algoritmo: Busca_Genérica
2.  Entrada problema inicial S, melhor solução x_1, valor da melhor solução U_1
3.
4.    H_ini ← {S}; U ← U_1; x_u ← x_1;
5.    Enquanto (H_i ≠ ∅, ∀H_i ∈ ℋ) Faça
6.      Op ← Esc(𝒪)
7.      Exec(Op)
8.    Retorna x_u;
```

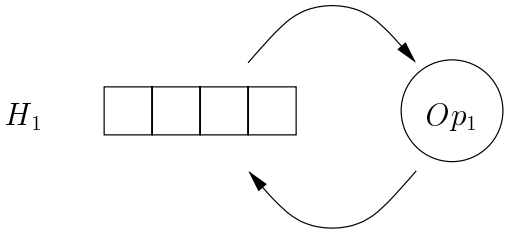
Figura 3.6. *Algoritmo da busca genérica*

3.6. Método branch-and-bound

Um algoritmo de *branch-and-bound* pode ser visto como um processo de enumeração, composto por sucessivas decomposições realizadas por uma função de ramificação. Os problemas gerados que não são simples o suficiente para serem solucionados devem ser novamente inseridos em um conjunto prioridade e decompostos com a finalidade de percorrer o espaço de solução e encontrar uma solução ótima. Entretanto, não é interessante ter que percorrer todo o espaço de solução, pois alguns destes subproblemas estendidos podem ser eliminados da enumeração usando operações de atualização da melhor solução corrente (U, x_u).

Definiremos o método de busca para o *branch-and-bound* por

- $\mathcal{H} = \{H_1\}$
- $h_\sigma = \textsc{Prio}(\sigma), \quad \forall \sigma \in H_1$
- $\mathcal{O} = \{\textsc{Op}_1\}$.
- O grafo de relacionamento do método para o *branch-and-bound* possui apenas um operador ($\textsc{Op}_1$) que retira e insere subproblemas de um único conjunto prioridade (H_1).



- $\textsc{Esc} = \textsc{Op}_1$

Vemos que a família de conjuntos prioridade $\mathcal{H}$ é composto por um único conjunto prioridade H_1. Sobre este conjunto prioridade H_1 atuará um único operador $\textsc{Op}_1$. Para cada subproblema σ selecionado de H_1 pelo operador $\textsc{Op}_1$, a função L_{inf} irá calcular

o limite inferior de σ. Analogamente a função L_{sup} irá calcular o limite superior para este subproblema. A função ATUALIZAÇÃO será executada sempre que uma nova solução para o problema for encontrada. O processo de ramificação do subproblema σ, realizado pela função RAM, irá gerar e inserir os novos subproblemas σ^i (com i menor ou igual a cardinalidade da ramificação) que não puderam ser solucionados no conjunto prioridade H_1 (Figura 3.7). Os algoritmos exatos estudados durante a elaboração desta dissertação seguem o modelo do método *branch-and-bound*.

```
1.   Algoritmo: OP_1
2.
3.     σ ← H_1.SELEÇÃO;
4.     l_σ = L_inf(p_σ);
5.     u_σ = L_sup(p_σ);
6.     h_σ = PRIO(σ);
7.     Se (u_σ é solucionável) Então
8.       ATUALIZAÇÃO(p_σ, f(p_σ));
9.     Senão
10.      Se (u_σ < U) Então
11.        ATUALIZAÇÃO;
12.      {σ^1, σ^2, ... σ^k} ← RAM(σ) onde k = cardinalidade da ramificação;
13.      H_1.INCLUSÃO(σ^i) para i = 1 ... k
```

Figura 3.7. *Operador do método de busca para o branch-and-bound.*

4

Problema de Coloração de Grafos

A maioria dos problemas difíceis de otimização em grafos pode ser formulado como um problema de programação inteira. Como visto no Capítulo 1, o problema de programação inteira é $\mathcal{NP}-$Difícil, o que torna pouco provável resolvê-lo em tempo polinomial. Uma alternativa para os métodos exatos (que são computacionalmente caros) são as heurísticas. O objetivo deste capítulo é descrever noções ligadas ao problema de coloração que serão usadas nos capítulos seguintes. Na Seção 4.1 será formulado o problema de decisão para a coloração de grafos, a seguir serão apresentadas algumas inequações válidas para o problema. Na Seção 4.2.2 apresentaremos uma formulação para o problema de coloração baseada em conjuntos independentes maximais. Na Seção 4.2.3 será apresentada uma formulação original para o problema. A seguir na Seção 4.3 serão descritas algumas heurísticas para o problema de coloração.

4.1. Problema de decisão e complexidade

O problema de decisão de coloração em grafos consiste em determinar a existência ou não de uma $K-$coloração para um dado inteiro K [15]. Seja x_{ik}, $i \in V$, $1 \leq k \leq K$, uma variável binária com valor 1 se o vértice i recebe a cor k e 0 caso contrário. O problema de decisão consiste em determinar se o poliedro $P_{decisão}$ é ou não vazio:

$$\begin{array}{rcll} x_{ik} + x_{jk} & \leq & 1, & \forall ij \in A \text{ e } \forall k \\ \sum_k x_{ik} & \geq & 1, & \forall i \in V \\ x_{ik} & \in & \{0,1\} & \forall i \in V \text{ e } \forall k \end{array} \tag{4.1}$$

Para um grafo G, o poliedro $P_{decisão}$ não será vazio se somente se G possuir uma $K-$coloração. Vemos que se existir uma $K-$coloração para G, então devemos escolher uma cor para cada vértice i dentre aquelas que lhe são atribuídas pelas variáveis binárias $x_{ik} = 1$.

Este problema de decisão de coloração em grafos é $\mathcal{NP}-$Completo, isto é, a existência de um algoritmo polinomial que resolva o problema implicará na solução polinomial de qualquer problema da classe $\mathcal{NP}$. A existência deste algoritmo é improvável pois estariamos afirmando que a classe $\mathcal{NP}$ seria igual à classe de problemas solucionados polinomialmente P.

Teorema 3. *[28] Um Grafo G é 2−colorível se somente se G não contém ciclos ímpares.*

O problema de decidir se existe um ciclo ímpar em um grafo é polinomial [29], logo, pelo Teorema 3, vemos que o problema de decidir se um grafo é $2-$colorível pertence à classe P. Entretanto, os problemas onde $K = 3$ ou 4 contêm todas as dificuldades apresentadas pelo problema geral [28] [7] [8].

4.2. Formulações

O problema de encontrar a $\chi(G)-$coloração de um grafo G pode ser formulado de várias maneiras diferentes. Formulações baseadas em permutações de cores tornam-se desinteressantes por gerar grande simetria nas soluções. Uma alternativa para este problema são as formulações baseadas em conjuntos independentes que apresentam um número pequeno de restrições; entretanto, estas formulações tendem a apresentar um grande número de variáveis.

4.2.1. Formulação simétrica

Uma forma de transformar a formulação do problema de decisão em um problema de otimização é apresentada a seguir [30] [15]. Definimos as variáveis binárias w_j para $j = 1, \ldots, \Delta(G)+1$, que indicam se a cor j é atribuída a algum vértice, isto é, $w_j = 1$ se somente se $x_{ij} = 1$, para algum vértice i. A formulação é dada por.

$$
\begin{array}{lrcll}
\text{Minimizar} & \sum_{j=1}^{\Delta(G)+1} w_j & & & \\
\text{Sujeito a} & \sum_{j=1}^{\Delta(G)+1} x_{ij} & \geq & 1, \quad \forall i \in V & (1) \\
& x_{ij} + x_{kj} & \leq & w_j, \quad \forall (ik) \in A, \text{ e } 1 \leq j \leq \Delta(G)+1 & (2) \\
& x_{ij} & \in & \{0,1\}, \quad \forall i \in V, \text{ e } 1 \leq j \leq \Delta(G)+1 & \\
& w_j & \in & \{0,1\}, \quad 1 \leq j \leq \Delta(G)+1 &
\end{array}
\tag{4.2}
$$

A restrição (1) assegura que cada vértice irá receber exatamente uma cor, enquanto que a restrição (2) impede que dois vértices adjacentes possam receber a mesma cor.

Notemos que esta formulação para o problema de coloração não descarta (torna inviável) soluções simétricas. Por exemplo, qualquer permutação de cores irá gerar uma nova solução pertencente a $P_{decisão}$. Isto define um número exponencial de colorações e torna inviável sua utilização na prática para grandes instâncias.

Apresentaremos a seguir duas novas formulações que buscam eliminar soluções simétricas através da inclusão de algumas desigualdades válidas na formulação 4.2 [30]. O primeiro conjunto de restrições adicionais é definido pelas facetas

$$
\begin{array}{rcll}
w_j & \leq & \sum_{i \in V} x_{ij}, & \forall 1 \leq j \leq \Delta(G)+1 \\
w_j & \geq & w_{j+1}, & \forall 1 \leq j \leq \Delta(G)
\end{array}
\tag{4.3}
$$

Esta nova formulação garante que uma cor j é atribuída a algum vértice somente se a cor $j-1$ já tiver sido atribuída anteriormente. Se forem utilizadas k cores na solução do problema, o modelo elimina permutações de cores maiores que k mas considera viáveis soluções geradas através de permutações das primeiras k cores.

O segundo conjunto de facetas busca eliminar soluções simétricas geradas por permutações de cores através da inserção das restrições 4.4 ao invés das restrições de 4.3 [30]. Para fazer isso, vamos definir apenas as variáveis binárias x_{ij} para $1 \leq j \leq \min\{i, \Delta(G)+1\}$ tal que

$$\begin{array}{lll} x_{ij} \leq & w_j, & \forall i \forall k+1 \leq j \leq \min\{i, \Delta(G)+1\} \quad k = \text{ maior rótulo de um vizinho de } i \\ x_{ij} \leq & \sum_{k=j-1}^{i-1} x_{kj-1}, & \forall i \forall 1 \leq j \leq i \end{array} \tag{4.4}$$

Esta formulação permite que o vértice i seja colorido apenas por uma das primeiras i cores. Para cada solução do modelo representada por uma partição de vértices, temos apenas uma atribuição de cores, pois todas as outras permutações que definem a mesma coloração são pontos inviáveis do problema.

Apesar desta formulação não considerar soluções simétricas de uma mesma coloração, este modelo depende da maneira que os rótulos dos vértices são atribuídos, o que dificulta a caracterização do poliedro do problema e torna desinteressante sua utilização na prática.

4.2.2. Formulação por conjuntos independentes maximais

Esta formulação possui um número menor de restrições que as formulações anteriores e tem como objetivo solucionar o problema de cobertura dos conjuntos independentes maximais que pela Proposição 2 é equivalente ao problema de coloração [15]. Esta formulação, chamada *formulação dos conjuntos independentes maximais* [15], possui uma variável para cada conjunto independente maximal no grafo.

Vamos considerar $\mathcal{S}$ como sendo a família de todos os conjuntos independentes maximais do grafo G. Vamos definir uma formulação com variáveis binárias $x_s \in \{0,1\}$ para cada conjunto independente maximal $s \in \mathcal{S}$. A variável binária $x_s = 1$ implica que todos os vértices do conjunto independente s recebem uma mesma cor, enquanto que ter $x_s = 0$ implica que esse conjunto independente não é escolhido para uma cor (os vértices de s podem receber cores distintas se cobertos por outros conjuntos independentes). O

problema da coloração mínima de um grafo pode ser definido como (denotado por CI, de conjunto independente):

$$\begin{array}{lllll} \text{(CI)} & \text{Minimizar} & \sum_{s\in\mathcal{S}} x_s & & \\ & \text{Sujeito a} & \sum_{\{s\in\mathcal{S}|i\in s\}} x_s & \geq \quad 1, & \forall i \in V \\ & & x_s & \in \{0,1\}, & \forall s \in \mathcal{S} \end{array} \tag{4.5}$$

Esta formulação possui uma restrição por vértice, mas o número de variáveis é tão grande quanto o número de conjuntos independentes maximais, podendo chegar a um número exponencial, o que tornaria o problema difícil de ser tratado.

Outra dificuldade para o problema CI é o fato de os vértices serem multi-coloridos, o que pode gerar colorações equivalentes. Uma forma de tentar evitar a simetria das soluções do problema é utilizar uma formulação alternativa para conjuntos independentes (não só maximais) através da substituição da restrição do problema CI pela restrição 4.6 [15]. Porém, esta abordagem tende a aumentar o número de variáveis do problema.

$$\sum_{\{s\in\mathcal{S}|i\in s\}} x_s = 1, \quad \forall i \in V \tag{4.6}$$

4.2.3. Formulação por subgrafo crítico

Esta formulação, denominada *formulação do subgrafo crítico*, busca explorar as propriedades do grafo crítico sobre o modelo anterior. De forma análoga à formulação por conjuntos independentes maximais, definimos o poliedro P:

$$P(G) = \{x \in \mathbb{R}^n \mid \sum_{\{s\in\mathcal{S}|i\in s\}} x_s \geq 1, \forall i \in V\}$$

onde $x_s \in \{0,1\}$, $\forall s \in \mathcal{S}$.

Seja G' um subgrafo *amigável* de G, isto é, um subgrafo induzido de G com a seguinte propriedade:

Propriedade 1. *Existe uma $\chi(G')$-coloração independente maximal de G' que pode ser estendida para uma $\chi(G')$-coloração de G.*

Como conseqüência da Propriedade 1, vemos que $\chi(G') = \chi(G)$. O problema de achar uma $\chi(G)-$coloração independente maximal de G pode ser reformulado em termos de G' como o problema de encontrar uma $\chi(G')-$coloração independente maximal de G' que possa ser estendida para uma $\chi(G')-$coloração independente maximal de G. A Figura 4.1 ilustra uma $3-$coloração independente maximal do subgrafo G' de G estendida para uma $3-$coloração independente maximal de G.

