RELATÓRIO DE INICIAÇÃO CIENTÍFICA

CÁLCULO EFICIENTE DE DERIVADAS VIA COLORAÇÃO DE GRAFOS

Profa. Dra. Margarida P. Mello
Orientadora

Robert Mansel Gower
Aluno

Departamento de Matemática Aplicada
IMECC – UNICAMP

Neste relatório, analizamos o efeito da utilizações de diferentes ordenações de nós nas rotinas de coloração dos grafos coluna-interseção e bipartido. Especificamente, estudaremos as ordenações *maior primeiro*, *menor por último*, *grau de incidência* e *grau de saturação*. Apresentamos resultados computacionais das implementações destes variantes, comparando sua eficiência relativa na coloração dos grafos correspondentes ás matrizes do *University of Florida Sparse matrix collection*, vide [6].

A última parte do relatório é dedicada a modelar o problema da partição simetricamente ortogonal da matriz hessiana. Utilizamos o grafo de adjacência e uma coloração estrela para esta finalidade. Implementamos duas variantes de coloração estrela para testes e comparações.

PARTE I :

A coloração do grafo coluna-interseção e a do grafo bipartido

1 Conceitos Básicos

De forma que o leitor não tenha necessidade de consultar o primeiro relatório, exponho nesta seção as premissas necessárias.

Seja $F : \mathbb{R}^n \to \mathbb{R}^m$ uma função diferenciável. Sua matriz jacobiana é a matriz $m \times n$ J de derivadas parciais. Duas colunas de uma matriz jacobiana são estruturalmente não ortogonais se possuem elementos não nulos em uma mesma linha. Repare na Figura 1 que as colunas 2 e 3 são estruturalmente ortogonais devido aos elementos j_{32} e j_{33}.

$$J = \begin{bmatrix} j_{11} & j_{12} & 0 & 0 & j_{15} \\ 0 & 0 & j_{23} & 0 & 0 \\ 0 & j_{32} & j_{33} & j_{34} & 0 \\ j_{41} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & j_{54} & j_{55} \end{bmatrix}$$

Figura 1: A matriz J é um exemplo de jacobiana extraída de [7]

Um grafo $G = (V, E)$ é composto por um conjunto V de vértices, ou nós, e por um conjunto E de pares não-ordenados de vértices, denominados arestas. Os nós u e v são vizinhos, ou adjacentes, se existir a aresta (u, v). O grafo coluna-interseção $GCI = (V_c, E)$ associado a uma matriz é o grafo cujos nós são as colunas da matriz, $V_c = \{c_1, c_2, \ldots, c_n\}$ e $(c_i, c_j) \in E$ se c_i e c_j são estruturalmente não ortogonais.

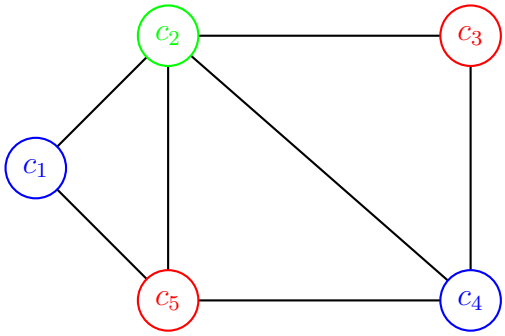


Figura 2: O GCI associado à matriz J da Figura 1

Uma coloração distância-1 do GCI induz uma partição estruturalmente ortogonal das colunas da matriz J. Na Figura 2 temos o GCI associado à matriz jacobiana da Figura 1. A coloração do GCI indica que tanto as colunas c_5 e c_3 como as colunas c_1 e c_4 são pares estruturalmente ortogonais.

O grafo bipartido $GB = (V_c, V_\ell, E)$ associado a uma matriz J $m \times n$, é um grafo que tem nós associados tanto às linhas quanto às colunas da matriz, com $V_c = \{c_1, c_2, \ldots, c_n\}$ e $V_\ell = \{\ell_1, \ell_2, \ldots, \ell_m\}$ e $(c_i, \ell_k) \in E$ se $j_{ki} \neq 0$. Na Figura 3 temos o GB associado ao jacobiano da Figura 1. Denotamos o quadrado de um grafo $G = (V, E)$ por $G^2 = (V, \widetilde{E})$. O grafo G^2 tem o mesmo conjunto de vértices qu e G e uma aresta $(u, v) \in \widetilde{E}$ se existir um caminho de distância menor ou igual a dois que liga u e v. Dizemos que dois vértices distintos estão à distância k se o menor caminho que os ligam tem comprimento k. Dado um grafo $G = (V, E)$, denotamos por $G[\widehat{V}] = (\widehat{V}, \widehat{E})$ o subgrafo induzido por $\widehat{V}$. Neste grafo, $(u, v) \in \widehat{E}$ se existir $(u, v) \in E$. É fácil concluir que, se dois nós $c_i, c_j \in V_c$ são adjacentes em GB^2, então c_i e c_j são estruturalmente não ortogonais. Portanto, o grafo $GB^2[V_c]$, o subgrafo induzido por V_c em GB^2, é precisamente o GCI da matriz J. Quando vizinhos de distância 1 e de distância 2 tem cores distintas, dizemos que é uma coloração distância-2. Consequentemente, uma coloração parcial distância-2 em GB dos nós colunas V_c, é também uma coloração distância-1 no grafo GCI, e vice-versa. Note que, não existe um par $(c_i, c_j) \in V_c$ de nós pertencentes ao GB a distância-1, dado que nós colunas são somente ligados diretamente a nós linhas. De agora em diante, vamos nos referir a essa coloração parcial distância-2 como a coloração do grafo bipartido. O número cromático de um grafo $\chi(G)$, é o menor número de cores que podemos usar para efetuar uma coloração distância-1. Limites inferior e superior para $\chi(G)$, é o tamanho do maior clique $\omega(G)$ e o grau máximo incrementado por um, $\Delta + 1$, respectivamente. Um clique de tamanho k, é um grafo com k

nós que são vizinhos dois a dois. Logo temos a relação:

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

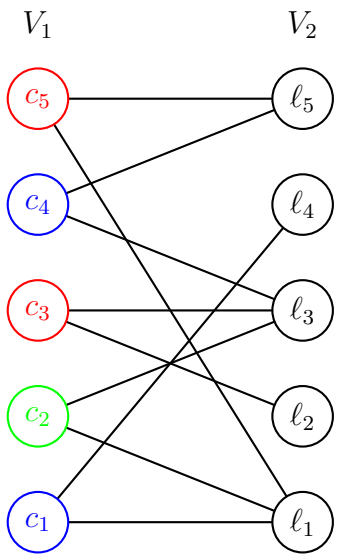


Figura 3: Grafo bipartido GB associado à matriz J da Figura 1

Portanto colorações que utilizam $\omega(G)$ são certamente otimais. Entretanto, o cáculo do número cromático de um grafo é um problema NP-completo. Então utilizamos uma cota inferior para $\chi(G)$, que pode ser facilmente calculada. Para tanto, observamos que se J contêm k elementos não nulos em uma linha, então o GCI associado contêm um clique de tamanho k. Chamando de ρ_{max} o número máximo de não nulos em uma mesma linha, temos as seguintes cotas para o número cromático do GCI associado a J:

$$\rho_{max} \leq \omega(G) \leq \chi(G_c) \leq \Delta + 1$$

2 Introdução a coloração do *GCI* e *GB*

Vamos abordar alguns incentivos teóricos para diversas formas de atravessar os nós e posteriomente comparar implementações. Antes disso, responderemos algumas questões fundamentais.

O problema geral de obter uma coloração para um grafo é NP-completo, entretanto será que isso é verdade para os grafos coluna-interseção? Ou será que esses grafos apresentam propriedades que tornam possível a criação de um algoritmo de coloração polinomial?

Coleman e moré [3] demonstram que, para todo grafo G, existe uma matriz J cujo GCI é isomórfo a G. A existência de um algoritmo polinomial que produz uma coloração ótima em grafos coluna-interseção, produziria uma coloração ótima em tempo polinomial para grafos

genéricos. Concluímos que o problema de obter uma coloração ótima para o GCI é NP-completo. E, se restringimos aos casos em que o grafo coluna-interseção provém de uma matriz quadrada, mesmo assim, para todo grafo G, existe uma matriz quadrada A cujo GCI aceita uma coloração-k, k$\geq$ 3, se, e somente se existe uma coloração-k para o grafo G, vide [3]. Para todo grafo coluna-interseção $GCI(V_c, E_c)$ existe um grafo bipartido associado $GB(V_c, V_\ell, E_b)$ onde $(GB^2[V_1]) = G_c$. Portanto, determinar a coloração distânica-2 ótima para o GB é NP-completo também.

Com isso em mente, procuramos heurísticas que aproximam a solução ótima, e, mais especifcamente, da forma apresentada no Algoritmo 1. Esse Algoritmo Sequencial Ganancioso executa uma coloração distância-1 de um grafo $G = (V, E)$. Cada vértice v_i é colorido fazendo uma busca pela sua vizinhança, $N(v_i)$, linha 5. Denotaremos o conjunto dos nós à distância k de um vértice v por $N_k(v)$. Logo, $N_1(v)$, ou simplesmente $N(v)$, é o conjunto dos vizinhos de v. Armazenando uma lista de cores proibidas para o nó v_i denominada *forbiddenColors*. Se $forbiddenColors_j = v_i$ sabemos que a cor j é proibido para o nó v_i. Após terminar a busca pelo vizinhança, a menor cor que não seja proibida é atribuida ao nó v_i, linha 7.

Subrotina $ASG(G = (V, E))$
(1) **Seja** $v_1, v_2, \cdots, v_n$ uma ordenação dos nós.
(2) **Inicialize** o vetor *forbiddenColors* com n posições com $a \notin V$
(3) Atribua a cor 1 a v_1.
(4) **Para** i de 2 até n
(5) **Para** cada w in $N(v_i)$ colorido
(6) $forbiddenColors[cor(w)] \leftarrow v_i$
(7) $cor(v_i) \leftarrow \min\{c \in N : forbiddenColors[c] \neq v_i\}$

Algoritmo 1: Algoritmo Sequencial Ganancioso (ASG)

3 *Maior primeiro* e *Menor por último*

Dada uma ordem dos nós $V_i = v_1, v_2, \cdots, v_i$, $G[V_i]$ é o grafo induzido pelos nós que forem visitados até i-ésimo passo de ASG. O número de vizinhos à distância k de um nó v é denotado por $d_k(v)$. Portanto $d_1(v)$, ou simplesmente $d(v)$, é o número de vizinhos do nó v, chamado de grau do nó v. Definimos $d(v_i, G[V_i])$ como o número de vizinhos do v_i que pertence ao grafo $G[V_i]$. Logo, no i-ésimo passo, o nó v tem $d(v_i, G[V_i])$ vizinhos coloridos. E se todos estes vizinhos têm cores distintas, o Algoritmo 1 irá colorir o i-ésimo v_i nó com a cor $d(v_i, G[V_i])+1$.

Seja q_i o número cores usadas pelo algoritmo AGS até o i-ésimo passo. Os seguintes

limites superiores para q_n, inspiraram as duas primeiras ordenações *maior primeiro* (MP) e *menor por último* (MU).

$$q_n \leq \max_{1\leq k\leq n}\{d(v_k, G[V_k]) + 1\} \quad (1)$$

$$q_n \leq \max_{1\leq k\leq n} \min\{d(v_k), k-1\} + 1 \quad (2)$$

Para a primeira equação suponha, por absurdo, que

$$q_n > \max_{1\leq k\leq n} d(v_k, G[V_k]) + 1.$$

Logo existe um nó v_j com a cor q_n, e portanto tem, pelo menos, $q_n - 1$ vizinhos com cores distíntas. Ou seja, v_j possui mais que $\max_{1\leq k\leq n} d(v_k, G[V_k])$ vizinhos com cores distíntas. Mas isso é impossível, dado que $\max_{1\leq k\leq n} d(v_k, G[V_k])$ é justamente o maior número de vizinhos coloridos de um nó.