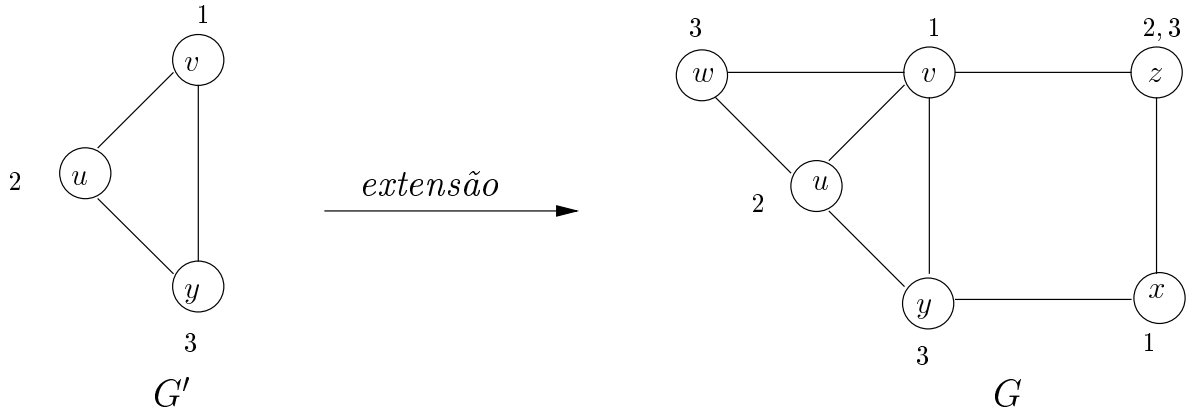


Figura 4.1. *Coloração estendida*

Vemos que o poliedro definido pelas colorações extensíveis do subgrafo G', denotado por $P_{ext}(G')$, está contido em $P(G')$ pois nem toda $K-$coloração independente maximal de G' pode ser estendida para uma $K-$coloração independente maximal de G. Vemos também que

$$\sum_{i:s_i \text{ é u-extensível}} x_i \geq 1, \ \forall u \in V - V'. \tag{4.7}$$

é uma inequação válida de $P_{ext}(G')$. Inequações desta classe serão denotadas de *inequações de vértices*. Esta classe de restrições garante que qualquer vértice do grafo que não pertença ao grafo amigável, possa ser estendido para pelo menos um conjunto independente maximal do grafo amigável.

Uma segunda classe de inequações válidas de $P_{ext}(G')$, denotadas de *inequações de arestas*, são definidas por

$$\sum_{i:s_i \text{ é } (v,w)\text{-extensível}} x_i \geq 2, \ \forall (v,w) \in A(V - V'). \tag{4.8}$$

A intuição por trás deste argumento é que os vértices v e w precisam receber 2 cores diferentes em qualquer coloração de G. Conseqüentemente, devem existir pelo menos 2 conjuntos independentes maximais (diferentes) escolhidos em G' nesta coloração que sejam (v, w)-extensíveis.

A formulação do problema de coloração por subgrafo crítico apresentada nesta seção é original e foi desenvolvida por Corrêa e Frota ao longo desta dissertação [31].

4.3. Heurísticas

As heurísticas se apresentam como uma alternativa de baixo custo para a obtenção de soluções em que o ótimo é desejável, mas não necessário. Em alguns casos, em que as instâncias do problema são muito grandes, as heurísticas são a única alternativa. Além disso, as heurísticas são por vezes úteis como componentes de algoritmos exatos, provendo limitantes para a solução dos problemas. Nesta seção apresentaremos algumas heurísticas para o problema de coloração de grafos.

4.3.1. DSATUR

Uma heurística polinomial gulosa pode conseguir uma coloração ótima de um grafo se os vértices forem visitados em uma ordem correta. Contudo a mesma heurística pode fornecer resultados muito ruins para uma ordem não apropriada dos vértices. Formularemos agora um algoritmo para fornecer uma coloração gulosa baseado em DSATUR (assim denominado por usar o conceito de grau de saturação) [11]. A saída desse algoritmo é

um vetor contendo as cores atribuídas a cada um dos vértices, definindo sua coloração obtida. Se k é o número total de cores atribuídas aos vértices ao final de uma execução, então $Cor[v] \in \{1, 2, \ldots, k\}$, para todo $v \in V(G)$.

No algoritmo, os vértices são ordenados em ordem decrescente do grau de saturação, caso dois vértices tenham o mesmo grau de saturação, a ordem será estabelecida pelo valor do grau dos vértices. A cada iteração do procedimento, é escolhido um vértice $v \in V$ não colorido, tal que $d_s(v) = \max_{w \in V}\{d_s(w)\}$. Este vértice v recebe a menor cor que ainda não tenha sido atribuída a seus vizinhos. O algoritmo DSATUR ilustrado pela Figura 4.2 possui complexidade $O(n^2)$ e é uma heurística simples e eficiente para encontrar uma rápida coloração de um grafo. Resultados da heurística DSATUR e comparações com outros algoritmos podem ser vistos em [11].

```
1.   Algoritmo: DSATUR
2.   Entrada G = (V, A)
3.   Saida  vetor Cor
4.
5.     ORDENAR_DECRESCENTE_GRAU(V, Cor);
6.     Cor[i] = 0 ∀i ∈ V;
7.     Cor[1] = 1;
8.     Enquanto (∃i | Cor[i] = 0)
9.       j = VÉRTICE COM MAIOR GRAU DE SATURAÇÃO;
10.      Cor[j] = MENOR COR DISPONÍVEL;
```

Figura 4.2. *Heurística DSATUR de coloração*

4.3.2. Busca TABU

A heurística tabu, aqui descrita, foi sugerida por Hertz e Werra [32] e apresenta grande eficiência e capacidade de resolver grandes instâncias do problema. Antes do algoritmo, vamos descrever brevemente o método tabu como uma busca, passo a passo, a partir de uma solução viável até uma solução mínima local de alguma função objetivo [33].

Para isso, representaremos cada solução viável do problema por um ponto s. Deno-

taremos por $N(s)$ o *conjunto de soluções vizinhas* do ponto s. A função $f(s)$ definirá o *custo do ponto* s.

O passo básico da técnica Tabu consiste em começar de um ponto s viável e ir gerando um conjunto de soluções vizinhas (de número fixo) em $N(s)$; então deve-se escolher o melhor vizinho s^* gerado e mover para este novo ponto, não importando se $f(s^*)$ é melhor ou não do que $f(s)$.

O que torna a técnica da busca Tabu interessante, é a construção de uma lista T de movimentos proibidos: estes movimentos não são permitidos na presente iteração. A razão para esta lista é a exclusão de movimentos que poderiam nos levar de volta a algum ponto em que nós estávamos em alguma iteração prévia. Vemos que um movimento fica proibido apenas por um certo número de iterações, então na verdade temos uma lista cíclica T onde a cada movimento $s \to s^*$ o movimento oposto $s^* \to s$ é adicionado ao fim da lista T enquanto que o movimento mais antigo de T é removido.

Concluindo, o passo básico da técnica Tabu consiste em gerar aleatoriamente um número fixo (num_{mov}) de movimentos possíveis a partir de s (quando um movimento $(s \to s') \in T$ é gerado, ele é descartado e um novo movimento é gerado). Então o melhor dos movimentos gerados $s \to s^*$ é realizado e a lista Tabu T é atualizada.

Seguindo Glover [34] é utilizada também uma função $\mathcal{A}(f(s))$ denotada de *função de aspiração*. Esta função representa o nível de aspiração a ser atingido pelo valor da função objetivo. Se um movimento para um vizinho s' é proibido ($s \to s' \in T$) mas temos que $f(s') \leq \mathcal{A}(f(s))$, então este movimento será permitido e será considerado um membro do conjunto de vizinhos gerados.

Agora uma condição de parada deve ser definida: geralmente é dado um número máximo nb_{max} de iterações. Neste caso específico, foi utilizada uma estimativa f^* do valor mínimo da função objetivo $f(s)$. Então assim que o valor f^* for alcançado (ou estiver perto o suficiente) o procedimento será finalizado.

Agora vamos descrever como o método da busca Tabu pode ser usado para coloração

de grafos. O algoritmo tenta achar uma coloração para um grafo G que utilize um número fixo k de cores. Depois, com esse algoritmo pode-se ir decrementando o valor de k até atinger a melhor coloração que a heurística possa oferecer.

Dado um grafo G, uma solução viável será uma partição $s = (V_1, V_2, \ldots, V_k)$ dos vértices do conjunto V em um número fixo de k subconjuntos. Denotamos $\mathcal{A}(V_i)$ como o conjunto de arestas $vw \in \mathcal{A}(G) \mid v$ e $w \in V_i$. Agora vamos definir a função objetivo f como o somatório do número de arestas de cada subconjunto V_i.

$$f(s) = \sum(\mid \mathcal{A}(V_i) \mid),\ i = 1, \ldots, k$$

Vemos que s será uma coloração própria de G com k cores se somente se $f(s) = 0$. De fato nós podemos estimar o melhor valor de $f(s)$ com $f^* = 0$; isto nos dará uma condição de parada do procedimento.

A partir de s é gerado um vizinho s' (outra partição de k subconjuntos de vértices) da seguinte maneira: é escolhido um vértice aleatório $x \in V_1 \cup V_2 \cup \ldots \cup V_k$. Então assumindo $x \in V_i$, é escolhida uma cor aleatória $j \neq i$ e obtemos s' de $s = (V_1, V_2, \ldots, V_k)$ pela atribuição:

$$V_j^{'} = V_j \cup \{x\};\quad V_i^{'} = V_i - \{x\};\quad V_r^{'} = V_r \text{ para } r = 1, \ldots, k; r \neq i, j$$

Tendo gerado num_{mov} vizinhos de s (que não pertençam a lista tabu), é realizado um movimento em direção ao melhor vizinho gerado.

A lista Tabu é obtida da seguinte maneira: se um vértice x é movido de V_i para V_j na obtenção de uma nova solução, o par (x, i) se torna proibido: o vértice x não pode retornar ao conjunto V_i por algumas iterações. Como descrito antes, a lista T de movimentos tabu é cíclica.

Agora devemos continuar as iterações até ser alcançada uma solução s onde $f(s) = f^*$ ou ser atingida o número máximo nb_{max} de iterações. Neste caso, não é obtida uma coloração própria de G se a última solução s tiver $f(s) > 0$.

Inicialmente é atribuído $\mathcal{A}(f(s)) = f(s) - 1$ para todos os valores de s. Mas com a geração de vizinhos s' com $f(s') \leq \mathcal{A}(f(s))$ é feita a atribuição $\mathcal{A}(f(s)) = f(s') - 1$. O algoritmo da busca tabu aplicada a coloração em grafos pode ser vista pela Figura 4.3.

```
1.  Algoritmo: TABU
2.  Entrada: G = (V, A), número de cores k, tamanho da lista tabu | T |,
3.   número de vizinhos de cada amostra num_mov, número máximo de iterações nb_max
4.  Saida: coloração s = (V_1, V_2, ..., V_k)
5.
6.    Gera uma solução aleatória s = (V_1, ..., V_k);
7.    nb_iter = 0;
8.    T = ∅;
9.    Enquanto f(s) > 0 e nb_iter < nb_max Faça
10.     Gera num_mov vizinhos s_i de s com (s → s_i ∉ T) ou f(s_i) ≤ A(f(s));
11.     s' = BEST (s_1, s_2, ..., s_num_mov);
12.     T = T ∪ {s' → s};
13.     T = T− (movimento mais antigo);
14.     s = s';
15.     nb_iter = nb_iter + 1;
```

Figura 4.3. *Heurística de busca TABU para coloração de grafos*

Segundo experimentos de Hertz e Werra, a técnica Tabu apresentada mostra resultados superiores quando comparada a outras técnicas como têmpera simulada [33]. O método Tabu consegue encontrar colorações com menos cores e em menos tempo computacional [32].

4.3.3. Outras heurísticas

Barbosa, Assis e Nascimento [35] apresentam duas formulações heurísticas para o problema de coloração em vértices baseado em algoritmos genéticos. A primeira é baseada na relação que existe entre o número cromático de um grafo e suas orientações acíclicas. Esta formulação considera o conjunto de orientações acíclicas de um grafo G como indivíduos que sofrem a ação de operadores evolutivos.

A segunda formulação, diferente da primeira, não objetiva colorir um grafo apenas, mas

sim toda uma classe de grafos que tenham o mesmo número de vértices e outros atributos comuns entre si. Esta formulação considera o conjunto de permutações das ordens dos vértices $\langle 1, \ldots, n \rangle$ como indivíduos capazes de atribuir uma coloração a qualquer grafo de n vértices. O objetivo desta formulação é criar modelos de indivíduos que na média possam colorir um grafo com o menor número de cores possível.

Várias outras heurísticas foram criadas para o problema de coloração em grafos, sempre com o objetivo de encontrar colorações que utilizavam menos cores e em um curto espaço de tempo. Recentemente um esforço concentrado foi feito para testar diversas heurísticas sobre um conjunto de grafos de vários tamanhos e de diferentes características. Este esforço foi parte do segundo desafio de implementação de DIMACS [14]. Hoje em dia grande parte dos trabalhos realizadas na área de coloração em grafos realizam experimentos computacionais sobre este conjunto de instâncias.

5

Problema de Coloração de Grafos: Algoritmos Exatos

A seguir apresentaremos alguns algoritmos exatos para resolução do problema de coloração de vértices. Na Seção 5.1 será apresentado um algoritmo de enumeração implícita baseado na heurística DSATUR (Seção 4.3), depois na Seção 5.2 será apresentado o algoritmo proposto por Herrmann e Hertz para encontrar o número cromático de um grafo utilizando o conceito de grafo crítico. Na Seção 5.3 apresentaremos um método de geração de colunas proposto por Mehrotra e Trick para solucionar a formulação dos conjuntos independentes maximais apresentada na Seção 4.2. A descrição desses métodos neste capítulo objetiva apresentar os algoritmos que serviram de inspiração para os novos métodos propostos no Capítulo 6. Finalmente, na Seção 5.4 serão mencionados outros algoritmos de enumeração implícita encontrados na literatura.

5.1. Método de busca baseado em DSATUR

Brélaz [11] desenvolveu um algoritmo exato de coloração de grafos baseado na heurística DSATUR que evita redundâncias na enumeração das soluções. O algoritmo funciona dividindo a instância de coloração em vários subproblemas. Um subproblema do algoritmo

exato de DSATUR é definido como uma coloração parcial do grafo, isto é, uma atribuição de cores a um subconjunto dos vértices do grafo. A cada iteração do algoritmo, existe um limite superior U sobre o número de cores necessário para colorir o grafo. Se um subproblema utiliza k cores numa coloração parcial do grafo e $k \geq U$, claramente vemos que este subproblema pode ser excluído da busca. Agora, se para todo vértice do grafo for atribuída uma cor (coloração completa), e tivermos $k < U$, então uma coloração com menos cores foi encontrada e devemos atualizar o limite superior U.

O método de busca para o DSATUR é definido como um método de busca do tipo *branch-and-bound* (Seção 3.6) no qual, para cada subproblema $\sigma \in H_1$, temos que Nível(σ) é o nível em que o subproblema σ se encontra na árvore de ramificação. Dado um subproblema $\sigma \in H_1$, temos que a prioridade de σ é igual a Nível(σ).

Um subproblema p_σ de σ, no método de busca para o DSATUR é definido como um vetor $cor \in \mathbb{N}^{|V|}$ que irá armazenar a coloração dos vértices do grafo, tal que a cor, $cor[i]$, é atribuída ao vértice v_i para todo $v_i \in V$. Se $cor[i] = 0$, então o vértice v_i não recebeu nenhuma cor. Para todo subproblema estendido σ selecionado de H_1 pelo operador Op_1, é chamada a função de Atualização se o subproblema possuir uma coloração completa, isto é, para todo vértice $v_i \in V$ temos que $p_\sigma.cor[v_i] \neq 0$. Caso uma nova solução não seja encontrada, devemos iniciar uma ramificação do subproblema σ. A função Vértice_saturação irá encontrar um vértice v_i tal que $d_s(v_i) = \max_{w \in V}\{d_s(w)\}$. A ramificação irá gerar um conjunto de novos subproblemas $\sigma^1, \ldots, \sigma^{|RAM|}$ que serão inseridos em H_1. O subproblema σ^j é criado atribuindo a cor j, onde j é uma das k cores utilizadas no problema, ao vértice v_i se e somente se esta atribuição induzir uma coloração própria em G. Adicionalmente é criado um novo subproblema atribuindo a cor $k+1$ ao vértice v_i (Figura 5.1).

Sewell [36] sugeriu uma modificação para o algoritmo DSATUR com o objetivo de diminuir o número de vértices a serem coloridos durante a enumeração. Esta modificação consiste em determinar uma clique maximal C do grafo e colorir seus vértices numa fase inicial do algoritmo. Vemos que os vértices de C nunca terão de ser recoloridos,

```
1.  Algoritmo: OP1
2.
3.    σ ← H1.SELEÇÃO;
4.    Se (p_σ.cor[v_i] ≠ 0 ∀v_i ∈ V(G)) Então
5.      Se l_σ ≤ U Então
6.        ATUALIZAÇÃO(p_σ, l_σ);
7.    Senão
8.      v_i = VÉRTICE_SATURAÇÃO(p_σ);
9.      k = NÚMERO_CORES_UTILIZADAS(p_σ);
10.     Para (j = 1...k + 1) Faça
11.       Se (NÃO_POSSUE_VIZINHO_COLORIDO(v_i, j) Então
12.         Se j ≤ k Então
13.           l_σ^j = k;
14.         Senão
15.           l_σ^j = k + 1;
16.         p_σ.cor[v_i] = j;
17.         h_σ^j = NÍVEL(σ) + 1;
18.         H1.INCLUSÃO(σ^j);
```

Figura 5.1. *Operador do método DSATUR de coloração*

pois os vértices de qualquer clique do grafo devem ter cores distintas. Logo, o problema inicial p_σ inserido em H_1 possui os vértices da clique já coloridos não sendo necessária sua enumeração. Isso sugere que pode ser útil achar a clique máxima no grafo e colorir estes vértices primeiro. Esta abordagem tem grande ganho quando o valor da clique e o número cromático do grafo são próximos.

Encontrar a clique máxima de um grafo é um problema tão difícil quanto o de coloração mínima. Logo, para a modificação sugerida por Sewell, não é interessante buscar uma clique máxima no grafo quando uma clique maximal pode ter bons resultados a um custo bem mais reduzido. A Figura 5.2 ilustra um algoritmo guloso para encontrar uma clique maximal do grafo. O algoritmo se divide em duas partes. Primeiro é construída, de maneira gulosa, uma clique maximal no vetor *candidatos* $\in \mathbb{N}^n$ a partir de uma ordem aleatória dos vértices fornecida pela função ORDEM_ALEATÓRIA. O número de repetições desta fase, denominada fase aleatória, é determinada pelo parâmetro de entrada α. Analogamente, na segunda fase do algoritmo, denominada fase grau, é construída, de

maneira gulosa, uma clique maximal a partir da ordem decrescente dos graus dos vértices do grafo fornecida pela função ORDEM_DECRESCENTE_GRAU. No fim do algoritmo o vetor $melhor \in \mathbb{R}^n$ guardará a clique maximal de maior cardinalidade encontrada.

```
1.  Algoritmo: CLIQUE_GULOSA
2.  Entrada G = (V, A), α
3.  Saida CliqueMaximal
4.
5.    melhor = ∅;
6.    Para (i = 1 ... α) Faça                          /* fase aleatória */
7.      candidatos = ORDEM_ALEATÓRIA(V);
8.      Para (j = 1 ... | candidatos |) Faça
9.        candidatos = candidatos − Anti(candidatos[j], G);
10.     Se | candidatos |>| melhor | Então
11.       melhor = candidatos;
12.
13.   candidatos = ORDEM_DECRESCENTE_GRAU(V);          /* fase grau */
14.   Para (i = 1 ... | candidatos |) Faça
15.     candidatos = candidatos − Anti(candidatos[i], G);
16.   Se | candidatos |>| melhor | Então
17.     melhor = candidatos;
18.   Retorne melhor
```

Figura 5.2. *Heurística gulosa para encontrar clique maximal.*

A Figura 5.3 ilustra o subproblema inicial p_σ inserido em H_1 onde a clique formada pelos vértices u, v, x foi identificada pela função CLIQUE_GULOSA e colorida na fase inicial do algoritmo. Os subproblemas p_σ^1 e p_σ^2 são resultado da ramificação sobre o vértice de maior grau de saturação y.

5.2. Método do subgrafo crítico

Herrmann e Hertz [16] propruseram um algoritmo para encontrar o número cromático de um grafo baseado na combinação de uma heurística e de um algoritmo exato para coloração em grafos. A idéia deste algoritmo repousa em duas fases: a primeira, denominada *fase decrescente*, se constitui na tentativa de determinar um subgrafo crítico do grafo

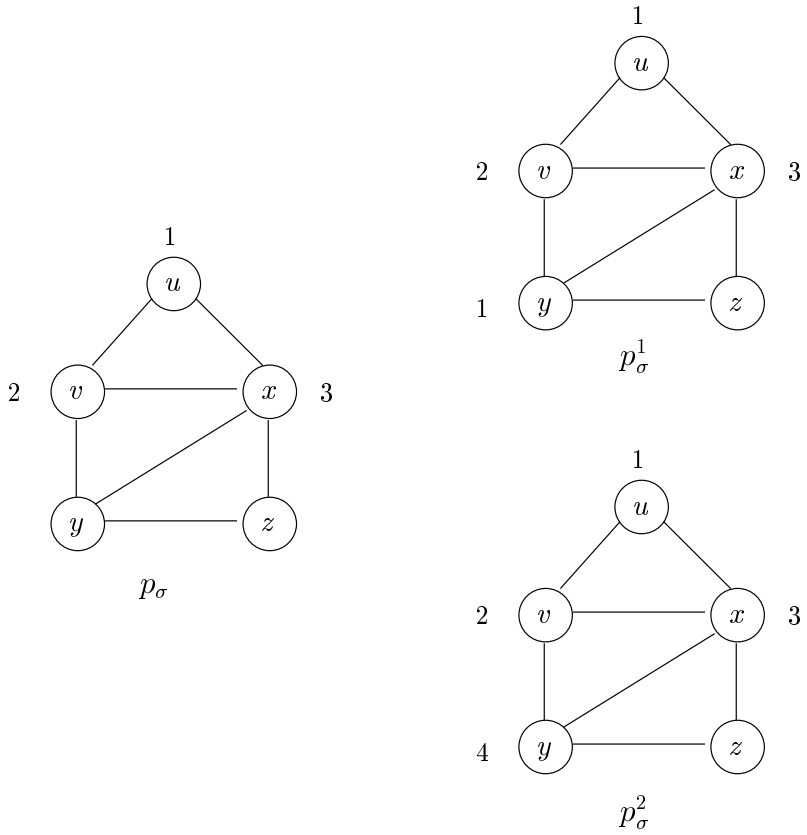


Figura 5.3. *Problema inicial do algoritmo DSATUR*

original através de uma heurística e encontrar seu número cromático "na esperança" que este subgrafo seja menor que o grafo original. A segunda fase, denominada *fase crescente*, é uma fase expansiva onde vértices são adicionados ao subgrafo seguindo critérios determinados pelo algoritmo. Este método é especificamente interessante quando se possui uma coloração de um grafo e desejamos provar sua otimalidade.

Vamos denotar Γ como sendo uma heurística que determina uma coloração própria do grafo G, onde $\Gamma(G)$ é o número de cores distintas usadas na coloração produzida por Γ. Claramente vemos que $\Gamma(G) \geq \chi(G)$. A seguir vamos apresentar uma heurística, denominada Criação_Grafo_Crítico, que é baseada em Γ. Esta heurística visa determinar um subgrafo induzido $\Gamma-$crítico G' a partir de G, onde G' é crítico com relação à coloração determinada por Γ (Figura 5.4). O subgrafo G' é obtido através da remoção iterativa de vértices de G. Para isto, G' é inicializado em G, e os vértices em V são examinados em

uma ordem qualquer. Para cada vértice $v \in V$, v será removido de G' se somente se $\Gamma(G' - \{v\}) = \Gamma(G)$. No fim da execução, a heurística retorna um subgrafo $\Gamma-$crítico G'.

```
1.   Algoritmo: Criação_Grafo_Crítico
2.   Entrada G = (V, A) e Γ
3.   Saida G' = (V', A')
4.
5.      G' = G;
6.      ∀v ∈ V' Faça
7.         μ = Γ(G' − v);
8.         Se μ = Γ(G) Então
9.            G' = G' − {v};
10.     Retorne G'
```

Figura 5.4. *Heurística que visa determinar o grafo crítico de G.*

Agora vamos detalhar o algoritmo do método do subgrafo crítico. No início da fase decrescente, é calculado um limite superior $\Gamma(G)$ para $\chi(G)$. Em seguida, é determinado um subgrafo $\Gamma(G)-$crítico G' de G através da função Criação_Grafo_Crítico (Figura 5.4). Sobre o subgrafo G' gerado é aplicado o método de coloração exato do DSATUR (Seção 5.1) que determinará o número cromático de G'.

Proposição 7. *[16] Seja $\Gamma(G)$ um limite superior sobre o número cromático $\chi(G)$ do grafo G, e seja G' um subgrafo induzido de G. Logo temos que $\chi(G') \leq \chi(G) \leq \Gamma(G)$. Se $\chi(G') = \Gamma(G)$, então $\chi(G) = \chi(G')$.*

Proposição 8. *[16] Se todos os limites superiores produzidos por Γ na fase decrescente do algoritmo da Figura 5.5 são iguais ao número cromático do grafo dado como entrada dessa heurística ($\Gamma(G') = \chi(G')$), então o algoritmo pára na fase decrescente.*

Pela Proposição 8, temos que o algoritmo pára se o limite superior $\Gamma(G)$ calculado sobre G for igual ao número cromático de G' . Entretanto, se $\chi(G') < \Gamma(G)$ então será iniciada a fase crescente do algoritmo. O objetivo desta fase é encontrar um vértice que pertencia ao grafo crítico mas foi retirado por engano na fase decrescente. Para isso é

construído um conjunto $List \subseteq V(G)$ tal que para todo vértice $x \in List$ temos que $\chi(G' + \{x\}) = \chi(G') + 1$. Este conjunto de vértices é construído percorrendo os vértices $x \in G - G'$ e calculando o valor de $\chi(G' + \{x\})$ somente se $\Gamma(G' + \{x\}) > \chi(G')$.

A seguir é adicionado um vértice x ao subgrafo G'. Se o conjunto $List$ não estiver vazio, então um vértice x é escolhido em $List$ e o número crómatico de G' é incrementado de uma unidade, caso contrário um vértice $x \in G - G'$ é escolhido e $\chi(G' + \{x\})$ recebe o valor de $\chi(G')$.

Se, durante a fase crescente, for encontrado um subgrafo G' cujo valor do limite superior $\Gamma(G)$ for igual a $\chi(G')$, temos, pela Proposição 7, que o algoritmo pára. Senão, os vértices $x \in G - G'$ serão iterativamente inseridos em G' até que $G = G'$, o que indica que $\chi(G) = \chi(G')$.

Vemos que, se o limite superior $\Gamma(G)$ for estritamente maior do que $\chi(G)$, então não existe benefício nenhum em usar o método do subgrafo crítico. Logo a escolha de uma heurística Γ eficiente é fundamental para o desempenho deste algoritmo. Experimentos realizados em [16] mostram que quando o grafo de entrada G é crítico, nenhum vértice é retirado durante a fase decrescente. Em todos os outros casos, o subgrafo crítico G', determinado pela função Criação_Grafo_Crítico, é menor que o grafo original G, e possivelmente mais fácil de colorir.

5.3. Método da geração de colunas

Nesta seção será apresentado um método de coloração de grafos baseado na formulação CI de programação inteira do problema de coloração de grafos descrita na Seção 4.2.

A formulação CI possui uma variável para cada conjunto independente maximal do grafo. Vemos que CI pode ter mais variáveis do que se pode tratar diretamente. Nós iremos resolver essa dificuldade usando apenas um subconjunto das variáveis e gerando as demais quando for necessário. Essa técnica, chamada de *geração de colunas*, é bem conhecida em programação matemática, porém a necessidade de gerar colunas a cada

```
1.  Algoritmo: MÉTODO_SUBGRAFO_CRÍTICO
2.  Entrada G = (V, A)
3.  Saida χ(G)
4.
5.  /* FaseDecrescente */
6.    k = Γ(G);
7.    G' = CRIAÇÃO_GRAFO_CRÍTICO (G, k);
8.    k' = MÉTODO_DSATUR (G');
9.    Se (k' = k) Então
10.     Retorne k;
11.
12.  /* FaseCrescente */
13.   List = ∅;
14.   ∀x ∈ G − G' Faça
15.     μ = Γ(G' + {x});
16.     Se μ > k' Então
17.       k'' = MÉTODO_DSATUR (G' + {x});
18.       Se (k'' = k' + 1) Então
19.         List = List ∪ {x};
20.   Se List ≠ ∅ Então
21.     G' = G' + {x} | x ∈ List;
22.     k' = k' + 1;
23.   Senão
24.     G' = G' + {x} | x ∈ (G − G');
25.   Se G' = G ou k' = k Então
26.     Retorne k';
27.   Senão
28.     Recomeçe Fase 2;
```

Figura 5.5. *Algoritmo do método do grafo crítico*

iteração do método torna o procedimento não trivial.

O procedimento de geração de colunas será aplicado sobre a relaxação linear do problema CI, denotado por CIR. Seja $\mathcal{S}$ o conjunto de todos os conjuntos independentes maximais de G. O procedimento de geração de colunas começa com um subconjunto $\overline{\mathcal{S}} \subseteq \mathcal{S}$ de conjuntos independentes que cubram os vértices de G ($\overline{\mathcal{S}}$ é calculado heuristicamente). A partir desta solução inicial, solucionamos o problema CIR restrito a $s \in \overline{\mathcal{S}}$. Isto dará uma solução viável para CIR e um vetor de multiplicadores duais π. Agora, devemos determinar se a solução ótima corrente de CIR restrito a $s \in \overline{\mathcal{S}}$ é ótima também para CIR, ou se seria vantajoso expandir o conjunto $\overline{\mathcal{S}}$ com o intuito de melhorar a função objetivo.