A segunda desigualdade é apenas uma versão mais fraca da primeira. É claro que $d(v_i, G[V_i]) \leq d(v_i)$. E, no i-ésimo passo, $d(v_i, G[V_i]) \leq i - 1$. E assim fica demonstrado. ■

Uma ordem não crescente dos nós em relação aos graus minimiza o limite superior na desigualdade (2). Essa ordenação é conhecida como *maior primeiro* (MP), e foi sugerida por Welsh e Powell [13]. Nessa ordem, para cada i, $1 \leq i \leq n$,

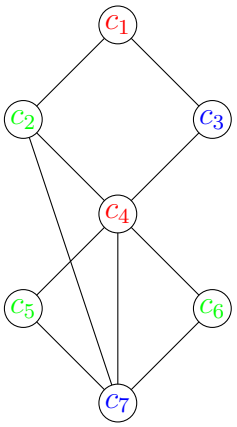
$$d(v_i, G) = \max_{v_j \in V - V_{i-1}} d(v_j, G).$$

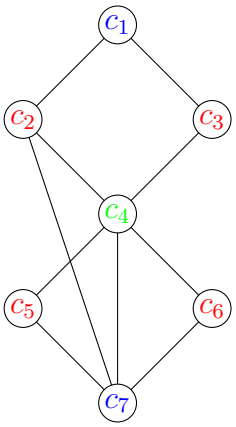
Para minimizar a desigualdade (1), temos a ordenação *menor por último* (MU). Nessa ordenção o último nó v_n, é escolido de tal forma que seja o nó de menor grau em G. E v_{n-1}, o nó de menor grau no grafo $G[V - v_n]$, e assim recursivamente. Imagine que os vértices $v_{i+1}, v_{i+2}, \cdots, v_n$ da ordenação, foram determinados. O nó v_i será o nó de menor grau no grafo induzido por $V - \{v_{i+1}, v_{i+2}, \cdots, v_n\}$. Repare que a ordenação é calculada na ordem inversa, de v_n até v_1, e depois os nós são de fato coloridos e percorridos começando em v_1 e terminando em v_n. A ordenação MU foi proposta por Matula [12]. Devido à construção da ordem MU, temos que, para cada i, $1 \leq i \leq n$,

$$d(v_i, G[V_i]) = \min_{v_j \in V} d(v_j, G[V_j]).$$

Por isso essa ordem minimiza a desigualdade (1). Na Figura 4, temos dois GCI associados à matriz à direita na figura. O grafo à esquerda foi colorido com a ordem MP e à direita com MU. As maiores cliques desses GCI são de tamanho 3, logo as duas colorações foram ótimas por usarem 3 cores. Coleman e Moré [3] mostram que tanto na ordem MP quanto na MU não há garantia de otimalidade, até mesmo em grafos simples, e que poderiam colorir

um grafo bipartido com muito mais que duas cores. Um exemplo da ordem *MP* usando três cores em um grafo bipartido está na Figura 5. É claro que toda ordem nesse grafo é uma ordem *MP*, dado que todos os nós têm o mesmo grau. Um exemplo em que a ordem *MU* usa muito mais que duas cores para colorir um grafo bipartido está em [3].





$$\begin{bmatrix} a_{11} & a_{12} & 0 & 0 & 0 & 0 & 0 \\ a_{21} & 0 & a_{23} & 0 & 0 & 0 & 0 \\ 0 & 0 & a_{33} & a_{34} & 0 & 0 & 0 \\ 0 & a_{42} & 0 & a_{44} & 0 & 0 & a_{47} \\ 0 & 0 & 0 & a_{54} & a_{55} & 0 & a_{57} \\ 0 & 0 & 0 & a_{64} & 0 & a_{66} & a_{67} \end{bmatrix}$$

Figura 4: Acima, grafo coluna-interseção associado à matriz ao lado. O grafo à esquerda foi colorido com a ordem *MP*, na ordem c_4, c_7, c_2, c_1, c_3, c_5 e c_6 e o grafo à direita foi colorido com a ordem *MU*, perocorrendo c_6, c_7, c_4, c_5, c_2, c_3 e c_1.

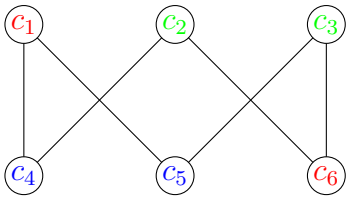


Figura 5: Um grafo bipartido colorido com uma ordem *MP* que percorrer os nós na sequência c_6, c_1, c_4, c_5, c_2, e c_3.

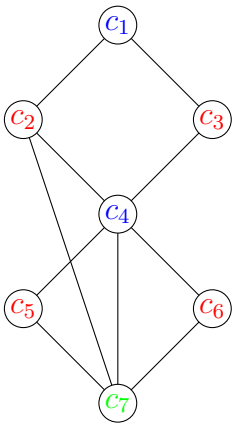
4 *Grau de incidência* e *grau de saturação*

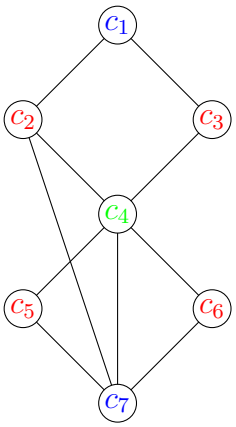
Duas ordens que garantem otimalidade em grafos bipartidos, Coleman [3], são as ordens *grau de incidência* (*GI*) e *grau de saturação* (*GS*). Dada uma iteração do Algoritmo *ASG*, a *MP* escolhe o nó com maior número de vizinhos coloridos. O *grau de incidência* de um nó é justamente $d(v_i, G[V_i])$. E a ordenação *GS* escolhe o nó com maior *grau de saturação*, o maior número de vizinhos com cores distíntas. A *GI* foi proposta por Coleman and Moré [3] e a *GS* por Breláz [1].

Dado um vértice, o seu *grau de incidência* (*id*), *grau de saturação* (*sd*) e grau usual (*d*) satifazem a desigualdade

$$sd \leq id \leq d$$

Na Figura 6, o grafo à esquerda foi colorido com uma ordem *GI* e à direita com uma ordem *GS*. A coloração de cada grafo foi iniciada no nó 1. Percebe-se que o primeiro nó a ser colorido é arbitrário, dado que todos os nós estão inicialmente com *GI* e *GS* nulo. As colorações foram ótimas por usarem 3 cores em grafos cujas maiores cliques são de tamanho três.





$$\begin{bmatrix} a_{11} & a_{12} & 0 & 0 & 0 & 0 & 0 \\ a_{21} & 0 & a_{23} & 0 & 0 & 0 & 0 \\ 0 & 0 & a_{33} & a_{34} & 0 & 0 & 0 \\ 0 & a_{42} & 0 & a_{44} & 0 & 0 & a_{47} \\ 0 & 0 & 0 & a_{54} & a_{55} & 0 & a_{57} \\ 0 & 0 & 0 & a_{64} & 0 & a_{66} & a_{67} \end{bmatrix}$$

Figura 6: Na Figura, temos dois grafos idênticos, cada um sendo o grafo coluna-interseção associado à matriz à direita. O grafo mais à esquerda foi colorido com a ordem *GI*, na ordem c_1, c_2, c_4, c_7, c_5, c_6 e c_3 e o grafo no centro foi colorido com a ordem *GS*, atravesando c_1, c_2, c_7, c_4, c_3, c_5 e c_6.

Entre as quatro ordens apresentadas, a ordem *GS* é a que melhor modela as opções de cores de um nó. Se um nó tem um alto *GS* alto, isso retringe a escolha de cores com que podemos colorí-lo. Portanto, escolher sempre o nó com o maior *GS* é uma tentativa de colorir o nó com menos opções de cores.

5 O grafo bipartido, coloração distância-2

Subrotina $AGS2(G = (V, E))$
Seja $v_1, v_2, \cdots, v_n$ uma ordenação dos nós V_1.
Atribua a cor 1 a v_1.
Para i de 2 até n
Para cada w in $N_2(v_i)$ colorido
$forbiddenColors[cor(w)] \leftarrow v_i$
$cor(v_i) \leftarrow \min\{c \in N : forbiddenColors[c] \neq v_i\}$

Algoritmo 2: Algoritmo Sequêncial Guloso distância-2 ($ASG2$)

Na Figura 2 temos o Algoritmo sequencial básico para a coloração do grafo bipartido.

Em relação ao que foi dito para a coloração distância-1, há uma analogia para a coloração do grafo bipartido.

Seja $G[V_k, V_2]$ o grafo induzido pelos nós $\{c_1, c_2, \ldots, c_k\} \in V_1$ e os nós contidos em V_2. E seja q_i o número de cores usado pelo Algoritmo 2 até i-ésimo passo. À medida que o Algoritmo 2 prossegue, $d_2(v_i, G[V_k, V_2])$ é o número de vizinhos distância-2 do nó v_i que já foram visitados. Apartir de agora vamos nos referir ao número de vizinhos distância-2 de um nó v como o grau-2 desse nó ou $d_2(v)$. O *ASG2* utilizará uma nova cor no nó v_i se todas as cores estão em uso pelos vizinhos distância-2 de v_i. Essa nova cor será a cor $d_2(v_i, G[V_k, V_2]) + 1$. Disso deduzimos as desigualdades análogas:

$$q_n \leq \max_{1 \leq k \leq n} \{d_2(v_k, G[V_k, V_2]) + 1\} \tag{3}$$

$$q_n \leq \max_{1 \leq k \leq n} \min\{d_2(v_k), k - 1\} + 1 \tag{4}$$

Deduzimos também a ordem *menor por último* de distância-2 (*MU2*) e a ordem *maior primeiro* de distância-2 (*MP2*). Na *MU2* v_n é o nó com o menor grau-2. Dado que os nós $\{v_{i+1}, v_{i+2}, \ldots, v_n\}$ foram determinados, seja v o nó que minimiza:

$$d_2(v, G[V_1 - \{v_{i+1}, v_{i+2}, \ldots, v_n\}, V_2]).$$

Esse nó v será o i-ésimo nó da ordem. E na *MP2*, v_1 é o nó com o maior grau-2, e v_2 o maior, tirando v_1. Dado que $\{v_1, v_2, \ldots, v_{i-1}\}$ foram determinados, seja v_i o nó com i-ésimo maior grau-2. A ordem *grau de saturação* de distância-2 (*GS2*) e *grau de incidência* (*GI2*) de distância-2 seguem da mesma forma. Enquanto na ordem *GI2* escolhemos sempre o nó com o maior número de vizinhos distância-2 coloridos, na ordem *GS2* escolhemos o nó com o maior número de vizinhos distância-2 com cores distintas. Na Figura 7, o grafo bipartido à esquerda foi colorido com uma ordem *MP2*, mas também satisfaz uma ordem *GI2*. O grafo à direita satisfaz uma ordem *GS2* e *MU2*.

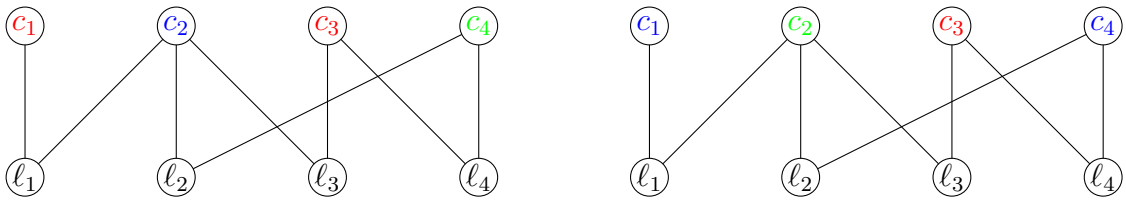


Figura 7: À esquerda um grafo bipartido colorido na ordem c_2, c_3, c_4 e c_1, que é uma ordem *GI2* e *MP2*. O grafo bipartido à direita, foi colorido na ordem c_4, c_3, c_2 e c_1, que é uma ordem *GS2* e *MU2*.

6 Os Algoritmos de *maior primeiro* e *menor por último*

Para obtermos o Algoritmo de *maior primeiro*, basta reorganizarmos os nós de acordo com seus graus, e depois executar o Algoritmos Sequential Guloso. Nas linhas 2, 3 e 4 do pseudo-

código da subroutina *MP* no Algoritmo 3, calculamos o grau de cada nó contando o número de vizinhos.