Pelo Teorema 2 podemos verificar se a solução corrente é ótima para CIR averiguando se seus custos reduzidos, associados a todas as variáveis x_s do problema, possuem valores não negativos. Note que o custo reduzido de uma variável x_s vale:

$$Z_s = 1 - \pi^T z^s$$

onde $z^s = [z_i^s]_{i \in V}$ é o vetor característico de $s \in \mathcal{S}$, definido por:

$$z_i^s = \begin{cases} 1, & \text{se } i \in V \\ 0, & \text{caso contrário} \end{cases}$$

Se determinarmos o menor custo reduzido e este for não negativo, a solução corrente é ótima para CIR (Teorema 2). Caso contrário, podemos promover uma troca de colunas que irá gerar um melhoramento na função objetivo. O menor custo reduzido é determinado por:

$$Z^{min} = \min_{s \in \mathcal{S}} Z_s = \min_{s \in \mathcal{S}} \{1 - \pi^T z^s\} = 1 - \max_{s \in \mathcal{S}} \{\pi^T z^s\}.$$

No contexto da nossa técnica de geração de colunas, não é necessário que nós consigamos o menor custo reduzido: é suficiente conseguir qualquer custo reduzido negativo, o que nos indicaria uma direção de melhoria da função objetivo do problema CIR. Então

devemos solucionar o seguinte problema de decisão: existe um conjunto independente definido por um vetor característico z tal que $\sum_{i \in V} \pi_i^T z_i > 1$, onde cada vértice i está associado a uma restrição do problema mestre CIR, que por sua vez está associado a um valor π_i?

Este problema é conhecido como o *problema de decisão do conjunto independente ponderado* (CIP). Vemos que, caso exista um conjunto independente que responda "sim" para CIP, ele irá definir uma coluna para o problema CIR, e conseqüentemente, uma variável x_s com um custo reduzido negativo. Por essa razão, o conjunto $\overline{\mathcal{S}}$ deve ser expandido para conter o novo CI, pois sua inclusão acarretará um ganho na função objetivo do problema mestre CIR. Logo, o vetor z define a nova coluna de CIR que será adicionada ao problema. Se, ao contrário, não existir um conjunto independente que satisfaça o problema CIP, então não existe nenhum custo reduzido negativo, ou seja, não existe nenhum conjunto independente que, ao ser adicionado a $\overline{\mathcal{S}}$, promoverá uma melhoria de CIR. Portanto, a solução ótima para o problema CIR restrito a $\overline{\mathcal{S}}$ também é a solução ótima do problema CIR.

Esse processo é repetido até que não exista mais nenhum conjunto independente que possa provocar uma melhoria no problema mestre CIR. Esta solução ótima para CIR representa um limite inferior para o problema original CI (Proposição 6). Se a solução ótima para o problema CIR tiver x_s inteiro para todo $s \in \overline{\mathcal{S}}$, então esta solução corresponde a uma solução ótima do problema CI. Quando alguma variável x_s for não inteira, teremos que forçar a integralidade através de ramificações sobre os conjuntos independentes. Este passo será detalhado na Seção 5.3.2.

Infelizmente, CIP é um problema da classe $\mathcal{NP}$, logo, temos que desenvolver técnicas capazes de solucionar este problema que sejam suficientemente velozes para serem utilizadas a cada iteração do procedimento.

5.3.1. Um algoritmo para o problema CIP

O problema de decisão do conjunto independente ponderado é um problema bastante estudado na teoria dos grafos e em pesquisa operacional. Nesta seção, exibiremos um algoritmo recursivo simples descrito no trabalho de Bassam [37] que, além de responder "sim" ou "não", retorna um conjunto independente que prova resposta "sim", se for o caso. Descreveremos ainda uma heurística gulosa que pode ser usada para reduzir a necessidade do algoritmo recursivo.

A idéia básica do algoritimo recursivo para achar o conjunto independente ponderado que satisfaça CIP é a seguinte: dado um grafo G e um vértice $i \in V$, o CIP em G é o CIP em G restrito a $(V - \{i\})$ ou o CIP do grafo induzido por i unido com $Anti_G(i)$. A idéia dessa recursão pode ser vista pela Equação 5.1 e que servirá de base para o algoritmo mostrado na Figura 5.6:

$$CIP(G \cup \{i\}) = \max(CIP(G), CIP(G(\{i\} \cup Anti_G(i)))) \tag{5.1}$$

```
1.   Algoritmo: CIP
2.   Entrada um grafo G
3.   Saída um vetor de vértices Best
4.   Variáveis Globais Best = NULL
5.
6.      H ← HEURÍSTICA_GULOSA_CIP(G);
7.      Best ← max{Best, H};
8.      {i_1, i_2, ..., i_k} = V(G) − V(H);
9.      Para j = 1...k Faça
10.        H ← H ∪ {i_j};
11.        Se (| ({i_j} ∪ Anti_H(i_j)) |>| Best |) Então
12.           CIP ({i_j} + Anti_H(i_j));
```

Figura 5.6. *Algoritmo recursivo para problemas (CIP).*

Como não é necessário que nós consigamos o melhor (máximo) conjunto independente ponderado, uma abordagem heurística para encontrar uma coluna de melhoramento no programa mestre CIR pode ser suficiente em muitos casos. Somente é necessário usar o

algoritmo da recursão quando for necessário provar que não existem conjuntos independentes com peso maior que 1 (quando a heurística falha).

Existem muitas heurísticas para conjuntos independentes ponderados. Uma das mais simples é a heurística gulosa demonstrada pela Figura 5.7 [38]. Este algoritmo descrito a seguir foi utilizado nos experimentos apresentados em [15]. A heurística começa com o vértice de maior peso. Adiciona vértices em ordem não crescente de seus pesos com o cuidado de que o conjunto resultante sempre seja um conjunto independente. Esta heurística é simples, muito rápida, e parece trabalhar razoavelmente bem. O conjunto independente resultante pode ser adicionado diretamente ao problema mestre CIR (se seu valor for maior que 1) ou pode servir de ponto de partida para o algoritmo recursivo.

```
1.   Algoritmo: HEURÍSTICA_GULOSA_CIP
2.   Entrada um grafo G
3.   Saída um conjunto independente H
4.
5.      H ← ∅
6.      {i_1, i_2, ..., i_n} ← ORDENA_PESOS(G);
7.      Para j = 1 ... n Faça
8.         Se (i_j v) ∉ A(G) ∀v ∈ V(H) Então
9.            H ← H ∪ {i_j}
10.     Retorna H
```

Figura 5.7. *Heurística gulosa para problemas (CIP).*

5.3.2. Ramificação para o método da geração de colunas

O operador de ramificação é responsável pela decomposição do problema original em subproblemas menores com o objetivo de percorrer o espaço de solução do problema. Uma dificuldade em usar a técnica de geração de colunas para programas inteiros é o desenvolvimento de uma regra de ramificação para assegurar a integralidade do problema. Regras que são apropriadas para problemas em que todo o conjunto de colunas está explicitamente disponível, não cabem para problemas onde as colunas são geradas por

técnicas implícitas [15].

Uma abordagem para esse problema é usar o operador de ramificação para buscar a integralidade, sem aumentar a complexidade dos subproblemas. Isto é feito através de uma regra de ramificação que garanta que o subproblema a ser resolvido em cada ramo seja um problema de coloração em grafos sem restrições adicionais e que pode ser resolvido também pela técnica de geração de colunas [15].

Considere uma solução fracionária do problema CIR. Vemos que existe uma variável básica x_{s_1} que representa um conjunto independente s_1 e que possui um valor fracionário (Figura 5.8).

Lema 1. *[15] Se x_{s_1}, que representa um conjunto independente s_1, é fracionário, então existem vértices i, j e um outro conjunto independente s_2 tais que $i \in s_1 \cap s_2$, e $j \in s_1 - s_2$.*

Prova: Para construir nossa prova temos que demonstrar que, dada uma solução fracionária do problema, existem dois conjuntos independentes s_1 e s_2 onde $s_1 \cap s_2 \neq \emptyset$ e $s_1 - s_2 \neq \emptyset$. Primeiro, vamos supor, por contradição, que não existe dois conjuntos independentes s_1 e s_2 onde $s_1 \cap s_2 \neq \emptyset$, em uma solução viável de CIR que não se constitui numa solução para CI porque possui uma variável fracionária x_{s_1}. Pelas restrições do problema CIR, todos os vértices do grafo devem pertencer a pelo menos um conjunto independente e, pela hipótese, os conjuntos independentes não possuem interseção. Logo, os conjuntos independentes induzem uma coloração no grafo que se constitui numa solução viável para o problema CI. Como segunda parte da prova, vamos agora supor, novamente por contradição, que $s_1 - s_2 = \emptyset$. Como s_1 e s_2 são conjuntos não vazios, obrigatoriamente o conjunto s_2 deve conter o conjunto s_1 ($s_1 \subseteq s_2$), contrariando o modelo proposto do problema CI, que afirma que os conjuntos independentes são maximais. □

Vemos que o vértice i pertence a dois conjuntos independentes, o que provoca a solução fracionária do problema. A ramificação adotada para este problema é definida na Seção 3.4 onde devemos criar dois novos subproblemas σ^1 e σ^2 através de operações sobre os

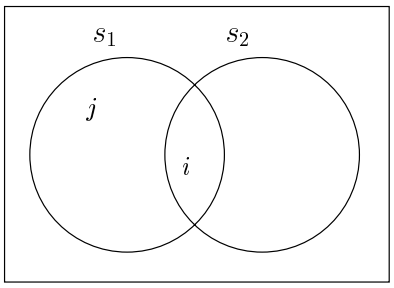


Figura 5.8. *Conjuntos independentes s_1 e s_2.*

vértices i e j. O subproblema σ^1 será gerado através da identificação dos vértices i e j em IDENT(i, j), o que forçará a sua presença no mesmo conjunto independente; no subproblema σ^2, os vértices i e j se tornarão adjacentes, o que forçará suas presenças em conjuntos independentes diferentes. Qualquer coloração viável deve estar em exatamente uma destas duas ramificações.

5.3.3. Método de busca para o método da geração de colunas

O método de busca para a geração de colunas é definido como um método de busca de *branch-and-bound* onde para cada subproblema $\sigma \in H_1$ temos que a prioridade do subproblema σ é o valor de limite inferior fornecido pela resolução da relaxação linear CIR do subproblema σ.

O operador OP$_1$ do método da geração de colunas (Figura 5.9) inicia seu processo selecionando um subproblema $\sigma \in H_1$. O valor de limite inferior l_σ é calculado através da execução do procedimento de geração de colunas, denotado por GERAÇÃO_DE_COLUNAS, sobre a relaxação linear de σ. É chamada a função de ATUALIZAÇÃO se o subproblema σ for solucionável, isto é, se todas as variáveis de uma solução ótima x para o problema CIR forem inteiras, implicando que x é uma solução ótima para o problema CI também. Caso a solução x de CIR não seja inteira, devemos forçar a integralidade através de uma ramificação sobre o subproblema.

A função de ramificação, denotada RAMIFICAÇÃO (Figura 5.10), receberá como entrada um subproblema σ que dará origem a dois novos subproblemas σ^1 e σ^2, onde σ^1 é

obtido de σ através da identificação dos vértices i, j; e σ^2 é obtido de σ pela inserção da aresta ij com $i, j \in V(p_\sigma)$.

A função IJ_RAMIFICAÇÃO é responsável pela escolha dos vértices $i, j \in V(p_\sigma)$. Para sua implementação, devemos primeiro determinar a variável x_1, que corresponde a uma coluna s_1 mais fracionária do problema CIR. Então devemos encontrar a primeira linha i (que corresponde a uma restrição do vértice i) coberta por essa coluna, depois devemos determinar outra coluna (s_2) que também cubra a linha i. Então devemos encontrar uma linha j, que corresponde ao vértice j, onde apenas uma das colunas escolhidas (s_1 ou s_2) cubra a coluna j.

O algoritmo descrito nesta seção foi implementado, o Capítulo 7 contém alguns resultados e análises.

```
1.   Algoritmo: OP1
2.
3.     σ ← H1.SELEÇÃO;
4.     l_σ = GERAÇÃO_DE_COLUNAS(σ);
5.     h_σ = l_σ;
6.     Se (x_s inteiro ∀x_s ∈ VARIÁVEIS(CIR)) Então
7.       Se l_σ ≤ U Então
8.         ATUALIZAÇÃO;
9.
10.    RAMIFICAÇÃO(σ, σ^1, σ^2)
11.    H1.INCLUSÃO(σ^1);
12.    H1.INCLUSÃO(σ^2);
```

Figura 5.9. *Operador do método geração de colunas.*

5.4. Outros algoritmos

Algoritmos de enumeração implícita são bastante utilizados na solução de problemas de coloração de grafos por serem flexíveis e eficientes. O primeiro algoritmo de coloração deste tipo foi proposto por Brown [10] em 1972. A idéia principal deste algoritmo é baseada na seguinte regra: dada uma ordem dos vértices do grafo G, devemos colorir estes vértices

```
1.   Algoritmo: RAMIFICAÇÃO
2.   Entrada um subproblema σ
3.   Saida  dois subproblemas σ¹ e σ²
4.
5.     IJ_RAMIFICAÇÃO(p_σ, i, j);
6.     p_σ¹ ← DUPLICA_GRAFO(p_σ);
7.     p_σ¹ ← IDENTIFICA_VERT(p_σ¹, i, j);
8.
9.     p_σ² ← DUPLICA_GRAFO(p_σ);
10.    p_σ² ← INSERE_ARESTA(p_σ², i, j);
11.    Retorna σ¹ e σ²
```

Figura 5.10. *Algoritmo de ramificação do método geração de colunas.*

executando dois passos alternadamente. O primeiro irá colorir os vértices seguindo a ordem crescente estabelecida até encontrar um vértice que não possa ser colorido com o número de cores corrente. O segundo passo irá retroceder seqüencialmente na ordem até encontrar um vértice que possa ser colorido com outra cor [9].

Brélaz [11] em 1979, observou que poderia diminuir o número de problemas analisados se ordenasse os vértices do grafo utilizando o conceito de grau de saturação. Esta nova abordagem demonstrou melhores resultados que o algoritmo de Brown e deu origem ao algoritmo exato de DSATUR descrito na Seção 5.1. Mais tarde Jürgen [39] fez uma correção no algoritmo de Breláz com respeito à atualização de vértices predecessores na ordem.

Korman [12] sugeriu também uma modificação para o algoritmo de Brown. Ele notou que durante a execução do algoritmo, podíamos atribuir um número maior de cores viáveis para alguns vértices e que os vértices com poucas possibilidades de cores viáveis deveriam ser coloridos primeiro. Então ele sugeriu uma regra de rearranjo dinâmico da ordem dos vértices previamente estabelecida [9]. Esta regra diz que dado um subconjunto de vértices não coloridos $\{v_i, \ldots, v_n\}$ devemos promover uma troca na ordem dos vértices entre o vértice v_i e o vértice v_j $(j > i)$ onde v_j é o vértice com o menor número de possibilidades de cores atribuidas a ele.

Notemos que os métodos derivados do modelo de Brown vistos aqui (Brélaz, Sewell e Korman) direcionaram seus esforços para melhorar o algoritmo inicial através da sugestão de novas ordens para a análise dos vértices. Isto porque a ordem em que os vértices de um grafo G são coloridos pode ter grande influência no desempenho do algoritmo.

6

Problema de Coloração de Grafos: Novos Algoritmos Exatos

Neste capítulo serão mostrados os novos métodos que foram desenvolvidos durante a elaboração desta dissertação. Na Seção 6.1 mostraremos uma modificação do método DSATUR apresentado na Seção 5.1. A idéia inicial deste novo método, denominado de DSATUR crítico, é aproveitar o conceito do grafo crítico para gerar uma nova ordem para os vértices com o objetivo de aprimorar o método DSATUR. Logo a seguir na Seção 6.2 apresentaremos um novo algoritmo de programação linear baseado nos trabalhos de Mehrotra e Trick [15] e Herrmann e Hertz [16]. O objetivo deste método, denominado de coloração amigável, é unir a força do método de geração de colunas proposto por Mehrotra e Trick [15], que possui a capacidade de gerar limites inferiores muito próximas do valor do número cromático, porém requer a solução de problemas de grande custo computacional, com a tentativa de Herrmann e Hertz [16] de construir um grafo crítico para diminuir a complexidade do problema. Os resultados e análises das implementações dos algoritmos propostos neste capítulo são apresentados no Capítulo 7.

6.1. DSATUR crítico

Como foi visto na Seção 5.1, o método de coloração DSATUR enumera as soluções viáveis do problema atribuindo cores aos vértices do grafo de acordo com uma ordem sobre seus graus de saturação. Porém, devemos colorir primeiro os vértices de uma clique (de preferência máxima), pois estes vértices não precisarão ser recoloridos. A idéia principal do método DSATUR crítico é que, analogamente aos vértices da clique do grafo, devemos colorir primeiro os vértices do subgrafo crítico pois estes têm menos chances de ser recoloridos que os demais vértices do grafo. A modificação do método DSATUR é apresentada a seguir.

Seja G' um subgrafo induzido obtido de G através do algoritmo de criação de um grafo $\Gamma-$crítico ilustrado pela Figura 5.4. Seja ainda C uma clique maximal obtida de G' através da heurística gulosa apresentada na Figura 5.2.

A nova ordem dos vértices a serem coloridos pelo método DSATUR crítico será

1) vértices de C

2) vértices de $G' - C$, usando método DSATUR

3) vértices de $G - G'$, usando método DSATUR

Os primerios vértices analisados nesta nova ordem serão os vértices pertencentes á clique C do subgrafo $\Gamma-$crítico G' que não precisarão ser coloridos novamente. O segundo grupo de vértices analisados serão os vértices que pertencem ao subgrafo crítico G' mas não pertencem á clique C. Deixamos por último os vértices do grafo G que não pertencem ao subgrafo $\Gamma-$crítico G'. A ordenação entre os vértices de cada grupo será determinada decrescentemente em relação ao grau de saturação dos vértices.

O método DSATUR crítico é similar ao método DSATUR (Seção 5.1), com a diferença que na escolha dos vértices para a ramificação, realizada pela função VÉRTICE_SATURAÇÃO, o vértice escolhido seguirá as regras estabelecidas acima.

Proposição 9. *Se $\Gamma(G') = \chi(G') = \chi(G)$ (em particular, se G' é crítico) então os subproblemas em que algum vértice de $G - G'$ é colorido são podados.*

Prova: Sejam $(H_1, \text{OP}_1, \text{ESC}, \mathcal{GR})$ os componentes de uma busca DSATUR, onde $U = \Gamma(G')$ é o valor da melhor solução viável do problema encontrada. Por contradição, vamos supor um subproblema $\sigma \in H_1$ em que o vértice $v \in V(G - G')$ é colorido. Como $\chi(G') = \chi(G)$ sabemos que para todo vértice $u \in V(G')$ foi atribuída uma cor pertencente a $\{1, \ldots, \chi(G)\}$ durante uma iteração preliminar da busca. Sabemos também, pela definição do método de busca DSATUR (Seção 5.1), que o limite inferior l_σ de um subproblema é igual ao número de cores distintas utilizadas na coloração parcial do subproblema. Contudo, esta afirmação contraria a hipótese de que $U = \chi(G)$ e a definição do método da busca genérica (Seção 3.3) que estabelece que para todo problema $\sigma \in H_1$ temos que $l_\sigma < U$. □

Vemos também que, se $\chi(G') < \chi(G)$, então o algoritmo não será capaz de podar todos os subproblemas em que os vértices de $G - G'$ são coloridos. Logo, o algoritmo continuará seu processo de enumeração de cores sobre estes vértices restantes, não sendo necessária uma fase crescente do método do subgrafo crítico (Seção 5.2) que iria iniciar um novo processo de enumeração sobre o subgrafo. Esta abordagem é mais interessante quando o número de vértices do subgrafo crítico é próximo do número de vértices do grafo original.

6.2. Método da coloração amigável

A relaxação da formulação CI (Seção 5.3) possui a capacidade de gerar limites inferiores muito próximos ao valor do número cromático do grafo, porém, a necessidade de resolvê-lo a cada iteração do método de geração de colunas constitui num grande custo computacional. Uma abordagem para tentar diminuir a complexidade do algoritmo seria aplicar a relaxação CIR sobre um subgrafo G' do grafo original, na esperança que este

novo modelo possua menos variáveis a se considerar.

Mais precisamente, seja $\mathcal{S}' = \{s_1^{'}, \ldots, s_{|\mathcal{S}'|}^{'}\}$ a família de todos os conjuntos independentes maximais de G'. Denominamos CIRR a formulação apresentada a seguir:

$$\begin{array}{lllll} \text{CIRR} & \text{Minimizar} & \sum_{i:\mathcal{S}'_i \in \mathcal{S}'} x_i & & \\ & \text{Sujeito a} & \sum_{i:v \in \mathcal{S}'_i} x_i & \geq 1 & \forall v \in V' \\ & & x_i & \geq 0 & \forall i \in \mathcal{S}' \end{array}$$

Vemos que CIRR é uma relaxação linear de CI, levando, portanto, a um limite inferior para o problema. Porém, como o número cromático do subgrafo G' já é um limite inferior para o número cromático de G, temos que o limite inferior fornecido por CIRR é de pior qualidade.

Este fato motiva a segunda formulação apresentada a seguir. Denominamos como CIRA a relaxação linear de CI adicionando à CIRR as inequações (4.7) e (4.8) definidas anteriormente.

$$\begin{array}{lllll} \text{CIRA} & \text{Minimizar} & \sum_{i:\mathcal{S}'_i \in \mathcal{S}'} x_i & & \\ & \text{Sujeito a} & \sum_{i:v \in \mathcal{S}'_i} x_i & \geq 1 & \forall v \in V' \quad \textbf{(1)} \\ & & x_i & \geq 0 & \forall i \in \mathcal{S}' \quad \textbf{(2)} \\ & & \sum_{i:\mathcal{S}'_i \text{ é u-extensível}} x_i & \geq 1 & \forall u \in V - V' \quad \textbf{(3)} \\ & & \sum_{i:\mathcal{S}'_i \text{ é } (v,w)\text{-extensível}} x_i & \geq 2 & \forall (v,w) \in A(V - V') \quad \textbf{(4)} \end{array}$$

Vemos que quando $G' = G$, somente as restrições do tipo **(1)** são consideradas, transformando o PPL em um problema CIR (Seção 5.3). Vemos também que a formulação CIRA leva a limites inferiores piores que CIR por se basear também no conceito de subgrafo. Porém, as restrições adicionais conseguem estreitar um pouco a diferença com a relaxação CIR e melhorar o limite inferior em relação a CIRR. Além disso, o conjunto de colunas obtido pela solução de CIRA torna-se um conjunto de colunas iniciais para CIR (Seção 6.2.3).

O procedimento de geração de colunas para o problema CIRA começa com um conjunto de colunas iniciais $S^{'}$. A partir desta solução inicial, solucionamos o problema CIRA

restrito a $s \in S^{'}$. Agora, devemos determinar se a solução corrente ótima de CIRA restrita a $S^{'}$ é também ótima para CIRA ou se seria vantajoso expandir o conjunto $S^{'}$. Na próxima seção será apresentado um procedimento para construir um conjunto de colunas iniciais para o problema CIRA.

6.2.1. Subgrafo amigável e colunas iniciais

Seja $G = (V, A)$ um grafo e G' um subgrafo induzido $\Gamma-$crítico de G construído pela função Criação_Grafo_Crítico. A idéia para o procedimento de obtenção das colunas iniciais para o problema CIRA é usar o subgrafo $\Gamma-$crítico como uma aproximação inicial para o subgrafo amigável. Observe que o subgrafo G' pode nos levar a um problema de programação linear CIRA inviável. Isto ocorre quando uma ou mais inequações do tipo vértices ou arestas não podem ser satisfeitas. Neste caso, temos que o sugrafo G' não é $\chi(G)-$crítico de G pois $\chi(G) > \chi(G')$. Vemos também que G' não é um grafo amigável de G. Se G' não é amigável, ele é atualizado durante o processo de geração de colunas iniciais. Se continuar não amigável, a correção será feita ao final da geração de colunas (Seção 6.2.3).

O primeiro passo para gerar um conjunto inicial de colunas para o problema CIRA é gerar uma coloração gulosa de G' pela heurística Γ, denotada por $(s_1, s_2, \ldots, s_{\Gamma(G')})$, onde $s_i \subseteq V$, para $i = 1 \ldots \Gamma(G')$, é um conjunto independente de G'. Esta coloração dará origem às colunas iniciais de CIRA. Claramente vemos que esta família de colunas iniciais satisfaz às inequações do problema original CIR sobre G'. Agora é preciso verficar se todas as restrições do tipo vértice e aresta associadas a $V - V'$ são cobertas por estas colunas.

Suponha que exista uma restrição do tipo vértice associada a um vértice $u \in V - V'$ que não foi coberto. É necessário incluir uma nova coluna (conjunto independente) $s^{'}$ de G' tal que

1) $s^{'}$ é $u-$extensível.

2) $s^{'}$ é maximal com respeito a **1)**.

Vemos que o conjunto independente $s^{'}$ não é necessariamente um conjunto independente maximal de G' pois pode existir um vértice de V' que não seja adjacente aos vértices de $s^{'}$ porém não é $u-$extensível. Se $s^{'} \neq \emptyset$, então devemos incluir este novo conjunto independente no conjunto das colunas inicias de G'. Se $s^{'} = \emptyset$ é porque, para todo vértice $v \in V'$, temos que $v \in Adj(u)$ em G. Logo vemos que o subgrafo G' não é $\chi(G)-$crítico pois ao colorirmos G' com $\Gamma(G')$ cores iremos precisar da cor $\Gamma(G') + 1$ para colorir o vértice u. Vemos também pela Propriedade 1 que G' não é um grafo amigável de G pois não podemos extender uma $\chi(G')-$coloração de G' para G. Neste caso, incluimos o vértice u em G' na tentativa de torná-lo $\chi(G)-$crítico, e incluimos um conjunto independente $s^{'} = \{u\}$ no conjunto de colunas iniciais de G' com o objetivo de viabilizar a restrição em **(3)**, equivalente ao vértice u.

Analogamente suponha agora que exista uma restrição do tipo aresta associada a uma aresta $uv \in A(V - V')$ que não foi coberta pelo conjunto de colunas corrente. Neste ponto do procedimento, todas as restrições do tipo vértices foram cobertas pela etapa anterior, logo se $s^{'}$ e $s^{''}$ são os conjuntos independentes de G' que cobrem as restrições de vértices equivalentes aos vértices u e v temos por definição que $s^{'}$ e $s^{''}$ são $uv-$extensíveis e poderiam satisfazer a restrição equivalente á aresta uv. Como a restrição não foi satisfeita, concluímos que $s^{'}$ e $s^{''}$ são idênticos. Logo deve-se gerar os conjuntos independentes $s_2^{'}$ de $G^{'}(V' - s^{'})$ e $s_2^{''}$ de $G^{'}(V' - s^{''})$ tal que

1) $s_2^{'}$ é $u-$extensível e $s_2^{'} \subseteq (V' - s^{'})$.

2) $s_2^{''}$ é $v-$extensível e $s_2^{''} \subseteq (V' - s^{''})$.

3) $s_2^{'}$ e $s_2^{''}$ são maximais com respeito a **1)** e **2)**, respectivamente.

Se $s_2^{'} \neq \emptyset$ ou $s_2^{''} \neq \emptyset$, então incluímos um dos dois conjuntos independentes (não vazio) no conjunto de colunas iniciais do problema com o objetivo de satisfazer a restrição da aresta uv. Caso contrário, temos que $(V' - s^{'}) \subseteq Adj(u)$ e $(V' - s^{''}) \subseteq Adj(v)$. Logo,

devemos adicionar os vértices u ou v à G' e incorporar as colunas correspondestes aos conjuntos independentes $\{u\}$ e $\{v\}$ ao conjunto inicial de colunas.

O algoritmo pára quando todas as restrições de vértices e arestas estão cobertas. No final do procedimento de obtenção das colunas iniciais para o problema CIRA, temos que G' pode não ser um subgrafo amigável de G pois pode ainda não ser possível expandir uma coloração de G' para uma coloração de G. A solução deste problema é apresentada na Seção 6.2.3.

6.2.2. O problema do conjunto independente máximo ponderado alterado

Assim como na formulação CI, se o menor custo reduzido for não negativo, a solução corrente é ótima para CIRA (Teorema 2). Caso contrário, devemos realizar uma troca de colunas e gerarmos um melhoramento na função objetivo. Nesta formulação, assumimos que G' é um grafo amigável de G. Logo, temos o acréscimo das restrições de vértices e arestas no problema. Vamos definir $CIPA(G')$ como o *problema de decisão do conjunto independente ponderado alterado* de G' onde o peso do conjunto independente s_i é calculado pelo seguinte somatório:

$$\pi^T z_i^s + \sum_{u \in V - V' | s_i \text{ é } u\text{-extensível}} \pi_u^T + \sum_{(uw) \in A(V-V') | s_i \text{ é } (u,w)\text{-extensível}} \pi_{(u,w)}^T \qquad (6.1)$$

Sendo π, π_u e $\pi_{(u,w)}$ os mutiplicadores duais relacionados as restrições **(1)**, **(3)** e **(4)** respectivavente.