Subrotina *MP* $(G = (V, E))$
(1) **Seja** $v_1, v_2, \cdots, v_n$ a ordem natural dos nós.
(2) **Para** i de 2 até n
(3) **Para** cada $w \in \mathrm{N}(v_i)$.
(4) $d(v_i) = d(v_i) + 1$
(5) **Ordena** os nós de V de maior até menor grau.
(6) **Para** i de 1 até n
(7) Atribua v_i a menor cor diferente de seus vizinhos

Algoritmo 3: Algoritmo do *maior primeiro*

As três ordenações, *MU*, *GI* e *GS* seguem um esquema de passos comum, detalhado abaixo. Supomos que os nós $v_1, v_2, \ldots, v_{i-1}$ foram determinados e que os nós restantes $V - \{v_1, v_2, \ldots, v_{i-1}\}$, estão ordenados de acordo com o grau especializado. Com grau especializado, estamos nos referindo ao *grau de saturação*, *grau de incidência* ou grau induzido pelo subgrafo $G[V - v_1, v_2, \ldots, v_{i-1}]$ dependendo da ordenação em questão.

1. Escolha o i-ésimo nó, v_i, da ordenação apartir de $V - \{v_1, v_2, \ldots, v_{i-1}\}$.
2. Atualize o grau especializado dos nós $V - v_1, v_2, \ldots, v_i$.
3. Ordene $V - v_1, v_2, \ldots, v_i$ de acordo com seus graus especializados.

O terceiro passo é o que requer mais atenção. Os nós $V - v_1, v_2, \ldots, v_i$ estavam em ordem de acordo com o grau especializado, então, após a atualização dos graus especializados, os nós $V - v_1, v_2, \ldots, v_i$ estão "quase em ordem". Precisamos de uma subrotina que aproveite essa estrutura "quase ordenada", produzindo uma reordenação eficiente.

No Algoritmo 4 apresentamos o pseudo-código do Algoritmo *menor por último*. Manteremos dois vetores, um com os possíveis candidatos a serem próximos na ordenação, *V_ordem*, e um com nós cujas posições na ordenação já foram determinadas, *V_MU*. Essa ordenação é calculada de traz para frente. Primeiro ordenamos os nós, do menor para o maior grau (linha 3) e armazenamos essa ordem em um vetor, *V_ordem*. O nó de menor grau será o último nó a ser colorido, o v_n, e o armazenamos em *V_MU*. Para determinar o penúltimo, atualizamos os graus dos nós, diminuindo os graus dos vizinhos de v_n em um, linha 10. Depois da atualização de cada vizinho, o deslocamos para a esquerda dentro de *V_ordem* até encontrarmos um nó com um grau menor, e o vetor estar em ordem, linha 11. Depois de atualizar todos os

vizinhos, excluímos v_n do grupo dos possíveis candidatos, *V_ordem*, linha 12. Continuamos assim até eliminarmos todos os nós de *V_ordem*. Após determinar a ordem *MU*, efetuamos a coloração. Repare que, para cada posição excluída de *V_ordem*, adicionamos um elemento ao *V_MU*, então, ao invés de termos dois vetores, *V_MU* e *V_ordem*, usamos um só na prática.

Subrotina $MU(G = (V, E))$
(1) **Seja** $v_1, v_2, \cdots, v_n$ a ordem natural dos nós.
(2) **Obtenha** o grau de cada nó $v \in V$
(3) **Ordene** os nós de V de menor até maior grau.
(4) **Seja** *V_ordem* o vetor dos nós ordenados.
(5) **Inicialize** *V_MU*, um vetor de n posições
(6) **Para** i de 1 até n
(7) **Seja** v o primeiro nó em *V_ordem*
(8) **Insira** v na i-ésimo posição de *V_MU*, $V\_MU_i \leftarrow v$
(9) **Para** cada $w \in \mathrm{N}(v)$.
(10) $d(w) = d(w) - 1$
(11) Desloque w dentro de *V_ordem* até que esteje em ordem.
(12) **Exclua** v de *V_ordem*
(13) **Colora** O grafo na ordem *V_MU*.

Algoritmo 4: Algoritmo do *menor por último*

7 Os Algoritmos de *grau de incidência* e *grau de saturação*

Obtemos as ordenações *menor por último* e *maior primeiro* antes de efetuar a coloração, diferentemente das ordenações *grau de incidência* e *grau de saturação* que são calculados durante a coloração.

Uma ordenação *GS* é essentialmente uma ordenação *GI* com um critério a mais. Então vamos começar examinando a ordenação *GI*. A estrutura básica dessa ordem está no Algoritmo 5.

Manteremos o *grau de incidência* de cada nó no vetor *V_incidência*, onde a i-ésimo posição desse vetor, $V\_incidência_i$ é o *grau de incidência* do i-ésimo nó. Lembrando que o *grau de incidência* de um nó é o número de vizinhos que estão coloridos em um dado momento, quando o Algoritmo inicia-se, incializamos *V_incidência* com n elementos, todos nulos. Portanto, o primeiro nó escolhido para ser colorido é arbitrário. Depois de atualizar o *grau de incidência* dos vizinhos, ordenamo-os, e escolhemos o nó com o maior *grau de incidência*. Se mantemos um vetor com os nós ordenados de acordo com seus *graus de*

Subrotina *GI* (I) ($G = (V, E)$)
(1) **Seja** v_1 um nó inicial.
(2) **Para** i de 1 até n
(3) Escolha o nó v_i com o maior *grau de incidência*
(4) e colora com a menor cor disponível.
(5) Atualize o *grau de incidência* dos vizinhos de v_i.
(6) Organize os nós restantes de acordo com seus *grau de incidência*.

Algoritmo 5: Algoritmo do *grau de incidência*

incidência, podemos facilmente determinar o próximo nó a ser colorido. Seja *V_ordem* um vetor de n posições para esse propósito.

De forma detalhada, temos o Algoritmo 6. Nas linhas 1 e 2 inicializamos *V_incidência* com n posições nulas e *V_ordem* com uma ordem arbitrária, dado que nemhum nó está colorido. Após colorir um nó, atualizamos o *grau de incidência* de seus vizinhos, linha 7. Depois de atualizar cada vizinho w_i, ordenamos *V_ordem*, deslocando w_i dentro de *V_ordem* até estar ordenado novamente.

Subrotina *GI* (II) ($G = (V, E)$)
(1) **Inicialize** *V_ordem* um vetor de n posições com a ordem natural dos inteiros
(2) **Inicialize** *V_incidência* um vetor de n posições nulas
(3) **Para** i de 1 até n
(4) **Seja** v o primeiro nó em *V_ordem*
(5) **Colora** v com a menor cor diferente de seus vizinhos.
(6) **Para** cada $w_i \in \mathrm{N}(v)$.
(7) $V\_incidência_i = V\_incidência_i + 1$
(8) Desloque w_i dentro de *V_ordem* até estar em ordem.
(9) **Exclua** v de *V_ordem*

Algoritmo 6: Algoritmo do *grau de incidência*

Uma ordenação pelo *grau de saturação* é muito similar. Adicionando apenas um critério, transformamos nosso Algoritmo *grau de incidência* em um Algoritmo *grau de saturação*. O *grau de saturação* de um nó é o número de vizinhos com cores distintas. Então, após colorir um dado nó v com a cor x, e, ao atualizar o *grau de saturação* de um nó vizinho w, precisamos manter em mente se w já tem um vizinho com a cor x.

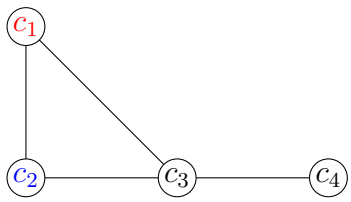
Uma forma de fazer isso é manter um vetor associado a cada nó, onde um 1 na i-ésima posição desse vetor indica que o nó em questão tem um vizinho com a i-ésima cor. Uma modificação cara em termos de memória. Também não seria razoável fazer uma busca para

checar se existe um vizinho do nó w colorido com a i-ésima cor, toda vez que desejamos atualizar o *grau de saturação* de w, isso serio caro em termos de tempo de execução.

Uma solução seria associar a cada cor em uso um número primo, lembrando que representamos uma cor por um número natural. Usamos o vetor *V_primo* para esse propósito, em que V_primo_i é o i-ésimo primo que é associado à i-ésima cor:

Cores em uso	1	2	3	4	5	6
V_primo	2	3	5	7	11	13

Para cada nó v_i anexamos um inteiro V_unique_i, que é um múltiplo dos primos contidos em *V_primo*. Se V_unique_i é múltiplo de V_primo_j, sabemos que o nó v_i tem pelo menos um vizinho com a j-ésima cor. Veja na Figura 8 que o V_unique_1 é 3 dado que o nó c_1 tem um vizinho com a cor 2 cujo primo associado é 3. O V_unique_3 é 6, isso porque o nó 3 tem vizinhos com as cores 2 e 1 cujos primos respectivos são 3 e 2, e claro $3 \cdot 2 = 6$. No



{1,2} Cores em uso.

$\{2, 3\}$ *V_primo*

$\{3, 2, 6, 1\}$ *V_unique*

Figura 8: Nessa Figura, só os nós c_1 e c_2 foram coloridos com a cor 1 (Vermelho) e a cor 2 (Azul). Os número primos 2 e 3 são associados às cores 1 e 2, respectivamente. Pelo vetor *V_unique* vemos que c_1 tem, pelo menos, um vizinho com a cor 2, c_2 possue um vizinho com a cor 1 e c_3 tem vizinhos com às cores 1 e 2.

Algoritmo 7, apresentamos o pseudo-código de uma coloração com a ordem *GS*. A única mudança significativa no código em relação ao Algoritmo *GI* está nas linhas 9 e 11. Na linha 9, realizamos a verificação se V_unique_j é um múltiplo de V_Primo_{ki} e, se não for, incrementamos o *grau de saturação* do nó w_j e, na linha 11, fazemos com que V_unique_j seja múltiplo de V_Primo_{ki}. Repare que *V_unique* é um vetor de n inteiros, enquanto *V_Primo* é um vetor com tantos elementos quantos cores. Mas a priori não sabemos quantas cores serão necessárias, mas temos o limite superior conveniente $\Delta + 1$, que é proporcionalmente pequeno em comparação a n. No nosso grupo de matrizes de teste, em média, $\Delta + 1$ é 2% de n.

Subrotina *GS* $(G = (V, E))$
(1) **Inicialize** *V_ordem* um vetor de n posições com a ordem natural dos inteiros
(2) **Inicialize** *V_Primo* um vetor dos $\Delta + 1$ primeiros primos
(3) **Inicialize** *V_unique* um vetor de n primeiros primos
(4) **Inicialize** *V_saturação* um vetor de n posições nulas
(5) **Para** i de 1 até n
(6) **Seja** v o primeiro nó em *V_ordem*
(7) **Colora** v com a menor cor diferente de seus vizinhos, e seja ki essa cor.
(8) **Para** cada $w_j \in \mathrm{N}(v)$.
(9) **Se** V_unique_j não é divisível por V_Primo_{ki}
(10) $V\_saturação_j = V\_saturação_j + 1$
(11) $V\_unique_j = V\_Primo_{ki} V\_unique_j$
(12) Desloque w_i dentro de *V_ordem* até que esteja em ordem.
(13) **Exclua** v de *V_ordem*

Algoritmo 7: Algoritmo do *grau de saturação*

8 Implementação das colorações no grafo bipartido

Em todos os Algoritmos, efetuamos varías buscas pela vizinhança de um nó. Agora, no grafo bipartido, vamos precisar fazer essa busca pelos vizinhos distância-2, o que é mais custoso em termos de tempo. Além desse custo, há um problema de verificação. Entre dois nós à distância 2, podem haver mais de um caminho de distância-2. Repare na Figura 9 que os caminhos c_1, ℓ_1, c_2 e c_1, ℓ_2, c_2 conectam c_1 e c_2. No Algoritmo 8, calculamos o grau-2 do nó

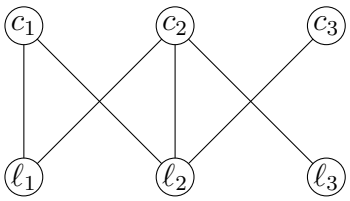


Figura 9: Um grafo bipartido onde os nós os c_1 e c_2 são connectados pelos caminhos c_1, ℓ_1, c_2 e c_1, ℓ_2, c_2.

c_1, sem esse cuidado de repetir visitas. Veja que, na linha (3), verificamos se $c_i \neq c_1$ porque o grafo não é orientado, logo, se $w \in N(c_1)$ temos que $c_1 \in N(w)$.