Decidir se existe uma coluna associada a um conjunto independente que irá promover uma melhora na função objetivo corresponde a decidir se existe um $CIPA(G') > 1$.

A estratégia para encontrar um conjunto independente maximal s que satisfaça o problema $CIPA(G')$ utiliza a seguinte heurística gulosa (Figura 6.1). Comece com $s = \emptyset$ e defina $EXT(s, V') = \{v \in V' - s \mid s \text{ é } v\text{-extensível}\}$ e $EXT(s, V - V') = \{v \in V - V' \mid s \text{ é } v\text{-extensível}\}$. O algoritmo executa iterativamente até $EXT(s, V') = \emptyset$. Inicialmente,

atribui-se $EXT(s, V') = V'$ e $EXT(s, V - V') = V - V'$. O valor do peso de s, denotado por $\omega(s)$, é calculado por

```
1.   Algoritmo: HEURÍSTICA_GULOSA_CIPMA
2.   Entrada Conjunto de Vértices V'
3.   Saida  Conjunto Independente Maximal s e seu peso ω(s)
4.
5.     EXT(s,V') = V';
6.     EXT(s,V - V') = V - V';
7.     Enquanto EXT(s,V') ≠ ∅ Faça
8.       v = MAXI_PESO(EXT(s,V'));                    equação (6.3)
9.       EXT(s,V') = EXT(s,V') - {v};
10.      s = s + {v};
11.      ATUALIZA(EXT(s,V - V'));
```

Figura 6.1. *Heurística gulosa para $CIPMA$*

$$\omega(s) = \sum_{v \in s} \pi_v^T + \sum_{u \in EXT(s,V-V')} \pi_u^T + \sum_{(uw) \in A | u \in EXT(s,V-V') \text{ ou } w \in EXT(s,V-V')} \pi_{(u,w)}^T \quad (6.2)$$

Para cada vértice v em $EXT(s, V')$, seu peso será

$$\pi_v^T - \left(\sum_{(vu) \in A(EXT(s,V-V'))} \pi_u^T + \sum_{(uw) \in A | (vu) \in A(EXT(s,V-V')) \text{ ou } (vw) \in A(EXT(s,V-V'))} \pi_{(u,w)}^T \right) (6.3)$$

Vemos que o peso do vértice v no conjunto $EXT(s, V')$ leva em consideração o valor dos multiplicadores duais das restrições de vértices e arestas que não serão mais extensíveis ao conjunto s devido a presença do vértice v. Esta quantidade corresponde ao valor adicionado ao peso do conjunto s se v for for incluído nele.

Iterativamente, o algoritmo escolhe um vértice v em $EXT(s, V')$ tal que v tenha peso máximo em relação a Equação (6.3). Remove este vértice de $EXT(s, V')$, e o adiciona a s. No final deste procedimento, s será um conjunto independente maximal de G' e seu peso será dado por $\omega(s)$ (Equação (6.2)).

6.2.3. Resolução de CIR

Pela construção das colunas inicias do problema CIRA vemos que o subgrafo G' pode não ser amigável, isto é, a coloração independente maximal de G' obtida pela solução de CIRA pode não ser extensível para G. Com o objetivo de alcançar uma coloração independente maximal para G e conseguir um limite inferior melhor para o problema, solucionamos o problema CIR utilizando um conjunto de colunas iniciais geradas a partir da solução ótima de CIRA na esperança de que este conjunto de colunas inicias de CIR esteja próximo da solução ótima do problema. Para cada conjunto independente $s_i^{'}$ de G' correspondente a uma variável básica x_i, defina G_i como subgrafo de $G(V - V')$ induzido pelo conjunto de vértices $\{v \in V - V' \mid s_i^{'}$ é $v-$extensível $\}$. Agora aplique uma heurística gulosa para construir uma coloração independente maximal de G_i denotada por $(C_1^i, C_2^i, \ldots, C_k^i)$.

Propriedade 2. *$s_i^{'} \cup C_j^i$ é um conjunto independente maximal de G, para $j = 1, \ldots, k$*

Prova: Primerio vemos que $s_i^{'}$ é $v-$extensível $\forall v \in C_j^i$. Logo, vemos, por definição, que $s_i^{'} \cup C_j^i$ é um conjunto independente. Segundo, sabemos que $s_i^{'}$ é um conjunto independente maximal de G'. Por contradição, vamos supor que exista um vértice $v_j \in V - \{s_i^{'} \cup C_j^i\}$ tal que $s_i^{'} \cup C_j^i$ é v_j-extensível. Vemos então que, para todo $u \in C_j^i$, temos que $u \notin Adj(v_j)$. Logo, o vértice u poderia receber a cor j e ainda assim teríamos uma $k-$coloração de G, o que contradiz o fato de C_j^i ser um conjunto independente maximal. □

Para finalizar, gere todas as colunas correspondentes aos conjuntos independentes maximais s_i^j formados pela união do conjunto $s_i^{'}$ e um conjunto independente maximal da coloração C_j^i de G_i para todo $j = 1 \ldots k$ (Propriedade 2). Após a geração das colunas iniciais de CIR, o problema é solucionado seguindo os moldes descritos na Seção 5.3.

6.2.4. Método de busca para coloração amigável

O método de busca para a coloração amigável é definido como um método de busca *branch-and-bound* onde para cada subproblema $\sigma \in H_1$ temos que a prioridade de σ é dada pela média aritimética entre o valor de limite inferior e o valor do limite superior, isto é, $\textsc{Prio}(\sigma) = \frac{l_\sigma + u_\sigma}{2}$. O valor do limite superior u_σ é determinado pela heurística Γ enquanto que a função Geração_de_Colunas_Amigável é responsável pela resolução dos problemas CIRA e CIR, detalhado nas seções anteriores, e determinar o valor do limite inferior l_σ. O algoritmo do método de busca para coloração amigável é ilustrado pela Figura 6.2.

```
1.   Algoritmo: Op_1
2.
3.     σ ← H_1.Seleção;
4.     u_σ = Γ(p_σ);
5.     G' = Criação_Grafo_Crítico(p_σ, u_σ);
6.     l_σ = Geração_de_Colunas_Amigável(G', p_σ);
7.     Se (x_s inteiro ∀x_s ∈ Variáveis_Básicas(CIR)) Então
8.       Se l_σ ≤ U Então
9.         x_u ← σ;
10.        U ← l_σ;
11.        Atualização;
12.
13.    Ramificação(σ, σ^1, σ^2)
14.    H_1.Inclusão(σ^1);
15.    H_1.Inclusão(σ^2);
```

Figura 6.2. *Operador do método coloração amigável.*

7

Resultados Computacionais

Durante a elaboração desta dissertação foram implementados os novos algorimos propostos no Capítulo 6. Na Seção 7.1 descreveremos brevemente as classes de grafos que compoem o conjunto de instâncias de teste extraídas de DIMACS [14]. Na Seção 7.2 serão apresentados os detalhes de implementação referentes a estes algoritmos. A seguir, na Seção 7.3, apresentaremos os resultados e análises do método DSATUR crítico (Seção 6.1) e a comparação entre o método da geração de colunas original e o método da coloração amigável apresentado na Seção 6.2.

7.1. Descrição das instâncias

Em nossos experimentos computacionais foram utilizadas instâncias extraídas do conjunto de problemas de coloração de grafos de DIMACS [14]. O objetivo dos experimentos foi determinar a robustez dos novos algoritmos propostos. Aqui nós iremos rapidamente descrever estas classes de grafos.

Grafos Aleatórios $\mathcal{G}_{n,p}$ são grafos com n vértices onde há uma probabilidade p de existir uma aresta entre quaisquer par de vértices, independente da existência ou não de outra aresta. Foram realizados testes em grafos aleatórios tendo $n = 50, 60, 70$ e uma

probabilidade de aresta $p = 0.5, 0.7, 0.9$. Foi gerado aleatoriamente um conjunto de 10 grafos $\mathcal{G}_{n,p}$ para cada par (n, p) e uma média de seus resultados é mostrada nas Tabelas 7.3 e 7.1.

Grafos de Registros Uma aplicação muito utilizada para coloração em grafos é o problema de alocação de registros compartilhados por uma compilação dado um código seqüencial. Os vértices nestes grafos representam variáveis e dois vértices são conectados por uma aresta se somente se as duas variáveis correspondentes a estes vértices utilizam o mesmo registro ao mesmo tempo em um fragmento de código. O número cromático destes grafos indicam o número mínimo de registros necessários naquele fragmento de código. **mulsol.i.x, inithx.i.x, zeroin.i.x** são grafos de registros (o caractere **x** é um número usado para diferenciar as instâncias).

Grafos Rainha Dado um tabuleiro de xadrez q por r, um grafo rainha **queen.q_r** é um grafo com qr vértices cada um correspondendo a um quadrado do tabuleiro, e uma aresta existe se somente se conecta dois quadrados na mesma linha, coluna ou diagonal. Esta classe de grafos representa o problema de decidir a questão: Existe uma maneira de se colocar q conjuntos de r rainhas num tabuleiro q por r onde rainhas de um mesmo conjunto não possam se atacar? A resposta é positiva se somente se o grafo correspondente ao problema tiver número cromático igual a r.

Grafos de Mycielski Dado um grafo $G = (V, A)$ onde $n = \mid V \mid$, definimos a transformação de Mycielski [40] como o processo de obtenção do grafo MY_G de G onde

$$V(MY_G) = \{x_1, x_2, \ldots, x_n, y_1, y_2, \ldots, y_n, z\}.$$

O conjunto de arestas $A(MY_G)$ é formado pelas arestas $x_i x_j$ para todo $v_i v_j \in A(G)$, $x_i y_j$ para todo $v_i v_j \in A(G)$ e $y_i z$ para todo i. Sabemos que esta transformação não afeta o tamanho da maior clique do grafo e incrementa o número cromático em uma unidade [41]. Definimos grafos de Mycielski, denotados por **mycielx**, como o conjunto de **x** trasformações de Mycielski sucessivas, a partir de um grafo com dois

vértices e uma aresta. Esta classe de grafos parece ser difícel de se resolver pois não possui triângulos (a clique máxima é 2) mas possui número cromático igual a $\mathbf{x}$+1.

7.2. Detalhes de implementação

- Os algoritmos aqui apresentados foram implementados num microcomputador Pentium III 800 MHz com 128 MBytes de memória usando a biblioteca CPLEX 7.1 para resolver problemas de programação linear.

- Uma heurística TABU sugerida por Hertz e Werra [32] e descrita na Seção 4.3.2 é utilizada na geração de uma coloração viável Γ e na construção de grafos $\Gamma-$críticos.

- Algumas melhorias sugeridas em [15] foram implementadas a respeito de estratégias de ramificação e geração de boas colunas para o problema.

- Durante os experimentos foi verificado que a tarefa de encontrar colunas úteis, isto é, que promovam a melhoria da função objetivo, se tornava difícil á medida em que o método da geração de colunas transcorria. A partir de um determinado ponto, a heurística gulosa para $CIPA$ não conseguia encontrar boas colunas, sendo necessária a utilização do algoritmo recursivo. Também foi verificado que o ganho efetivo na função objetivo promovido pelas últimas colunas geradas não compensava seu custo. Por estas razões, o processo de geração de colunas é parado para um problema CIRA(G'), para cada grafo G em dois casos: quando o peso da última coluna gerada for menor do que 1.1; ou quando o peso da última coluna gerada for menor do que 1.2 e o valor da solução corrente do problema CIRA(G') é menor do que 105% do valor de limite inferior do problema pai de G. Foi observado que quando o peso de uma coluna gerada atinge um valor menor que 1.1, as próximas colunas geradas pouco têm a contribuir para o valor da função objetivo ou para o conjunto de colunas da solução de CIRA. Quando a coluna gerada possui peso menor que 1.2 existe a possibilidade de alcançar um conjunto de colunas iniciais melhor para CIR, porém, verificamos que quando o valor da função objetivo do

problema CIRA fica menor que o limite inferior do problema pai de G, podemos gerar colunas muito ruins para CIR.

- Foi verificado também que este critério de parada tende a aumentar ligeiramente o número de colunas geradas para o problema CIR(G). Porém verificamos uma queda considerável no número de colunas geradas para CIRA(G'). Foi experimentado alterar o valor do critério de parada para um valor superior (na tentativa de encontrar um bom conjunto de colunas iniciais para CIR(G) em um menor tempo), o que conduziu a resultados ruins pois enquanto que uma solução para CIRA(G') era encontrada rapidamente, o número de colunas geradas para CIR(G) aumentava drasticamente. Foi utilizada a heurística gulosa Clique_Gulosa descrita na Seção 5.3.3 para aproximar a clique máxima em G e utilizar este valor como limite inferior do pai de G.

- Observamos que o custo de atualizar o peso dos vértices de $EXT(s, V')$ (Equação 6.3) iterativamente durante o procedimento de solução do problema $CIPA$ era alto. Logo, com o objetivo de tornar o procedimento mais rápido, decidimos calcular os pesos dos vértices pertencentes a $EXT(s, V')$ apenas no começo do procedimento. Esta abordagem demonstrou um aumento na velocidade do método com uma pequena perda na qualidade das colunas geradas.

- Devido ao custo de se solucionar os problemas CIRA e CIR a cada iteração, foi tentado solucionar apenas o modelo baseado no grafo $\Gamma-$crítico (CIRR), sem as restrições de vértices e arestas. Os resultados foram ruins pois os limites inferiores encontrados em cada subproblema eram de má qualidade.

- Foi tentado, também, diminuir o número de restrições do modelo CIRA através do seguinte procedimento. Iterativamente, eram incluídas restrições de vértices e arestas no modelo, à medida em que elas se tornavam violadas. Esta abordagem se mostrou ineficiente porque no final do procedimento, todas as restrições eram incluídas. Portanto, observamos que seria melhor incluí-las no início de uma vez só.

- Com o objetivo de diminuir o custo da geração de grafos $\Gamma-$críticos a cada subproblema, foi tentado manter um conjunto global de vértices pertencentes ao grafo crítico de todos os subproblemas. Entretanto, esta abordagem também se mostrou ruim devido às diferenças estruturais dos subproblemas.

7.3. Sumário de resultados

Os resultados das colorações sobre as intâncias descritas anteriormente são apresentados nas tabelas a seguir. Para cada grafo G são apresentadas as seguintes informações :

- Tamanho: informação relativa ao número de vértices e ao número de arestas da instância.
- TABU: informação relativa ao limite superior do número cromático do grafo fornecido pela heurística TABU.
- LP: informação relativa ao limite inferior do número cromático do grafo. É denotado por LP o valor da solução ótima do problema CIR(G) onde G é o grafo original. As instâncias que não possuem valor neste campo, foram solucionadas diretamente sem a necessidade de gerar colunas, isto é, o valor do limite superior (TABU) é igual ao valor da maior clique maximal encontrada (Clique_Gulosa).
- $\chi(G)$: representa o valor da coloração ótima do grafo.
- Nós: informa o número de subproblemas explorados durante a busca. Um valor em branco neste campo, indica que o problema não pôde ser resolvido dentro do intervalo de tempo limite (24 horas).
- *CIR*: informações relativas ao método de geração de colunas. Denotamos de $Tempo$ o tempo em minutos para a resolução da instância, denotaremos de Col o número total de colunas geradas em toda a árvore de ramificação para solucionar o problema CIR.

- *CIRA_CIR*: informações relativas ao método de coloração amigável. Similarmente ao anterior, a variável *Tempo* listará o tempo em minutos para a resolução da instância. Denotaremos de $G' - Col$ e $G - Col$ o número total de colunas geradas em toda a árvore de ramificação para solucionar o problema CIRA e CIR respectivamente.

- $DSATUR_{crit}$: informações relativas ao método DSATUR crítico. Novamente, a variável *Tempo* designará o tempo em minutos para a resolução da instância.

- $Tam_{G'}$: número de vértices do subgrafo crítico gerado pelo método DSATUR crítico.

- $Des_{G'}$: média do número de vértices coloridos do subgrafo G' em cada subproblema descartado pelo limite superior do método DSATUR crítico.

- $Des_{G-G'}$: média do número de vértices coloridos em $G - G'$ em cada subproblema descartado pelo limite superior do método DSATUR crítico.

7.3.1. DSATUR crítico

Observamos que o método DSATUR crítico possui uma grande velocidade na análise de seus subproblemas, porém, a quantidade de subproblemas analisados impossibilita a solução de instâncias de grande porte. Este grande número de subproblemas é gerado devido aos limites inferiores pobres que são fornecidos pelo método que impossibilitam um processo de poda eficiente.

Observamos pela Tabela 7.1 que o método DSATUR crítico teve uma dificuldade maior na resolução dos grafos aleatórios com probabilidade de aresta de 70%, enquanto que os grafos aleatórios com probabilidade de aresta de 90 % foram solucionados rapidamente. Este comportamento se deve à alta densidade de arestas dos grafos com $p = 0.9$ pois estes grafos possuem um valor Δ elevado o que tende a gerar menos colorações viávies durante o processo de ramificação do método. Logo, os grafos menos densos ($p = 0.5, 0.7$) tendem a gerar mais subproblemas. Podemos verificar também, que o número de problemas analisados pelos grafos aleatórios com $p = 0.5$ é menor do que os grafos com $p = 0.7$. Isto

ocorre devido ao valor pequeno do número cromático dos grafos aleatórios com $p = 0.5$ que conseguem alcançar mais rapidamente esta coloração e provar sua otimalidade.

Pelas colunas $Des_{G'}$ e $Des_{G-G'}$ das Tabelas 7.1 e 7.2 podemos verificar que os subproblemas descartados pelo limite superior do método DSATUR crítico possuem a seguinte característica: os vértices que foram coloridos durante a enumeração implícita pertencem, na maioria absoluta, aos subgrafos críticos gerados pelo método. Percebemos que o tempo de solucionar uma instância do problema de coloração pelo método DSATUR crítico é dado pelo tempo de solucionar o subgrafo crítico G' desta mesma instância utilizando o método DSATUR. Vemos também, pela coluna $Tam_{G'}$, que os tamanhos dos subgrafos críticos gerados são, na sua maioria, menores que os tamanhos dos grafos originais, portanto, é conseguida uma diminuição sobre o tamanho das instâncias a serem solucionadas.

Grafo	Tamanho		TABU	$\chi(G)$	Nós	$DSATUR_{crit}$	$Tam_{G'}$	$Des_{G'}$	$Des_{G-G'}$
	n	m				$Tempo$			
$\mathcal{G}_{50,.5}$	50	600.2	9.4	9.4	22071.1	0.799	39.1	22.37	0.00
$\mathcal{G}_{50,.7}$	50	814.6	14.2	14.2	37013.3	1.270	41.6	26.91	0.00
$\mathcal{G}_{50,.9}$	50	1096.8	22.2	22.2	425.4	0.805	42.7	32.88	0.13
$\mathcal{G}_{60,.5}$	60	960.2	10.2	10.2	1023516.4	18.468	48.2	26.90	0.00
$\mathcal{G}_{60,.7}$	60	1236.2	14.8	15.0	3771985.3	90.361	51	31.61	0.00
$\mathcal{G}_{60,.9}$	60	1604.4	25.4	25.4	10176.22	1.733	55.3	42.60	0.07
$\mathcal{G}_{70,.5}$	70	1218.6	11.6	11.6	-	-	-	-	-
$\mathcal{G}_{70,.7}$	70	1678.6	16.2	17.0	-	-	-	-	-
$\mathcal{G}_{70,.9}$	70	2173.6	28.1	28.6	601275.75	33.574	64.66	47.27	0.04

Tabela 7.1. *Sumário de resultados dos grafos aleatórios do método DSATUR crítico.*

7.3.2. Coloração Amigável

A partir das Tabelas 7.3 e 7.4 observamos que:

1. O número de subproblemas analisados pelos métodos baseados em geração de colunas é bem menor que o número de problemas analisados pelo método DSATUR crítico. Este comportamento é explicado pela alta qualidade dos limites inferiores conseguidos pelos métodos baseados em geração de colunas, entretanto seu custo

Grafo	Tamanho		TABU	$\chi(G)$	Nós	$DSATUR_{crit}$	$Tam_{G'}$	$Des_{G'}$	$Des_{G-G'}$
	n	m				$Tempo$			
mulsol.i.1	197	3925	49	49	1	3.65	49	49	0
mulsol.i.2	188	3885	31	31	1	1.57	106	49	0
mulsol.i.3	184	3916	31	31	1	1.60	107	31	0
mulsol.i.4	185	3946	31	31	1	1.97	101	31	0
mulsol.i.5	186	3973	31	31	1	1.70	31	31	0
inithx.i.1	864	18707	54	54	1	63.721	54	54	0
inithx.i.2	645	13979	31	31	1	34.05	31	31	0
inithx.i.3	621	13969	31	31	1	27.07	31	31	0
zeroin.i.1	211	4100	49	49	1	3.82	49	49	0
zeroin.i.2	211	3541	30	30	1	2.58	30	30	0
zeroin.i.3	206	3540	30	30	1	2.53	30	30	0
queen.6_6	36	290	7	7	1156	0.098	28	17.86	0.03
queen.7_7	49	476	7	7	1008	0.407	10	7	0
queen.8_8	64	728	9	9	4517943	60.693	53	29.22	0
queen.9_9	81	2112	10	10	-	-	-	-	-
queen.8_12	96	1368	12	12	1	0.70	12	12	0
myciel3	11	20	4	4	27	0.0	11	7.93	0
myciel4	23	71	5	5	8728	0.017	23	15.83	0
myciel5	47	236	6	6	-	-	-	-	-

Tabela 7.2. *Sumário de resultados de algumas instâncias de DIMACS para o método DSATUR crítico.*

computacional é bem mais elevado. Observamos pela coluna LP nas Tabelas 7.3 e 7.4 que os métodos *CIR* e *CIRA_CIR* obtêm valores de limite inferior muito próximos do valor do número cromático das instâncias.

2. O número de colunas necessárias para alcançar a solução em *CIR* é bem maior que o número de colunas geradas por *CIRA_CIR* na resolução do modelo CIR. Isto ocorre devido à qualidade superior do conjunto de colunas iniciais fornecidas pelo modelo CIRA no método *CIRA_CIR*.

Em comparação com *CIR*, o método *CIRA_CIR* apresentou melhores resultados quando o número de colunas necessário para solucionar o problema é grande, isto é, o método *CIRA_CIR* se mostrou melhor que *CIR* nas instâncias em que o procedimento de geração de colunas teve mais dificuldade. Vemos que para grafos aleatórios (Tabela 7.3) com probabilidade de aresta de 50%, o método *CIRA_CIR* solucionou em um tempo menor

Grafo	Tamanho		TABU	LP	$\chi(G)$	Nós	*CIR*		*CIRA_CIR*		
	n	m					$Tempo$	Col	$Tempo$	$G' - Col$	$G - Col$
$\mathcal{G}_{50,.5}$	50	600.2	9.4	8.6	9.4	45.8	3.0	996.2	2.4	419.9	359.3
$\mathcal{G}_{50,.7}$	50	814.6	14.2	13.5	14.2	8.0	0.9	312.9	1.2	313.4	202.5
$\mathcal{G}_{50,.9}$	50	1096.8	22.2	21.9	22.2	1.4	0.1	51.2	0.4	48.4	32.8
$\mathcal{G}_{60,.5}$	60	960.2	10.2	9.2	10.2	39.6	5.4	902.3	3.9	428.3	364.0
$\mathcal{G}_{60,.7}$	60	1236.2	14.8	15.0	15.0	10.6	2.5	398.3	2.8	311.1	207.2
$\mathcal{G}_{60,.9}$	60	1604.4	25.4	24.9	25.4	12.1	1.5	47.1	2.1	34.0	48.4
$\mathcal{G}_{70,.5}$	70	1218.6	11.6	10.7	11.6	44.4	10.1	921.1	9.0	419.6	374.7
$\mathcal{G}_{70,.7}$	70	1678.6	16.2	16.2	17.0	35.0	8.5	390.8	8.9	316.8	210.2
$\mathcal{G}_{70,.9}$	70	2173.6	28.1	28.1	28.6	7.2	5.9	49.4	6.8	41.2	44.3

Tabela 7.3. *Sumário de resultados dos grafos aleatórios do método coloração amigável.*

que *CIR* as instâncias de $n = 50, 60, 70$, enquanto que nos grafos aleatórios mais densos ($p = 0.7, 0.9$), em que o procedimento de geração de colunas apresentou mais facilidade, o método *CIR* foi mais rápido. Vemos também na Tabela 7.4 que, para o grafo rainha **queen.6_6**, o método *CIRA_CIR* obtém um tempo pior que *CIR*, porém, á medida em que os grafos rainhas se tornam mais difíceis e precisam gerar mais colunas para sua solução, o método *CIRA_CIR* melhora seu rendimento, alcançando o mesmo tempo na instância **queen.8_8** e superando o o método *CIR* na instância **queen.9_9**.

O método *CIRA_CIR* consegue obter uma diminuição do tempo de resolução das instâncias de coloração, devido à sua capacidade de gerar um conjunto de boas colunas iniciais para o modelo CIR, porém, o custo computacional destas colunas é grande, tornando interessante sua utilização apenas em instâncias onde o método de geração de colunas encontra mais dificuldade.

Vemos que a eficiência do método *CIRA_CIR* depende de dois pontos principais: o tempo necessário para se gerar uma coluna no modelo CIRA, e o número de colunas geradas para alcançar a solução de CIR a partir do conjunto de colunas iniciais fornecido por CIRA. O primeiro ponto está relacionado com a solução do problema $CIPA$. Vemos que, pela concepção do problema $CIPA$, o cálculo do peso de um conjunto independente ponderado leva um tempo maior para ser realizado, entretanto, pelo problema $CIPA$ ser modelado sobre um subgrafo crítico, ele provavelmente possui menos conjuntos inde-

Grafo	Tamanho		TABU	LP	$\chi(G)$	Nós	*CIR*		*CIRA_CIR*		
	n	m					$Tempo$	Col	$Tempo$	$G' - Col$	$G - Col$
mulsol.i.1	197	3925	49	-	49	0	0.026	0	0.026	0	0
mulsol.i.2	188	3885	31	-	31	0	0.015	0	0.015	0	0
mulsol.i.3	184	3916	31	-	31	0	0.016	0	0.016	0	0
mulsol.i.4	185	3946	31	-	31	0	0.016	0	0.016	0	0
mulsol.i.5	186	3973	31	-	31	0	0.013	0	0.013	0	0
inithx.i.1	864	18707	54	-	54	0	0.193	0	0.193	0	0
inithx.i.2	645	13979	31	-	31	0	0.167	0	0.167	0	0
inithx.i.3	621	13969	31	-	31	0	0.154	0	0.154	0	0
zeroin.i.1	211	4100	49	-	49	0	0.025	0	0.025	0	0
zeroin.i.2	211	3541	30	-	30	0	0.012	0	0.012	0	0
zeroin.i.3	206	3540	30	-	30	0	0.012	0	0.012	0	0
queen.6_6	36	290	7	7	7	1	0.033	37	0.090	60	11
queen.7_7	49	476	7	-	7	0	0.004	0	0.004	0	0
queen.8_8	64	728	9	8.4	9	1	1.101	175	1.102	173	71
queen.9_9	81	2112	10	9	10	27	601.43	10796	461.09	4334	4913
queen.8_12	96	1368	12	-	12	0	0.009	0	0.009	0	0
myciel3	11	20	4	2.9	4	5	0.002	29	0.002	47	6
myciel4	23	71	5	3.2	5	1462	1.408	15926	2.882	8731	3275
myciel5	47	236	6	3.5	6	-	-	-	-	-	-

Tabela 7.4. *Sumário de resultados de algumas instâncias de DIMACS para o método coloração amigável.*

pendentes a considerar. Logo, o método *CIRA_CIR* terá vantagem sobre o método *CIR*, quando o número de conjuntos independentes que o problema CIP tiver que considerar, for grande o suficiente para que o tempo de resolução do problema $CIPA$ seja menor, mesmo com um cálculo de conjunto independente ponderado mais lento.

O segundo ponto está relacionado com a qualidade das colunas iniciais geradas pelo modelo CIRA. Vemos que quanto melhor a qualidade do conjunto de colunas iniciais, menor é o número de colunas necessárias para alcançar a solução ótima do modelo CIR. Vemos também que a qualidade das colunas iniciais está diretamente relacionada á semelhança entre os poliedros formados pelas soluções viáveis dos modelos CIRA e CIR. Esta característica estimula o estudo de cortes sobre o poliedro do modelo CIRA com o objetivo de aproximar sua estrutura ao poliedro do modelo CIR.

8

Conclusão

Durante a elaboração da dissertação foram desenvolvidos dois novos métodos apresentados nos capítulos anteriores. O primeiro, denominado de DSATUR crítico, demonstrou que o tempo de um processo de enumeração implícita de cores baseado em DSATUR sobre um grafo G é equivalente ao tempo de enumeração de cores sobre um subgrafo $\Gamma-$crítico G', gerado a partir de G. Observamos pelas colunas $Tam_{G'}$ e $Des_{G'}$ das Tabelas 7.1 e 7.2 que o número de vértices coloridos de G' dos subproblemas descartados durante a busca ainda é inferior ao tamanho do subgrafo $\Gamma-$crítico gerado. Então, vemos que ainda é possível estreitar a diferença entre o subgrafo $\Gamma-$crítico gerado e o subgrafo crítico. Isto poderia promover uma diminuição ainda maior na instância do problema.