Para evitar fazer varías contagems do mesmo vizinho distância-2, anexamos a cada nó c_i um inteiro ult_visita_i. Esse inteiro é igual o último vizinho distância-2 a visitá-lo. No Algoritmo 9, adicionamos mais um loop na linha 1 para calcular o grau-2 de cada nó, e, nas linhas 4 e 6 incluímos um critério e um atualização, respectivamente. O critério serve para

Seja $G = (V_1, V_2, E)$ um grafo bipartido
(1) **Para** todo $w \in N(c_1)$
(2) **Para** todo $c_i \in N(w)$
(3) **Se** $c_i \neq c_1$
(4) $d_2(c_1) = d_2(c_1) + 1$

Algoritmo 8: Algoritmo de busca sem o cuidado de repetir visitas.

verificar se o último nó a visitá-lo foi o nó c_j. E atualização para informar que já foi visitado pelo nó c_j. Portanto durante a contagem do grau-2 do nó c_j, cada vizinho distância-2 será contado apenas uma vez.

Seja $G = (V_c, V_\ell, E)$.
Inicialize um vetor *ult_visita* de n inteiros, todos nulos.
(1) **Para** j de 1 até n
(2) **Para** todo $w \in N(c_j)$
(3) **Para** todo $c_i \in N(w)$
(4) **Se** $c_i \neq c_j$ e $ult\_visita_i \neq j$
(5) $d_2(c_j) = d_2(c_j) + 1$
(6) $ult\_visita_i = j$

Algoritmo 9: Algoritmo de busca que não repete visitas

Tendo uma forma de contar quantos vizinhos distância-2 tem cada nó, para obter um Algoritmo de *maior primeiro* distância-2 (*MP2*), basta organizar os nós de acordo com seus graus 2, e executar a coloração sequencial guloso nessa ordem (Algoritmo 10).

Para obter o Algoritmo *menor por último* distância-2 (*MU2*) a partir do algoritmo *MU*, precisamos tormar cuidado com a atualização do graus-2 dos nós, para não atualizar o mesmo nó mais de uma vez. Apresentamos o pseudo-código do *MU2* no Algoritmo 11. Em relação ao Algoritmo 4, adicionamos mais um loop na linha 10, para ter uma busca pela vizinhança distância-2 e não distância-1. Nas linha 11 e 13, a verificação é necessária para não repetir atualizações. E claro, na linha 3, está contido o Algoritmo 8. Fazendo uma alteração parecida no Algoritmo *GI*, obtemos um Algoritmo distância-2 (*AGI2*). Repare no Algoritmo 12 na linha 8 adicionamos um *loop*, e nas linhas 9 e 11 o necessário para não repetir atualizações do mesmo nó. Entretanto, para o Algoritmo *GS2*, essa verificacção a mais usando *ult_visita* não será necessária.

Repare o pseudo-código no Algoritmo 13. Após um nó ser colorido, linha 7, o Algoritmo começa a atualização do *grau de saturação* dos seus vizinhos distância-2, linhas 8 até 13.

Seja $(G = (V_c, V_\ell, E))$ um grafo bipartido.
(1) **Seja** $v_1, v_2, \cdots, v_n$ a ordem natural dos nós V_1.
(2) **Inicialize** um vetor *ult_visita* de n inteiros, com todos elementos nulos.
(3) **Calcule** o grau-2 de cada nó v, $v \in V_1$
(4) **Ordena** os nós de V_1 de maior até menor grau-2.
(5) **Para** i de 1 até n
(6) Atribua v_i a menor cor diferente de seus vizinhos

Algoritmo 10: Algoritmo do *maior primeiro*

Subrotina *AMU2* $(G = (V_c, V_\ell, E))$
(1) **Seja** $v_1, v_2, \cdots, v_n$ a ordem natural dos nós V_1.
(2) **Obtenha** o grau-2 de cada nó $v \in V_1$
(3) **Ordene** os nós de V_1 de menor até maior grau-2.
(4) **Seja** *V_ordem* o vetor dos nós ordenados.
(5) **Inicialize** *V_MU*, um vetor de n posições
(6) **Para** i de 1 até n
(7) **Seja** v_i o primeiro nó em *V_ordem*
(8) **Insira** v_i na i-ésimo posição de *V_MU*, $V\_MU_i \leftarrow v_i$
(9) **Para** cada $w \in \mathrm{N}(v_i)$.
(10) **Para** cada $c_j \in \mathrm{N}(w)$.
(11) **Se** $c_i \neq c_j$ e $ult\_visita_i \neq j$
(12) $d_2(w) = d_2(w) - 1$
(13) $ult\_visita_i = j$
(14) Desloque w dentro de *V_ordem* até que esteje em ordem.
(15) **Exclua** v de *V_ordem*
(16) **Colora** O grafo na ordem *V_MU*.

Algoritmo 11: Algoritmo do *menor por último*

Subrotina *GI2*$(G = (V_c, V_\ell, E))$

(1) **Inicialize** *V_ordem* um vetor de n posições com a ordem natural dos inteiros
(2) **Inicialize** *V_incidência* um vetor de n posições nulas
(3) **Inicialize** *ult_visita* um vetor de n posições nulas
(4) **Para** i de 1 até n
(5) **Seja** v o primeiro nó em *V_ordem*
(6) **Colora** v com a menor cor diferente de seus vizinhos.
(7) **Para** cada $w_i \in \mathrm{N}(v)$.
(8) **Para** cada $z \in \mathrm{N}(w)$.
(9) **Se** $c_i \neq c_j$ e $ult\_visita_i \neq j$
(10) $V\_incidência_i = V\_incidência_i + 1$
(11) $ult\_visita_i = j$
(12) Desloque w_i dentro de *V_ordem* até estar em ordem.
(13) **Exclua** v de *V_ordem*

Algoritmo 12: Algoritmo do *grau de incidência*

Mesmo visitando um vizinho distância-2 mais que uma vez, o *grau de saturação* será incrementado apenas uma vez, dado que V_unique_j será alterado após a primeira visita, linha 12. Ou seja, o que importa no *GS2*, é o número de cores distintas na vizinhança distância-2, e um nó evidentemente tem somente uma cor.

Veja que, para obter um *GI2* apartir do *GI*, foram necessário mais n inteiros, o vetor *ult_visita*. Enquanto não foi necessário nenhum memória adicional para implementar o *GS2*.

9 Resultados numéricos das colorações

Os Algoritmos descritos foram implementados em linguagem C++ e executados em um Intel Pentium 4, 3.00GHz com 896MB de RAM, no sistema Microsoft Windows XP 2002 usando o compilador MingW32.

As matrizes provieram do mesmo banco de matrizes esparsas que os autores de [7], o *University of Florida Sparse Matrix Collection* [6], com exceção das matrizes e40r0100_2.mtx e e30r2000.mtx, que foram tiradas do sítio do Matrix Market [14], Tabela 1.

Todas as matrizes com prefixo "lp" provieram de problemas de programação linear. A partir de agora, omitiremos esse prefixo para fins de facilitar a formatação de gráfico e tabelas. Dividimos as matrizes em dois grupos. Repare a linha horizontal que divide a matriz cage12 e cavity13.As matrizes acima dessa linha foram escolhidas por Gebremedhin et al [7] para testar as suas heurísticas, e as matrizes abaixo dessa linha foram escolhidas propositalmente

Subrotina $GS(G = (V_c, , V_\ell, E))$
(1) **Inicialize** *V_ordem* um vetor de n posições com a ordem natural.
(2) **Inicialize** *V_Primo* um vetor dos $\Delta + 1$ primeiros primos
(3) **Inicialize** *V_unique* um vetor de n primeiros primos
(4) **Inicialize** *V_saturação* um vetor de n posições nulas
(5) **Para** i de 1 até n
(6) **Seja** v o primeiro nó em *V_ordem*
(7) **Colora** v com a menor cor diferente de seus vizinhos, e seja ki essa cor.
(8) **Para** cada $w \in \mathrm{N}(v)$.
(9) **Para** cada $z \in \mathrm{N}(w)$.
(10) **Se** V_unique_j não é divisível por V_Primo_{ki}
(11) Incremente $V\_saturação_j$ em um
(12) $V\_unique_j = V\_Primo_{ki} V\_unique_j$
(13) Desloque w_i dentro de *V_ordem* até que esteja em ordem.
(14) **Exclua** v de *V_ordem*

Algoritmo 13: Algoritmo do *menor por último*

por serem mais densos. A Tabela 1 contém dados relativos às matrizes. As colunas m, n e nnz contêm o número de linhas, colunas e elementos não nulos, respectivamente. O máximo e mínimo de não nulos por linha são denotados por $\rho_{\max}$ e $\rho_{\min}$, respectivamente. A tabela 2 contém o número de cores usadas e os tempo de execução de cada Algoritmo, *MP*, *MU*, *GI* e *GS*, aplicado ao grafo coluna-interseção. Incluímos também, os resultados do relatório anterior da coloração na ordem natural, cuja sigla é *ON*, para comparar com as novas ordens. Optamos por usar a ordem natural para comparar ao invéz de uma média de ordens aleatórias pelo fato que a orden natural tem um desempenho melhor. Apesar de as matrizes usadas não exibirem uma simetria explícita, a ordem natural tem, em geral, alguim significado. Repare na Figura 10 da matriz lp_maros_r7, é evidente que a ordem natural "desce" as diagonais. E nessa matriz, a coloração *ON* teve o melhor desempenho.

E na tabela 3 temos os resultados das versões distância-2 aplicados ao grafo bipartido. Incluímos nessas tabelas ρ_{max} e $\Delta + 1$ por serem o limite inferior e superior no número de cores usadas, respectivamente. Diferentemente do primeiro relatório, onde a ordem natural nos nós do grafo bipartido produzia a mesma coloração que uma ordem natural no grafo coluna-interseção, existem várias ordens *MP*, *MU*, *GI* e *GS*, e consequentemente a ordem *GS* efetuada pelo Algoritmo *GS2* no grafo bipartido não é necessáriamente a mesma ordem produzida pelo Algoritmo *GS*. Mas, mesmo não sendo exatamente a mesma ordem, o número de cores usadas pelas ordens *MP*, *MU*, *GI* e *GS* foi quase a mesma que as suas versões distância-2.

Tabela 1: Dados das Matrizes de teste

Matrizes	m	n	nnz	$\rho_{\max}$	$\rho_{\min}$
lp_cre_a	3,516	7,248	18,168	360	0
lp_dfl001	6,071	12,230	35,632	228	2
lp_ken_11	14,694	21,349	49,058	122	1
lp_ken_13	28,632	42,659	97,246	170	1
lp_pds_10	16,558	49,932	107,605	96	1
lp_maros_r7	3,136	9,408	144,848	48	5
lhr10	10,672	10,672	232,633	63	1
lp_pds_20	33,874	108,175	232,647	96	0
lp_cre_d	8,926	73,948	246,614	808	0
lp_cre_b	9,648	77,137	260,785	844	0
e30r2000	9,661	9,661	306,356	62	8
lhr14	14,270	14,270	307,858	63	1
lp_ken_18	105,127	154,699	358,171	325	1
e40r0100	17,281	17,281	553,956	62	8
cage11	39,082	39,082	559,722	31	3
lhr34	35,152	35,152	764,014	63	1
lhr71c	70,354	70,304	1,528,092	63	1
cage12	130,228	130,228	2,032,536	33	5
cavity13	2597	2597	76367	62	1
Ill_Stokes	20896	20896	191368	12	3
coater2	9540	9540	207308	63	1
airfoil_2d	14214	14214	259688	23	1
rat	3136	9408	268908	90	46
sinc12	7500	7500	294986	75	1

Figura 10: A matriz “lp_maros_r7”

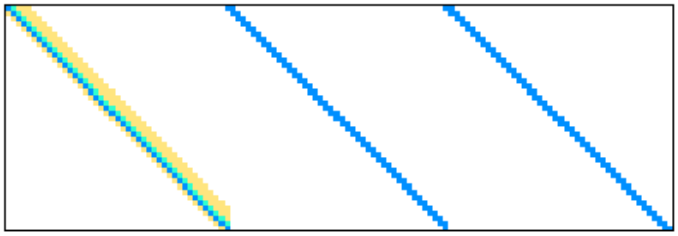


Tabela 2: O número de cores usados e tempo de execução de cada Algoritmo, *AMP*, *AMU*, *AGI* e *AGS*

			Total de cores usados					Tempo de execução				
	ρ_{max}	$\Delta+1$	*ON*	*AMP*	*AMU*	*AGI*	*AGS*	*ON*	*AMP*	*AMU*	*AGI*	*AGS*
lp_cre_a	360	455	360	360	360	360	360	0	0.015	0.703	0.703	0.968
lp_dfl001	228	424	228	228	228	228	228	0.015	0.032	0.578	2.547	2.797
lp_ken_11	122	139	130	122	125	125	122	0.031	0.078	2.61	5.328	5.797
lp_ken_13	170	187	176	170	174	172	171	0.047	0.219	11.422	25.343	27.031
lp_pds_10	96	107	96	96	96	96	96	0.031	0.156	9.609	10.14	10.875
lp_maros_r7	48	315	76	113	88	88	92	0.031	0.094	0.609	0.828	1.625
lhr10	63	102	65	65	63	64	64	0.032	0.047	1.969	0.515	0.938
lp_pds_20	96	116	96	96	96	96	96	0.062	0.579	47.484	32.937	35.063
lp_cre_d	808	1125	813	808	808	808	808	1.234	1.25	157.125	151.719	187.338
lp_cre_b	844	1169	845	844	844	845	844	3.766	1.781	109.14	163.687	205.686
e30r2000	62	210	65	83	70	71	72	0.032	0.062	0.906	0.469	1.297
lhr14	63	102	65	65	63	64	64	0.031	0.078	3.219	0.906	1.485
lp_ken_18	325	341	330	326	326	327	325	4.422	2.187	159.719	281.016	290.218
e40r0100_2	21	210	65	80	70	71	72	0.031	0.141	2.578	1.375	3.015
e40r0100	62	210	95	83	71	70	70	0.047	0.156	2.688	2.094	3.937
cage11	31	341	84	67	63	65	65	0.172	0.312	7.032	26.688	29.326
lhr34	63	102	65	65	63	64	64	0.063	0.328	16.906	5.907	7.317
lhr71c	63	102	65	65	64	64	64	0.172	1.094	69.687	16.109	20.22
cage12	33	401	95	72	68	74	73	1.235	2.157	70.328	295.843	308.656
cavity13	62	210	66	79	69	69	69	0.016	0	0.125	0.047	0.219
Ill_Stokes	12	64	28	29	26	27	28	0.063	0.125	3.344	4.234	4.983
coater2	63	335	98	88	80	79	80	0.063	0.11	0.89	0.359	1.234
airfoil_2d	23	74	30	38	29	31	30	0.031	0.094	1.687	0.766	1.281
rat	90	615	131	209	144	157	159	0.062	0.156	0.922	1.219	2.531
sinc12	75	1295	112	100	102	106	92	0.219	0.203	26.891	2.328	3.999
Total	3883	8751	4279	4351	4190	4221	4208	11.908	11.454	708.171	1033.107	1157.836

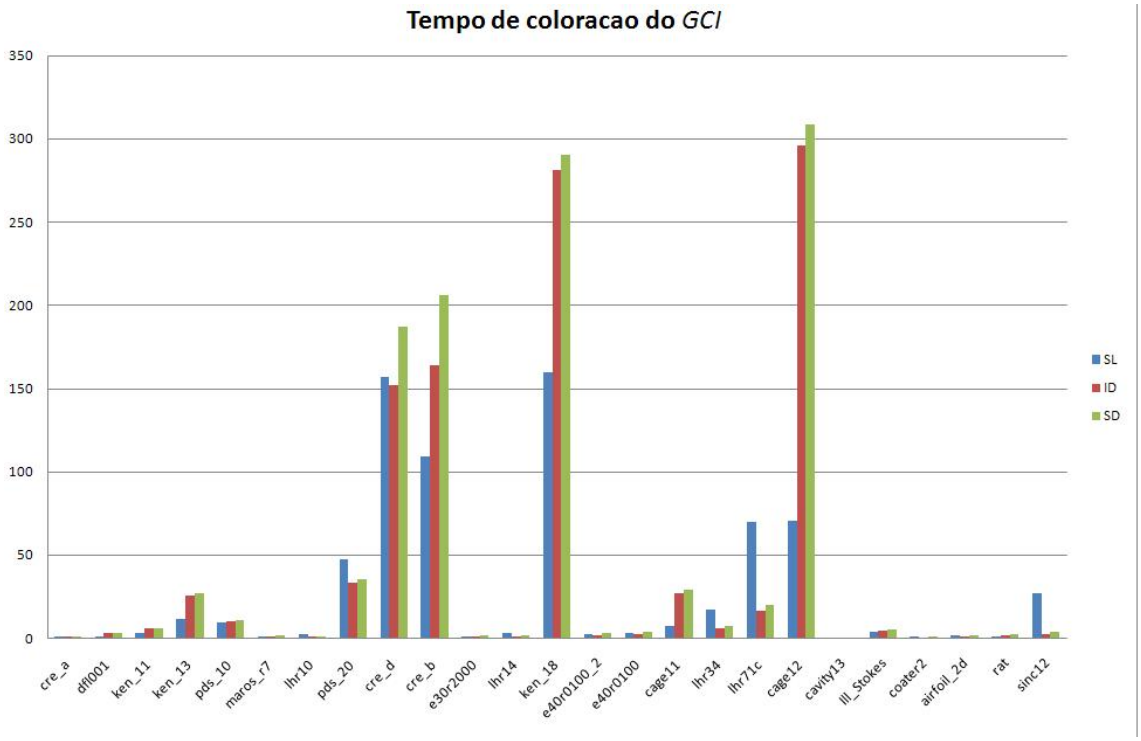


Figura 11: Tempo de exeção das ordens *MU*, *GI* e *GS*. *MP* não foi incluido por ser comparativamente muito pequeno

Em termos de número de cores usadas, repare na última linha, no total de cores nas tabelas 2 nem uma das colorações se sobressai. O melhor desempenho em termos de número de cores foi da ordem *MU* seguido por *GS*, *GI*, *ON* e *MP*, mas todas tiveram desempenho similar. E no grafo bipartido, tabela 3, tivemos um resultado análoga, com *MU2* usando menos cores seguido por *GS2*, *GI2*, *ON2* e *MP2*. Isso contradiz o esperado pelos autores de [7], onde citam que, devido a argumentos intuitivos, acreditam que a ordem *GS* terá melhor desempenho. Mas, além do fato que a ordem *GI* e *GS* são ótimas em grafos bipartidos, não há motivo teórico para o uso delas. Os Algoritmos diferem muito em termos de tempo de execução, sendo a *ON* e *MP* consideravelmente mais rápidas, seguidas por *MU*, *GI* e *GS*. Plotamos o tempo de execução desses últimos três na Figura 11, e das ordens *MU2*, *GI2* e *GS2* na Figura 12.

O fato que a *ON* (*ON2*) utilizou menos cores no total que a *MP* (*MP2*) foi um resultado inesperado. Como mencionamos anteriormente, essas matrizes apresentam uma certa estrutura de tal forma que a ordem natural tem um desempenho superior em relação a uma média de ordens aleatórias. Outro fato inesperado foi a execução mais rapida da Algoritmo *MP* em relação a *ON*. Uma explicação plausível é o tempo gasto na busca de uma cor mínima que não está na lista das cores proibidas. Quanto maior o número de vizinhos coloridos que um dado nó tem, maior será a lista proibida, e mais tempo será gasto. Os nós candidatos a terem as maiores listas proibidas são os nós de maior grau. E na ordem *MP*, os nós de maior

Tabela 3: O número de cores usados e tempo de execução dos Algoritmos, *AMP2*, *AMU2*, *AGI2* e *AGS2*

			Total de cores usados					Tempo de execução				
Matrizes	ρ_{max}	$\Delta+1$	*ON2*	*AMP2*	*AMU2*	*AGI2*	*AGS2*	*ON2*	*AMP2*	*AMU2*	*AGI2*	*AGS2*
lp_cre_a	360	455	360	360	360	360	360	0	0,03	0,78	0,75	1,06
lp_dfl001	228	424	228	228	228	228	228	0,031	0,05	0,61	2,55	2,78
lp_ken_11	122	139	130	122	125	124	122	0,031	0,08	2,49	5,56	5,89
lp_ken_13	170	187	176	170	171	171	170	0,05	0,28	10	25,66	27,03
lp_pds_10	96	107	96	96	96	96	96	0,031	0,17	9,24	10,22	10,86
lp_maros_r7	48	315	76	113	88	85	85	0,125	0,33	1,02	1,13	4,52
lhr10	63	102	65	65	63	64	64	0,218	0,53	6,58	1,06	6,23
lp_pds_20	96	116	96	96	96	96	96	0,079	0,63	46,42	33,27	35,23
lp_cre_d	808	1125	813	808	808	808	808	0,906	2,41	90,06	162,11	207,84
lp_cre_b	844	1169	845	844	844	845	844	0,906	2,39	87,16	165,94	203,44
e30r2000	62	210	65	83	71	71	71	0,219	0,53	2,11	1,03	6,61
lhr14	63	102	65	65	63	64	64	0,297	0,74	9,25	1,67	8,59
lp_ken_18	325	341	330	326	325	327	326	0,484	2,5	147,75	290,08	291,7
e40r0100_2	21	210	65	80	71	71	72	0,407	1,03	4,74	2,36	12,44
e40r0100	62	210	95	83	70	70	70	0,406	1,08	4,95	3,16	13,44
cage11	31	341	84	67	65	65	67	0,219	0,72	7,81	28,28	32,47
lhr34	63	102	65	65	63	64	64	0,735	1,92	31,7	7,77	24,73
lhr71c	63	102	65	65	63	64	64	1,437	4,28	97,89	19,98	55,27
cage12	33	401	95	72	68	75	73	0,953	3,91	74,27	308,23	335,82
cavity13	62	210	66	79	68	69	70	0,062	0,13	0,47	0,19	1,55
Ill_Stokes	12	64	28	29	26	26	26	0,078	0,23	3,61	4,44	5,23
coater2	63	335	98	88	79	79	82	0,11	0,41	1,39	0,69	3,89
airfoil_2d	23	74	30	38	29	32	30	0,11	0,38	2,11	1,06	3,36
rat	90	615	131	209	144	164	157	0,437	1,05	3,02	2,44	13,67
sinc12	75	1295	112	100	102	93	101	0,297	0,78	33,14	1,78	10,28
Total	3883	8751	4279	4351	4186	4211	4210	8,625	26,56	678,54	1081,38	1323,93