O segundo método apresentado é baseado em uma nova formulação gerada a partir da construção de grafos críticos e conjuntos independentes maximais. Este novo método, denominado de coloração amigável, mostrou competitividade com o método de geração de colunas, sendo mais eficiente em instâncias onde o processo de geração de colunas encontrava mais dificuldade. Observamos também que, a possibilidade de conseguir limites inferiores de uma qualidade razoável, porém, com um baixo custo computacional através da solução dos modelos CIRR e CIRA, pode nos levar a bons procedimentos heurísticos para grafos que são muito grandes ou muito difíceis de serem solucionados.

Com o objetivo de aprimorar o método da coloração amigável, devemos buscar uma melhoria no tempo de resolução do problema do conjunto independente ponderado alterado ($CIPA$) para atingir colunas mais velozes para o modelo CIRA. Observamos também que o sucesso deste novo método está relacionado com a capacidade de gerar grafos críticos rapidamente a cada iteração do processo de geração de colunas.

Uma abordagem para tentar diminuir o tempo computacional poderia ser alcançada com uma paralelização para os novos métodos. Existe a perspectiva de implementação de uma versão paralela para a biblioteca PARADISE e explorar o paralelismo inerente de uma estrutura *branch-and-bound* sobre o método do DSATUR crítico, o que seria interessante devido ao grande número de subproblemas analisados durante sua enumeração implícita. Uma outra abordagem paralela que poderia vir a ser estudada seria a paralelização do processo de geração de colunas.

Foram realizados experimentos computacionais em grafos aleatórios e em grafos de DIMACS. Estes experimentos demonstraram que os novos métodos podem resolver instâncias de grafos de grande porte e ser competitivo com outros métodos aqui apresentados.

A

PARADISE

Durante o estudo e desenvolvimento da dissertação, foi utilizada uma biblioteca de funções construida sobre o modelo da busca genérica. Os elementos previamente descritos e agora implementados permitem diversas abordagens de busca. Neste apêndice iremos especificar suas principais rotinas de configuração e acesso, assim como suas sintaxes e parâmetros.

A PARADISE é uma biblioteca modular e de fácil configuração. Sua grande flexibilidade permite ao usuário configurar a abordagem de busca desejada através da passagem de funções e parâmetros que especificarão o modelo assim como o problema a ser abordado. Deixando transparente ao usuário apenas suas funções de acesso e configuração, a PARADISE torna-se um instrumento simples, prático e eficaz na resolução de problemas de otimização combinatória.

Para utilizar a biblioteca é necessário, primeiro adicionar em seus programas o arquivo *CPD.c*, com isso o usuário poderá utilizar as funções de configuração e acesso. Depois é necessário definir a família de conjuntos prioridades que serão utilizadas no processo. Em seguida, o usuário deve implementar os operadores que serão passados como parâmetros para a biblioteca e que servirão como guia durante a busca realizada sobre o espaço de solução. Por fim, deve-se configurar o modelo desejado através da especificação de "quando" e "onde" cada operador deverá agir, isto é realizado pelos escalonadores. Após

feito estas configurações, deve-se chamar a função CPD para iniciar o processo de busca.

A.1. Rotinas de configuração

SETBUFFER

- **Definição :** Função responsável por receber as funções de salvar, restaurar e trocar instâncias do problema para um buffer de caracteres e vice-versa. As funções devem receber como parâmetros uma instância do problema e um buffer de caracteres.
- **Sintaxe :** `int erro = setbuffer(int *sas, int *res, int *exs);`
- **Parâmetros :**

  sas - função para salvar uma instância do problema para um buffer.

  res - função para restaurar uma instância do problema de um buffer.

  exs - função para trocar dois problemas de posição.

SETSEARCH

- **Definição :** Principal função de configuração, responsável por receber as funções que irão direcionar a busca como limites inferior e superior, prioridade do problema e o escalonador de tarefas responsável por escolher o operador que irá atuar naquele momento.
- **Sintaxe :** `int erro = setsearch(int *l_sup, int *l_inf, int *prioridade, int *escalonador);`
- **Parâmetros :**

  l_sup - Função que receberá uma instância do problema e retornará um limite superior para ele.

  l_inf - Função que receberá uma instância do problema e retornará um limite inferior para ele.

prioridade - Função que receberá uma instância do problema e retornará sua prioridade.

escalonador - Função que escolherá qual operador deverá ser executado.

SETCPD

- **Definição :** Função responsável pela passagem de parâmetros que serão necessários para busca.
- **Sintaxe :** `int erro = setsub(int nproc, int tamanho);`
- **Parâmetros :**

tamanho - Tamanho máximo da instância do problema.

NEWCP

- **Definição :** Função responsável pela criação de um novo conjunto prioridade. Ela deve retornar um inteiro que será o índice desse conjunto prioridade.
- **Sintaxe :** `int numcp = newcp(int *comparação);`
- **Parâmetros :**

comparação - Função que receberá dois índices de problemas do conjunto e retornará o índice daquele de melhor prioridade.

BESTSOLCP

- **Definição :** Função responsável por receber a solução inicial (U, x_u) do problema, assim como, a função de comparação entre possíveis soluções melhores para um determinado conjunto prioridade.
- **Sintaxe :** `int erro = bestsolcp(int cp, int valor, ptgrafo instância, int *comparação);`

- **Parâmetros :**

  cp - Índice do conjunto prioridade.

  valor - Valor da melhor solução (U).

  instância - Instância da melhor solução (x_u).

  comparação - Função que irá comparar uma solução candidata com a melhor solução corrente daquele conjunto prioridade.

NEWOPERATOR

- **Definição :** Função responsável pela criação de um novo operador. Ela deve retornar um inteiro que será o índice desse operador.
- **Sintaxe :** `int numop = newoperator(char *nome, int *operador);`
- **Parâmetros :**

  nome - Identificação do operador.

  operador - Função de execução do operador. Esta função deve receber como entrada uma instância do problema e sua prioridade.

A.2. Rotinas de acesso

INSERTCP

- **Definição :** Função para inserir uma instância de um problema diretamente em um determinado conjunto prioridade.
- **Sintaxe :** `int erro = insertcp(int cp, ptgrafo instância);`
- **Parâmetros :**

  cp - Índice do conjunto prioridade.

instância - Instância do problema.

POSICAOCP

- **Definição :** Função que retorna uma posição livre na estrutura de um determinado conjunto prioridade.
- **Sintaxe :** `int pos = posicaocp(int cp);`
- **Parâmetros :**

  cp - Índice do conjunto prioridade.

  pos - Índice da posição livre

LIBERACP

- **Definição :** Função que libera um problema armazenado numa determinada posição
- **Sintaxe :** `void liberacp(int posição);`
- **Parâmetros :**

  posição - Índice da posição a ser liberada no conjunto prioridade indicado.

SELECAOCP

- **Definição :** Função que retorna a instância de maior prioridade de um determinado conjunto prioridade.
- **Sintaxe :** `int pos = selecaocp(int cp);`
- **Parâmetros :**

  cp - Índice do conjunto prioridade.

  pos - Índice da posição do problema de maior prioridade.

VAZIACP

- **Definição :** Função que checa se um determinado conjunto prioridade está vazio.
- **Sintaxe :** `int resultado = vaziacp(int cp);`
- **Parâmetros :**

  cp - Índice do conjunto prioridade.

  resultado - resultado da função

VAZIACCP

- **Definição :** Função que checa se todos os conjuntos prioridade estão vazios.
- **Sintaxe :** `int resultado = vaziaccp();`
- **Parâmetros :**

  resultado - resultado da função

CPD

- **Definição :** Função de inicialização do processo de busca genérica paralela.
- **Sintaxe :** `CPD(int prioridade, ptgrafo instância);`
- **Parâmetros :**

  prioridade - prioridade do problema inicial.

  instância - instância do problema inicial.

Referências Bibliográficas

[1] D. de Werra, "An introduction to timetabling," em *Eutopian Journal of Operations Research, Vol 19*, pp. 151–162, 1985.

[2] A. Gamst, "Some lower bounds for a class of frequency assignment problems," em *IEEE Transactions of Vehicular Technology, Vol 35, n. 1*, pp. 8–14, 1986.

[3] F. Chow, J. Hennessy, "Register allocation by priority-based coloring," in *Proceedings of ACM SIGPLAN 84 Symposium on Compiler Construction, New York, NY, USA*, pp. 222–232, 1984.

[4] F. Chow, J. Hennessy, "The priority-based coloring approach to register allocation," em *ACM Transactions on Programming Languages and Systems, Vol 12, n. 4*, pp. 501–536, 1990.

[5] Y. Saad, "Iterative methods for sparse linear systems," em *PWS Publishing Company, Boston, MA, USA*, 1996.

[6] R. M. Karp, "Reducibility among combinatorial problems," em *Complexity of Computations, Advances in Computing Research*, pp. 85–103, 1972.

[7] Michael R. Garey, David S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, 1979.

[8] Carsten Lund, Mihalis Yannakakis, "On the hardness of approximation minimization problems," em *Proceedings of 25th Annual ACM Symposium on Theory of Computing*, 1993.

[9] Marek Kubale, Boguslaw Jackowski, "A generalized implicit enumeration algorithm for graph coloring," em *Communications of the ACM 28/4*, pp. 400–412, 1985.

[10] R. J. Brown, "Chromatic scheduling abd the chromatic number problem," in *Management Science, Vol. 19, n. 4*, pp. 451–463, 1972.

[11] Daniel Brélaz, "New methods to color the vertices of a graph," em *Communications of the ACM, Vol. 22, n. 4*, 1979.

[12] S. M. Korman, "The graph-coloring problem," em *Combinatorial Optimization, pp. 211-235, Wiley, New York*, 1979.

[13] Avrim Blum, "New approximation algorithms for graph coloring," em *Journal of Association for Computing Machinery*, pp. 470–516, May 1994.

[14] D. Johnson, M. Trick, "Cliques, coloring and satisfiability: Second dimacs implementation challenge," em *Vol 26 of DIMACS Series in Discrete Mathematics and Theoretical Computer Science, American Mathematical Society*, 1996. Também encontrado em *http://mat.gsia.cmu.edu/COLOR/instances.html*.

[15] Anuj Mehrotra, Michal A. Trick, "A column generation approch for graph coloring," em *INFORMS Journal on Computing, Vol. 8, pp. 344-354*, 1996.

[16] Alain Hertz, Francine Herrman, "Finding the chromatic number by means of critical graphs," em *Research Report, Ecole Polytechnique Fédérale de Lausanne, Suiça*, 2000.

[17] J. A. Bondy, U. S. R. Murty, *Graphy theory with applications.* American Elsevier Publishing, 1976.

[18] Frank Harary, *Graphy Theory.* Addison-Wesley Publishing, 1969.

[19] Ben Noble, James W. Daniel, *Applied Linear Algebra.* Prentice Hall LTDA, 1977.

[20] Mokhtar S. Bazaraa, John J. Jarvis, Hanif D. Sherali, *Linear Programming and Network Flows.* John Wiley & Sons, 1977.

[21] Wayne L. Winston, *Operations Research: Applications and Algorithms.* International Thomson Publishing, 1994.

[22] Carlos E. Ferreira, Yoshico Wakabayashi, *Combinatória Poliédrica e Planos de Cortes Faciais.* X Escola de Computação, 1996.

[23] Christos H. Papadimitriou, Kenneth Steiglitz, *Combinatorial Optimization.* Dover Publications, 1998.

[24] L. G. Khachian, "A polynomial algorithm for linear programming," em *Doklady Akad. Nauk USSR, 244, n. 5*, 1979.

[25] V. Kumar, L. Kanal, "The CDP: a unifying formulation for heuristic search, dynamic programming, and branch-and-bound," em *National Conference on A.I.*, 83.

[26] D. Nau, V. Kumar, L. Kanal, "General branch and bound, and its relation to A* and AO*," em *Artificial Intelligence, Vol. 23*, pp. 29–58, 1984.

[27] R. Corrêa, *Recherche Arborescente Paralléle : de la Formulation Algorithmique aux Applications.* PhD thesis, Institut National Polytechnique de Grenoble, France, 1997.

[28] Ronald Graham, Martin Grötschel, László Lovász, *Handbook of Combinatorics.* Elsevier Science B. V., 1995.

[29] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, *Introduction to Algorithms.* McGraw-Hill Book Company, 1996.

[30] Isabel Méndez Días, Paula Zabala, "A polyhedral approach for graph coloring," em *Proceedings of the Brazilian Symposium on Graphs, Algorithms and Combinatorics, Vol. 7 of Electronic Nole in Discrete Mathematics*, 2001.

[31] R. C. Corrêa, Y. A. M. Frota, "Computational experiments with the column generation approach for finding the chromatic number of a graph," in *Proceedings of Combinatorial Optimization 2002. , v.1. p.87 - 87*, 2002.

[32] A. Hertz, D. de Werra, "Using tabu search techniques for graph coloring," em *Computing, Vol. 39*, pp. 345–351, 1987.

[33] G. V. Rodrigues Viana, *Meta-Heurísticas e programação paralela em otimização combinatória.* UFC Edições, 1998.

[34] F. Glover, "Future paths for integer programming and links to artificial intelligence," em *CAAI Report 85-8*, 1985.

[35] Valmir C. Barbosa, Carlos A. G. Assis, Josina O. do Nascimento, "Two novel evolutionary formulations of the graph coloring problem," em *Research Report, Programa de Engenharia de Sistemas e Computação, COPPE/UFRJ*, 2001.

[36] Edward C. Sewell, "An improved algorithm for exact graph coloring," em *DIMACS Series on Discrete Mathematics and Theoretical Computer Science*, 1995.

[37] Bassam N. Khoury, Panos M. Pardalos, "An algorithm for finding the maximum clique on an arbitrary graph," em *DIMACS Challenge*, 1993.

[38] B. Pittel, "On the probable behavior of some algorithms for finding the stability number of a graph," em *Mathematical Proceedings of the Cambridge Philosophical Society, 92:511-526*, 1982.

[39] Jürgen Peemöller, "A correction to brélaz's modification of brown's coloring algorithm," em *Communications of the ACM, Vol 26, n. 8*, 1983.

[40] J. Mycielski, "Sur le coloriage des graphes," em *Colloquim Mathematiques, Vol. 3, pp. 161-162*, 1955.

[41] Michael Larsen, James Propp, Daniel Ullman, "The fractional chromatic number of a graph and a construction of mycielski," em *preprint*, 1994.