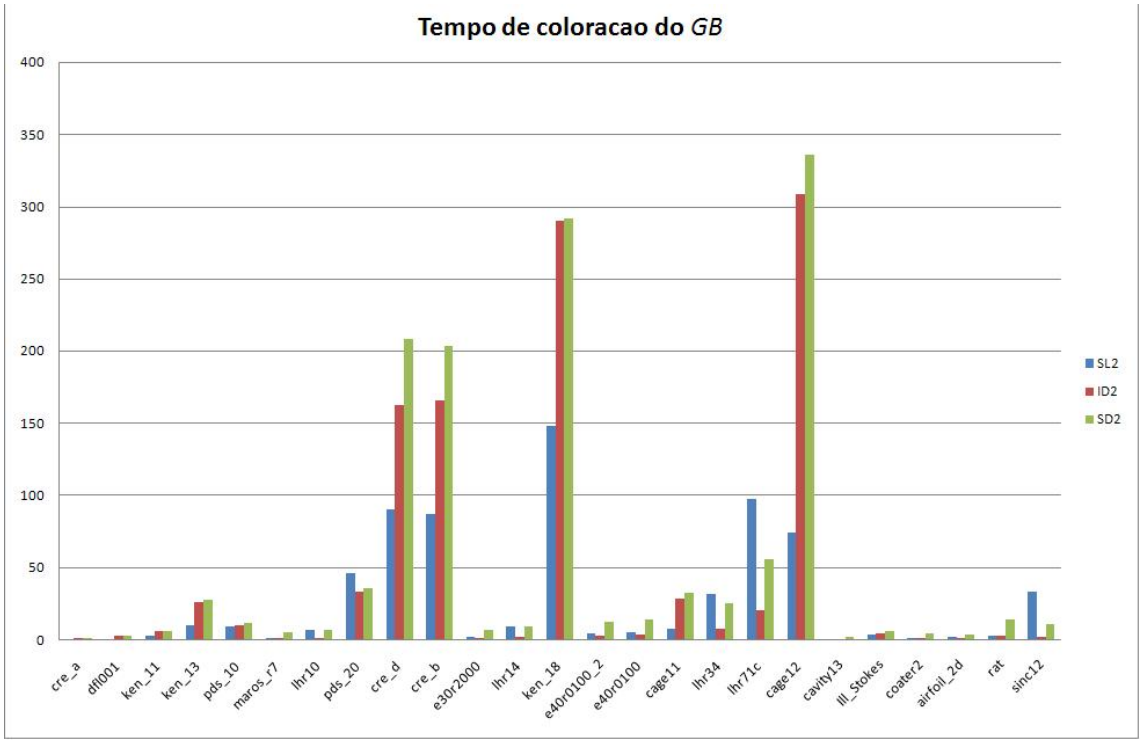


Figura 12: Tempo de exeção das ordens *MU2*, *GI2* e *GS2*. *MP2* não foi incluido por ser comparativamente muito pequeno

grau são coloridos primeiro, logo, provavelmente não vêm a ter muitos vizinhos coloridos, proporcionando uma lista proibida curta. Um resultado muito difícil de prever com a análise assintotíco prévia dos Algoritmos. Mas esse tempo ganho pelo *MP2* não compensou o tempo gasto para realizar a busca distância-2 no grafo bipartido, tabela 3, e a ordem *MP2* teve um execução mais demorado que a *ON2*.

Comparando a ordem *MU* com a ordem *GI* e *GS*. A ordem *MU* usou menos cores e menos tempo de execução no total. Logo, de acordo com essa amostra, o Algoritmo *MU* é melhor. O mesmo se verifica para as versões distância-2. Entre as colorações distância-1 no *GCI* e distância-2 no grafo bipartido há pouca distinção. O número de cores usadas por cada método difere pouco da versão distância-2. E o tempo de coloração ficou similar, repare a comparação percentual na Figura 13, onde 100% é a soma das duas versões. Vemos que, tirando *MP* e *MP2*, as colorações ficaram em torno de 50%. Teria sido razoável esperar que as colorações no grafo bipartido fossem mais demoradas, por fazerem buscas pela vizinhança distância-2. Porém, como concluímos no relatório anterior, o grafo bipartido associado a uma matriz tende a ter menos arestas, e consequentemente ser mais esparso, que o grafo coluna-interseção associado. E esparso o suficiente para compensar as buscas de distância-2.

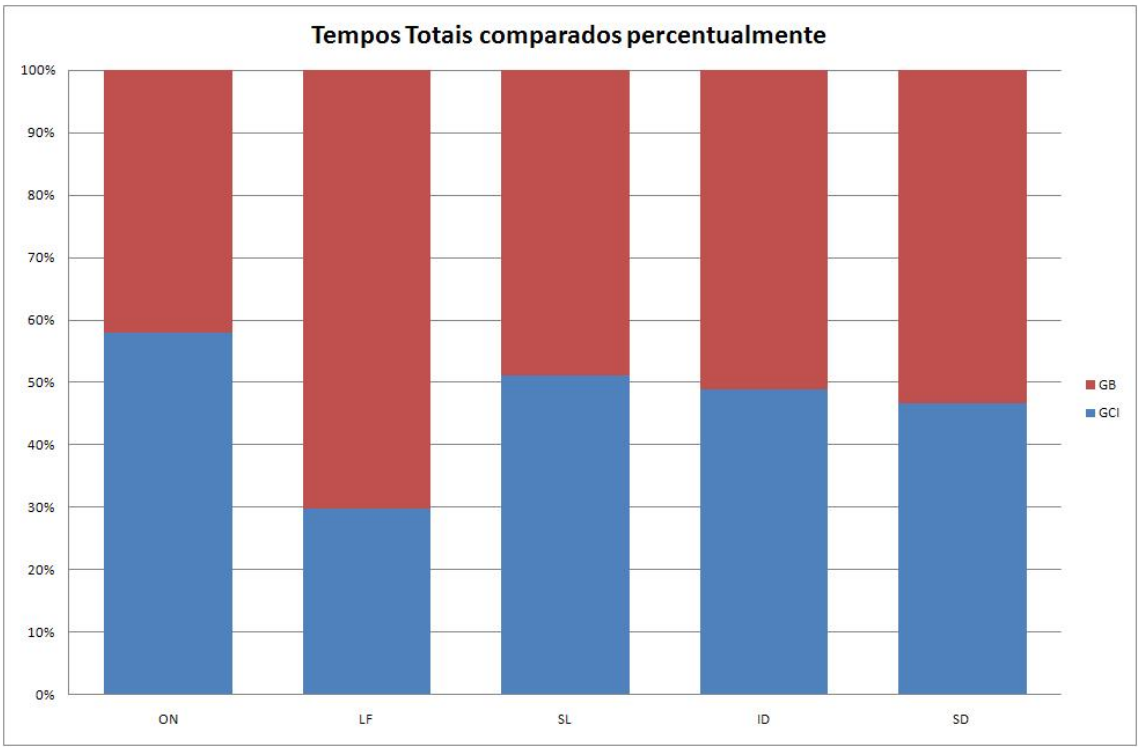


Figura 13: O tempo total de cada ordem comparado ao seus versão distância-2. 100% equivale ao soma dos dois

10 Conclusões sobre ordens de coloração

Baseado nessa amostra, *MU* teve desempenho superior às ordens *GI* e *GS*. Resultado análogo para *MU2*, *GI2* e *GS2*. Enquanto a ordem *ON* superou *MP* em número de cores e empatou em tempo de execução, a *ON2* superou a *MP2* em cores e tempo de execução. Isto difere dos resultados reportados por Coleman e Moré [3], onde em termos de números de cores usadas a *GI* e *MU* empataram seguidas por *MP* e *ON*[1]. E, se não fosse por duas matrizes lp_maros_r7 e rat, nossos resultados também confirmariam que a ordem *MP* usa menos cores que a ordem *ON*. Repare as matrizes lp_maros_r7 e rat nas Figuras 14 e 10, as semelhanças nas estruturas de esparsidades são evidentes. E as duas matrizes derivam de problemas sequencias de programação linear. Com uma investigação do sucesso da ordem natural nesses exemplos, criamos um algoritmo híbrido entre a ordem *MU* e *ON*. No total esse híbrido foi ligeramente mais rápido, e usou 5 cores a menos no total. Talvez um híbrido que incorporasse um análise de casos teria sucesso.

No relatório anterior, devido aos resultados Gebremedhin et al [7], comprovados pelos nossos testes computacionais, estávamos inclinados a acreditar que o método associado ao grafo bipartido era mais eficiente em termos de memória alocado e tempo gasto em relação

[1]Os autores não testaram a *GS* por acreditarem ter um tempo assintótico inaceitável, enquanto nossos análises teóricos e resultados indicam que *GS* tem um tempo de execução proporcional a *GI*

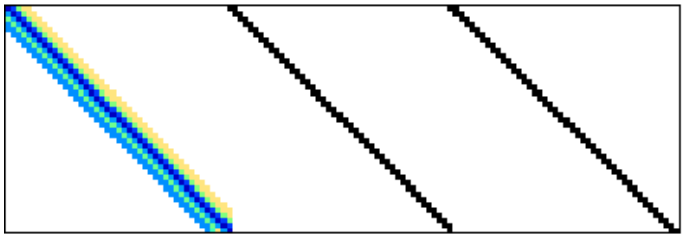


Figura 14: A padrão de esparsidade da matriz “rat”.

ao método associado ao *GCI*. Porém, os dois métodos utilizam o mesmo número de cores quando atravesamos os nós na mesma ordem. Faltava conferir a capacidade de melhorias em termos de número de cores para cada método. As diversas ordens já foram implementadas para o *GCI* por Coleman e Moré [3] mas não há relato da implementação no grafo bipartido de ordens modificadas para distância-2. E a priori não sabíamos se o tempo ganhando na construção mais veloz do grafo bipartido iria ser o suficiente para tornar o método como um todo mais eficaz. Mas dado que as colorações usando diversas ordens produziram resultados parecidos no grafo bipartido e no *GCI*, concluimos que o método associado ao grafo bipartido continua o favorito entre os dois.

PARTE II:
Partição simetricamente ortogonal da matriz Hessiana

1 Computando a hessiana

Seja $F : \mathbb{R}^n \to \mathbb{R}$ uma função duas vezes diferenciável. Sua matriz hessiana é a matriz simétrica $n \times n$ $\nabla^2 F$ cujo elemento na posição (i, j) é

$$\nabla^2 F_{ij} = \frac{\partial^2 F}{\partial x_i \partial x_j}$$

.

Suponhamos que temos o gradiente de F ∇F. Podemos obter uma aproximação para $\nabla^2 F$ usando uma fórmula de diferenças finitas, por exemplo a diferença avançada:

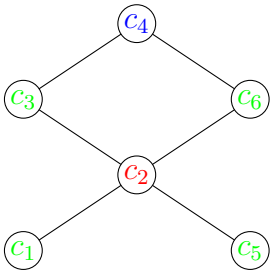
$$\nabla^2 F e_i = \frac{\partial}{\partial x_i} \nabla F \approx \frac{\nabla F(x + h e_i) - \nabla F(x)}{h}$$

Então, a princípio, seria necessário fazer n avaliações da função ∇F para se obter uma aproximação da matriz $\nabla^2 F$. Nosso objetivo é diminuir o número de avaliações da função ∇F. Para fazer isso seja d_i um vetor binário, combinação dos vetores $\{e_1, e_2, \ldots, e_n\}$. Dada a estrutura de esparsidade da matriz $\nabla^2 F$, queremos encontrar o menor número de vetores binários $d_1, d_2, \ldots, d_p$ tais que os produtos $d_1 \nabla^2 F$, $d_2 \nabla^2 F$, ..., $d_p \nabla^2 F$ nos permitem determinar $\nabla^2 F$. Conhecendo esses vetores, podemos obter $\nabla^2 F$ fazendo os p avaliações:

$$\nabla^2 F d_i \approx \frac{\nabla F(x + h d_i) - \nabla F(x)}{h}$$

1.1 Grafo de adjacência

Seja A uma matriz simétrica, cujos elementos da diagonal são não nulos e cujas colunas são indexadas por $\{c_1, c_2, \ldots, c_n\}$. A maioria das matrizes hessianas tem diagonal não nula, então essa suposição é razoável. O seu grafo de adjacência é $G(A) = (V, E)$, onde $V = \{c_1, c_2, \ldots, c_n\}$ e $(c_i, c_j) \in E$ se $i \neq j$ e o elemento $a_{ij} \neq 0$. Repare que, tanto o elemento a_{ij} como o elemento a_{ji} são representados pela aresta (c_i, c_j) portanto essa estrutura aproveita a simetria da matriz.



$$\begin{bmatrix} a_{11} & a_{12} & 0 & 0 & 0 & 0 \\ a_{21} & a_{22} & a_{23} & 0 & a_{25} & a_{26} \\ 0 & a_{32} & a_{33} & a_{34} & 0 & 0 \\ 0 & 0 & a_{43} & a_{44} & 0 & a_{46} \\ 0 & a_{52} & 0 & 0 & a_{55} & 0 \\ 0 & a_{62} & 0 & a_{64} & 0 & a_{66} \end{bmatrix}$$

Figura 15:

Relembramos que duas colunas são estruturalmente ortogonais se não tiverem elementos não nulos em uma mesma linha. Afirmamos que duas colunas são estruturalmente ortogonais se e somente se estão a uma distância maior que dois no grafo de adjacência.
Prova: *Se c_i e c_j são estruturalmente não ortogonais, então existe uma linha k que contêm dois elementos não nulos a_{ki} e a_{kj}. Se $k = i$ ou $k = j$ então existe a aresta (c_i, c_j) logo c_i e c_j são vizinhos. Se $k \neq i$ e $k \neq j$ então existem as arestas (c_i, c_k) e (c_j, c_k) consequentemente c_i e c_j são ligados através dessas duas arestas. Portanto, se c_i e c_j estão a uma distância maior que dois então as colunsa correspondentes são estruturalmente ortogonais.*

Suponha aogra que c_i e c_j são estruturalmente ortogonais. Isso implica que não existe k tal que $a_{ki} \neq 0$ e $a_{kj} \neq 0$ e com isso concluímos que: (1) $a_{ij} = 0$ e $a_{ji} = 0$, logo não existe a aresta (c_i, c_j); (2) não existe nenhum caminho c_i, c_k, c_j para todo k, $k \neq i$ e $k \neq j$. Consequentemente, c_i e c_j estão a uma distância maior do que dois. ■

Dada a simetria da hessiana, podemos permitir que duas colunas estejam no mesmo grupo mesmo sendo estruturalmente não ortogonais. Repare novamente a Figura 15. Veja que, se as colunas c_1 e c_3 pertencem ao mesmo grupo, não poderíamos determinar os elementos a_{21} e a_{23} diretamente. Porém, a coluna c_2 contém a_{12} e a_{32} que são iguais a a_{21} e a_{23}. Contudo que c_2 pertencer a um grupo que não contêm outros não nulos nas linhas 2 e 3, podemos permitir que as colunas c_1 e c_3 pertencem ao mesmo grupo.

Uma partição das colunas de uma matriz A é simetricamente ortogonal, se para todo não nulo a_{ij} ou (1) o grupo que contém c_j não contém nemhum outro não nulo na linha i ou (2) o grupo que contém o a_{ji} não contém nemhum outro não nulo na linha j.

E que coloração do grafo de adjacência induziria uma partição simetricamente ortogonal? Primeiramente, note que os elementos da diagonal são únicos, e precisam pertencer a um grupo que não contém outro não nulo na mesma linha. Isso exige que vizinhos tenham cores diferentes no grafo de adjacência. Em uma partição simetricamente ortogonal, já vimos que podemos permitir que dois vizinhos distância-2 tenham a mesma cor, repare os nós c_2 e c_3 da Figura 16. Com isso, o nó que os conecta, c_5, deve pertencer a um grupo onde a_{25} e a_{35} são os únicos não nulos nas linhas 3 e 2. Logo, os vizinhos de c_2 e c_3 não podem pertencer ao mesmo grupo que c_5. Ou seja, c_5 precisa ter uma cor distinta da cor de c_1 e c_4. Com isso,

$$\begin{bmatrix} a_{11} & a_{12} & 0 & 0 & 0 \\ a_{21} & a_{22} & 0 & 0 & a_{25} \\ 0 & 0 & a_{33} & a_{34} & a_{35} \\ 0 & 0 & a_{43} & a_{44} & 0 \\ 0 & a_{52} & a_{53} & 0 & a_{55} \end{bmatrix}$$

Figura 16:

mostramos que são necessárias pelo menos três cores para colorir caminhos de comprimento três, e que vizinhos tenham cores distintas. Isto motiva a seguinte definição.

Um mapeamento $\Phi : V \rightarrow \{1, 2, \ldots, p\}$ é uma coloração estrela do grafo $G(V, E)$ se Φ é uma coloração distância-1 e todo caminho por quatro vértices usa pelo menos três cores.

Agora falta mostrar que uma coloração estrela induz uma partição simetricamente ortogonal. Seja a_{ij} um elemento não nulo da matriz hessiana. Se $i = j$, então a_{ij} é um elemento da diagonal e dado que c_j tem uma cor diferente do seus vizinhos, então não existe outro não nulo na linha i. Se $i \neq j$, vamos supor que nem a_{ij} e nem a_{ji} pertencem a grupos que podem ser diretamente determinados. Logo existe c_k e c_r tal que $\Phi(c_k) = \Phi(c_j)$ com $a_{ik} \neq 0$ e $\Phi(c_r) = \Phi(c_i)$ com $a_{jr} \neq 0$. Dado que $a_{ik} \neq 0$ temos no grafo de adjacência que $(c_k, c_i) \in E$, e da mesma forma $a_{ij} \neq 0$ e $a_{jr} \neq 0$ implica que $(c_i, c_j) \in E$ e $(c_j, c_r) \in E$,

Figura 17:

veja a Figura 17. O resultado é um caminho por quatro nós que usa duas cores, logo não é uma coloração estrela e por contradição demonstramos que uma coloração estrela induz uma partição simetricamente ortogonal. Para formalizar nosso problema anunciamos:

Problema da Hessiana: Dado um grafo de adjacência que representa a estrutura de esparsidade de uma matriz hessiana com diagonal não nula, encontre uma coloração estrela que utilize o menor número de cores.

1.2 O número cromático simetrica

Vamos investigar a relação entre o número cromático de um grafo $\chi(G)$, e o número cromático simetrica $\chi_\sigma(G)$, o menor número de cores que podemos usar para efetuar uma coloração estrela. $\chi_\sigma(G)$ é de difícil determinação. Coleman e Moré [4] demonstram que o problema de obter o número cromático simetrica para grafos genéricos é NP-hard, e até o problema de obter o número cromático simetrica para grafos bipartidos é NP-hard. Isso também reflete na dificuldade de obter bons limites inferiores e superiores. Mas, dado que uma coloração estrela é uma coloração distância-1 com um critério a mais, temos:

$$\chi(G) \leq \chi_\sigma(G)$$

Dado que toda coloração distância-2 induz uma partição estruturalmente ortogonal na hessiana, e uma partição estruturalmente ortogonal é uma partição simetricamente ortogonal, que por sua vez é uma coloração estrela:

$$\chi(G) \leq \chi_\sigma(G) \leq \chi(G^2)$$

Aqui $G^2 = (V, \widetilde{E})$ é o quadrado do grafo $G = (V, E)$, nesse grafo $(u, v) \in \widetilde{E}$ se existe um caminho entre u e v de comprimento $l \leq 2$. Portanto a coloração distância-1 de G^2 é uma coloração distância-2 de G. Com isso, podemos usar o limite inferior do $\chi(G)$ e o limite superior do $\chi(G^2)$. Porém, o limite inferior de $\chi(G)$ é o tamanho da maior clique, denotado por $\omega(G)$, o que é custoso obter. Fertin et al [8], demonstraram o seguinte limite inferior para um grafo $G = (V, E)$, $|V| = n$ e $|E| = m$:

$$\text{Seja } \Lambda = 4n(n-1) - 8m + 1$$

$$\chi_\sigma(G) \leq \frac{2n + 1 - \sqrt{\Lambda}}{2}$$

Mas esse limite se mostrou pouco útil para medir a qualidade das colorações, sendo um limite inferior com muita folga. Portanto, não teremos uma comparação absoluta da qualidade das coloração, e sim uma comparação relativa entre dois Algoritmos diferentes que calculam uma coloração estrela.

2 Os Algoritmos de coloração estrela

Implementamos duas heurísticas. O primeiro foi criado por Gebremedhin [7], e o segundo por Toint e Powell [15] em termos de matrizes, e traduzido para grafos por Gebremedhin [7]. Enquanto a primeira heurística visita até os vizinhos distância-3, o segundo visita até os vizinhos distância-2. Enquanto o primeiro sacrifica tempo de execução para fazer uma coloração mais eficiente, o segundo faz o oposto.

2.1 O primeiro Algoritmo coloração estrela

No Algoritmo 14 temos o primeiro Algoritmo *ESTRELAd3*, assim denominado porque visita até os vizinhos distância-3 de cada nó, e por isso a heurística é $O(|V|\delta_3)$. Lembrando que $d_3(v_i)$ é o grau-3 do nó v_i, o número de nós a distância-3 da v_i. E δ_3 é a média dos graus-3. Na Figura 18, temos a esquematização de como o Algoritmo decide colorir o nó v_i. Um P no nó indica que a cor deste nó é proibida para v_i. Iniciamos um vetor de cores proibidas, *forbiddenColors*, com $\min\{\Delta^2, n\}$ posições, linha 2. Na linha 4, visitamos os nós adjacentes w, e na linha 5 asseguramos que v_i tem uma cor distinta dos seus vizinhos. Na Figura 18, os nós nas colunas w são os vizinhos de v_i.

ESTRELAd3

(1) **Seja** $v_1, v_2, \ldots, v_n$ uma ordenação dos nós pertencentes a V.
(2) **Incializa** uma lista *forbiddenColors* com $a \notin V$
(3) **Para** i de 1 até n
(4) **Para** w colorido $\in N(v_i)$
(5) **Se** w é colorido
(6) $forbiddenColors\, cor[(w)] \leftarrow v_i$
(7) **Para** cada x colorido $\in N(w)$, $x \neq v_i$
(8) **Se** w não é colorido
(9) $forbiddenColors[cor(x)] \leftarrow v_i$
(10) **Se não**
(11) **Para** cada y colorido $\in N(x)$,$w \neq y$
(12) **Se** $cor(y) = cor(w)$
(13) $forbiddenColors[cor(x)] \leftarrow v_i$
(14) $cor(v_i) \leftarrow \min\{c \in N : forbiddenColors[c] \neq v_i\}$

Algoritmo 14: Coloração *ESTRELAd3*

Na linha 7, todos os vizinhos coloridos de w são visitados, ou seja, os nós coloridos nas colunas x da Figura 18. Se w não estiver colorido, linha 8, incluímos a cor do nó na coluna x nas cores proibidas, linha 9. Veja que as cores dos nós na coluna x à esquerda são proibidas. Mas se w está colorido, linha 10, precisamos verificar se há outro nó a distância-2 de w com a mesma cor, repare os nós verdes nas colunas w e y da Figura 18. Nesse caso de $cor(y) = cor(w)$, linha 11, v_i não pode assumir a mesma cor do nó entre y e w, caso contrário teríamos um caminho entre quatro nós com duas cores, logo a cor $cor(x)$ entra na lista proibida, linha 13.

Essa Algoritmo permite v_i ter a mesma cor de um vizinho distância-2 x, somente quando garante que x não está no meio de dois nós com a mesma cor ($cor(w) \neq cor(y)$). Veja que na esquematização, a cor vermelha do nó na coluna x é permitido, dado que os nós nas colunas

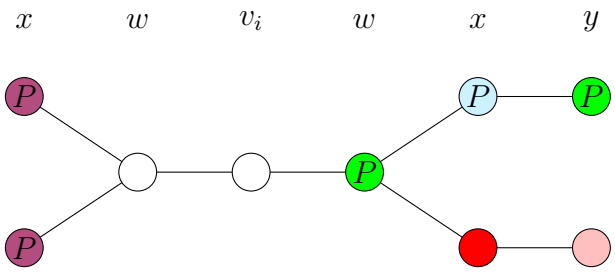


Figura 18: Um exemplo de uma iteração do Algoritmo 14

w e y tem cores diferentes (cores verde e rosa, respectivamente).

2.2 O segundo Algoritmo coloração estrela

A segunda heurística *ESTRELAd2*, tem seu pseudo-código no Algoritmo 15. Como o nome indica, essa heurística executa buscas até vizinhos de distância-2 logo tem tempo assintótico de $O(|V|\delta_2)$.

ESTRELAd2
(1) **Seja** $v_1, v_2, \ldots, v_n$ uma ordenação dos nós pertencentes a V.
(2) **Incializa** uma lista *forbiddenColors* com $a \notin V$
(3) **Para** i de 1 até n
(4) **Para** cada $w \in N(v_i)$
(5) **Se** w é colorido
(6) $forbiddenColors\, cor[(w)] \leftarrow v_i$
(7) **Para** cada x colorido $\in N(w)$
(8) **Se** w não é colorido
(9) $forbiddenColors[cor(x)] \leftarrow v_i$
Se não $cor[x] < cor[w]$
(10) $forbiddenColors[cor(x)] \leftarrow v_i$
(11) $cor[v_i] \leftarrow \min\{c \in N : forbiddenColors[c] \neq v_i\}$

Algoritmo 15: Coloração *ESTRELAd2*

Esse Algoritmo usa o fato que cores são representadas por números. E um nó v_i, Figura 19, pode ter a mesma cor que seu vizinho distância-2 x, se a cor do nó que os conectam é menor que a cor de x, $cor(w) < cor(x)$. Veja na Figura 19 a cor azul é representado pelo número 1 e a cor vermelha pelo número 2.

Figura 19: Um exemplo de uma iteração do Algoritmo 15

Tabela 4: Dados das Matrizes de teste

Matrizes	$n = \|V\|$	nnz	definida positiva	T const.(s)	$\|E\| = nnz - n$
commanche	7920	31680	não	0	11880
msc01050	1050	29156	sim	0.016	14053
bloweybq	10001	69991	sim	0.032	29995
linverse	11999	95977	não	0.031	41989
SiNa	5743	198787	não	0.109	96522
benzene	8219	242669	não	0.094	117225
vibrobox	12328	342828	não	0.156	165250
crplat2	18010	960946	sim	0.39	471468
tsyl201	20685	2454957	sim	1.609	1217136
Total	95955	4426991	sim	2.437	2165518

No grafo a esquerda o nó v_i não foi colorido ainda, e pode ser colorido com a cor 2. No grafo a direita a cor 1 é proibído para o nó y por ser menor que 2. Se o nó w não fosse colorido, a cor do x seria proibído, linhas 8 e 9 do algortimo 15.

3 Resultados númericos do *ESTRELAd2* e *ESTRELAd3*

Os Algoritmos descritos foram implementados em linguagem C++ e executados em um Intel Pentium 4, 3.00GHz com 896MB de RAM, no sistema Microsoft Windows XP 2002 usando o compilador MingW32.

As matrizes de teste provieram do mesmo banco de matrizes esparsas que os autores de [7], o *University of Florida Sparse Matrix Collection* [6]. Na tabela 4 as matrizes estão em ordem crescente de número de não nulos. Incluimos dados gerais das matrizes e grafos de adjacência provinientes, onde n é o número de colunas, e linhas por serem simetricas, e também o número de nós no grafo. nnz é o número de não nulos. Dado que, nas proximidades de mínimos locais as matrizes hessianas tendem a ser definidas positivas, escolhemos que uma proporção considerável das nossas matrizes de teste sejam definidas positivas. O número de

arestas tem uma relação direta com nnz e n, dado que para cada par $(a_{ij}\ ,\ a_{ji})\ i \neq j$ de não nulos existe um aresta $(c_i, c_j) \in E$. Logo $|E| = (nnz - n)/2$. Colocamos o tempo de construção em segundos do grafo na coluna T const.

Tabela 5: O número cores usados, e tempo de execução em segundos.

Matrizes	# cores		Tempo(s)			
Matrizes	d2	d3	d2	d3	Tempo %	Cores %
commanche_dual	7	6	0.016	0.000	-	0.167
msc01050	183	171	0.016	0.515	32.19	0.070
bloweybq	6	6	0.688	9.015	13.10	0.000
linverse	9	9	0.016	0.063	3.94	0.000
SiNa	308	239	0.234	27.125	115.92	0.289
benzene	100	73	0.14	3.266	23.33	0.370
vibrobox	131	100	0.25	7.312	29.25	0.310
crplat2	88	81	0.516	30.172	58.47	0.086
tsyl201	204	193	4.093	511.868	125.06	0.057
Total	1036	878	5.969	589.336	401.26	1.182

Na tabela 5, abreviamos os Algoritmos *ESTRELAd2* e *ESTRELAd3* para d2 e d3. Os tempos listados incluim as execuções das colorações em segundos, mas não o tempo de construção do grafo de adjacência. E a coluna # cores é o número de cores usados. Incluímos também uma comparação relativa entre os dois Algoritmos. Nas colunas Tempo % e Cores % temos a diferença relativa dos tempos de execução $T_{d3} - T_{d2}/T_{d2}$ e das cores usadas: $K_{d3} - K_{d2}/K_{d2}$. Temos também uma representação gráfica do número de cores usadas na Figura 20. Em média *ESTRELAd2* usou 18% de cores a mais. Em média *ESTRELAd3* demorou 99 vezes mais que o *ESTRELAd2*.

Devido a esse contraste grande entre os tempos de execucções, uma representação gráfica seria pouco informativa. Esse resultado é tambem em relação ao resultados do Gebremedhin et al [7]. Nesta referência, *ESTRELAd3* demorou, em média, 3.7 vezes a mais. Mas as matrizes e grafos usados em [7] são consideravelmente menos densos em relação aos nossos. Não existe uma medida única de densidade de grafos, mas se utilizamos a medida $\frac{|E|}{|V|}$, a densidade média dos grafos em [7] é de 5.5 e dos nossos é 16.8. Possivelmente com nossos grafos mais densos expomos mais a diferença do tempo assintótico dos Algoritmos.

4 Conclusões

O Algoritmo *ESTRELAd3* usou menos cores mas demorou consideravelmente mais nas matrizes maiores e mais densos. Mas a escolha de qual Algoritmo usar em um problema real depende de diversos fatores. Sabemos que, quanto menos cores, menos cáculos para obter

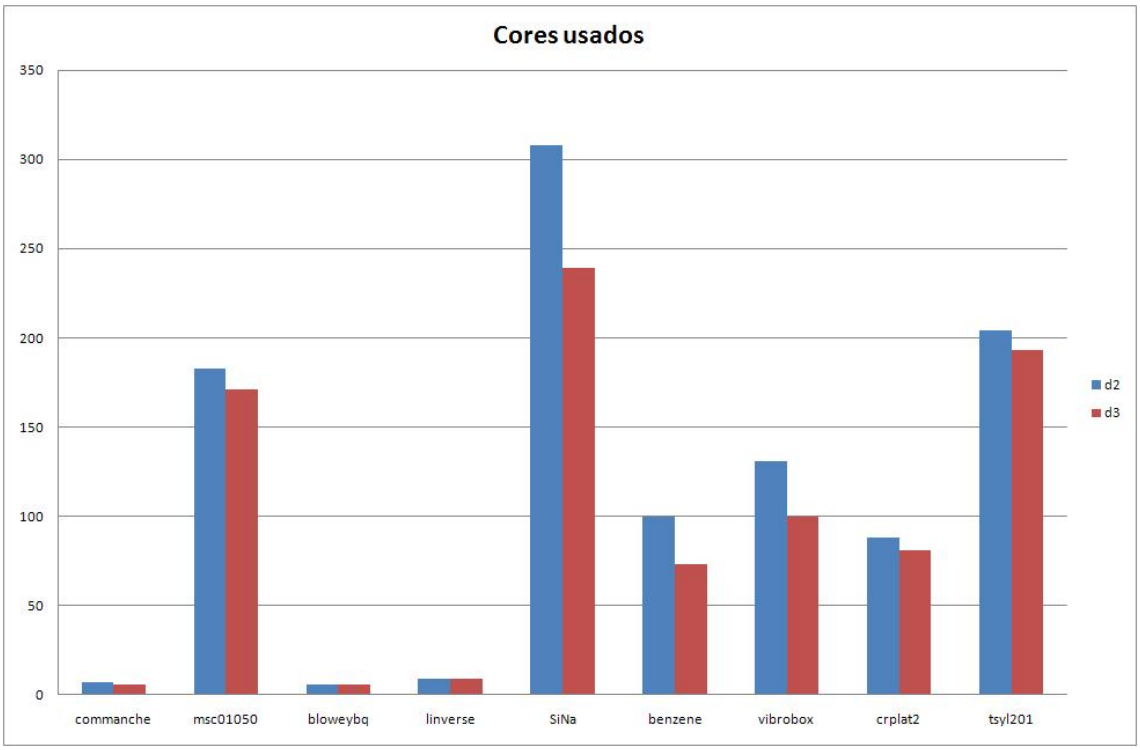


Figura 20: O número de cores usadas para os Algoritmos *ESTRELAd2* e *ESTRELAd3*.

uma aproximação da matriz hessiana. Logo, menos cores se traduz em menos tempo de execução. E será que o tempo poupado em menos avaliações compensa o tempo perdido em usar a heurística *ESTRELAd3*? Um aplicação desse modelo é na minimização usando o método de Newton. A cada iteração é preciso obter uma aproximação da matriz hessiana. Se a matriz hessiana mantém a sua estrutura de esparsidade durante a execução inteira, poderíamos aplicar a heurística *ESTRELAd3* apenas uma vez, e desfrutar da obtenção mais veloz da matriz hessiana em todas as iterações. Porém, se a própria estrutura de esparsidade da hessiana muda frequentemente, talvez a heurística *ESTRELAd2* seria mais interesante.

Mas, como um todo, os métodos tiveram êxito no seus propósitos. Antes era necessário 7920, 10001 e 11999 avaliações para obter as matrizes commanche_dual, bloweybqe e linverse. Após a execução do *ESTRELAd2* serão necessárias apenas 7, 6 e 9, respectivamente.

Referências

[1] D. Brélaz. "New methods to color the vertices of a graph". Comm. ACM, 22 (1979), 251-256.

[2] R.L. Brooks; "On Colouring the Nodes of a Network", *Proc. Cambridge Philos. Soc.*, 37 (1941), 194–197.

[3] T.F. Coleman, J.J. Moré; "Estimation of sparse Jacobian matrices and graph coloring problems", *SIAM J. Numer. Anal.*, 20 (1983), 187–209.

[4] T.F. Coleman and J.J. Moré; "Estimation of sparse Hessian matrices and graph coloring problems", *Math. Program.*, 28 (1984), 243-270.

[5] A.R. Curtis, M.J.D. Powell, J.K. Reid; "On the estimation of sparse Jacobian matrices", J. Inst. Math. Appl., 13 (1974), 117-119.

[6] T. Davis; "University of Florida Sparse matrix collection", Technical Report disponível em *http://www.cise.ufl.edu/research/sparse/matrices/*.

[7] A.H. Gebremedhin, F. Manne, A. Pothen; "What color is your Jacobian? Graph coloring for computing derivatives", *SIAM Review*, 47 (2005), 629–705.

[8] G. Fertin, A. Raspaud, B. Reed, "On star coloring of graphs in Graph-Theoretic Concepts in Computer Science", 27th International Workshop, Lecture Notes in Comput. Sci. 2204, Springer-Verlag, Berlin, 2001, 140-153.

[9] A.H. Gebremedhin, F. Manne, A. Pothen. "Graph coloring in optimization revisited", Reports in informatics, Report NO 226, 23 p. Department of Informatics, University of Bergen, January 2002.

[10] D. Goldfarb, P.L. Toint; "Pptimal estimation of Jacobian and Hessian matrices that arise in Finite Difference Calculations", *Math. Comp.*, 43 (1984), 69–88.

[11] D.W. Matula, G. Marble, J. Isaacson; "Graph coloring algorithms", in Graph Theory and Computing, R. Read, ed., Academic Press, New York, 1972, 109-122.

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

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

[14] *http://math.nist.gov/MatrixMarket/*

[15] M.J.D. Powell e P.L. Toint "On the estimation of sparse Hessian matrices", SIAM J. Numer. Anal., 16 (1979), 1060-1074.