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

# 1 Introdução

Nesse relatório apresentaremos dois métodos do artigo "What Color is your jacobian" [4] que procuram tornar mais eficiente a aproximação numérica, via fórmula centrada, de matrizes jacobianas. Expomos e comparamos dois métodos que utilizam modelagens via grafos e coloração de grafos. Forneceremos o arcabouço de duas heurísticas baseadas nos métodos para justificar a escolha de uma estrutura de dados. Com a estrutura de dados determinada, fornecemos detalhes da implementação. As heurísticas foram programadas e sua eficiência relativa testada em um conjunto de matrizes disponível na Internet, utilizado também em [4]. Finalmente, comparamos nossos resultados computacionais aos reportados em [4].

# 2 O problema

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. Utilizando a fórmula avançada[1] para a discretização de derivadas temos uma aproximação para a $k$-ésima coluna de $J$:

$$J(x)e_k = \frac{\partial}{\partial x_k}F(x) \approx \frac{F(x + he_k) - F(x)}{h}.$$

Então, a princípio, seria necessário fazer $n$ avaliações da função $F$ (estamos supondo $F(x)$ conhecida) para se obter uma aproximação da matriz $J$.

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

Figura 1: A matriz $\widehat{J}$ é um exemplo de jacobiana extraída de [4]

No entanto, se $J$ é esparsa, o cálculo de sua aproximação pela fórmula avançada pode ser feito de forma mais eficiente. Para ilustrar, tomemos um exemplo de [4], com $n = m = 5$, a matriz $\widehat{J}$ reproduzida na Figura 1. O truque consiste em particionar a matriz jacobiana em subconjuntos de colunas estruturalmente ortogonais, ou seja, que não possuem dois elementos não-nulos em uma mesma linha. No exemplo da Figura 1, há três subconjuntos estruturalmente ortogonais: as colunas 1 e 3, as colunas 1 e 4 e as colunas 3 e 5. Então, se

[1] Na prática, utilizaremos a fórmula centrada para a discretização. A fórmula avançada nesse exemplo serve para fins didáticos.

calcularmos a fração

$$\frac{F(x + he_1 + he_3) - F(x)}{h},$$

obteremos simultaneamente os elementos das colunas 1 e 3, ao "custo" de apenas uma avaliação de $F$. Com essa informação em mãos, o trabalho de calcular a jacobiana $\widehat{J}$ pode ser reduzido a calcular a versão compacta de $\widehat{J}$ exibida na Figura 2. Nessa matriz compacta, colunas estruturalmente ortogonais são agrupadas em uma única coluna, por exemplo, a coluna 1 da Figura 2 é a soma das colunas 1 e 4 de $\widehat{J}$.

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

Figura 2: Uma versão compacta da matriz $\widehat{J}$ da Figura 1

Portanto, se conseguirmos particionar o conjunto de colunas no menor número de subconjuntos de colunas estruturalmente ortogonais, tornaremos o cálculo mais eficiente, isto é, com o menor número possível de avaliações da função $F$. Isso implica que o problema do cálculo eficiente de uma aproximação de uma matriz jacobiana é equivalente ao:

Encontrar uma partição das colunas de uma matriz no menor número de grupos estruturalmente ortogonais.

Curtis et al. [3] foram os primeiros a usar a estrutura de esparsidade da matriz jacobiana para diminuir o número de avaliações a serem feitas. Resolveremos esse problema para jacobianos sem uma estrutura de esparsidade específica, utilizando uma modelagem através de grafos. Para jacobianos com padrões previsíveis dos não nulos, existem outros métodos com garantia de otimalidade, como o método apresentado por Goldfarb [7] para jacobianos que provieram de métodos de diferenças finitas. Antes de mais nada, apresentaremos algumas das notações a serem usadas.

# 3 Notações

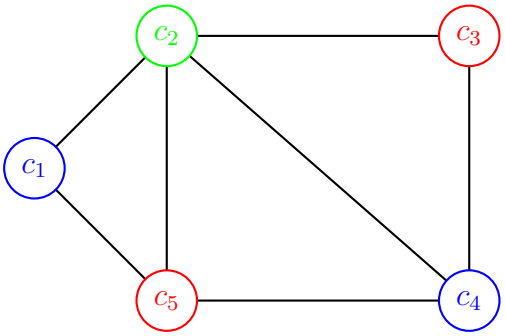
Um grafo $G = (V, E)$ é composto por um conjunto $V$ de vértices, ou nós, e por um conjunto $E$ de pares não-ordenados de vértices, denominados arestas. Os nós $u$ e $v$ são vizinhos, ou adjacentes, se existir a aresta $(u, v)$. Um grafo é completo se seus vértices são vizinhos dois a dois. Um caminho de comprimento $\ell$ (arestas) em um grafo é uma sequência $(v_1, v_2, \ldots, v_{l+1})$ de nós distintos, de forma que $v_i$ é adjacente a $v_{i+1}$, para $1 \leq i \leq \ell$. Dizemos que dois vértices distintos são vizinhos de distância-$k$ se o menor caminho que os interliga tem comprimento $k$.

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 $v$. 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 $v$. Dado $S \subset V$, denotamos a média aritmética, o mínimo e o máximo dentre os graus de $S$ por $\overline{\delta}(S)$, $\delta(S)$ e $\Delta(S)$. Além disso, para simplificar a notação, convencionamos que $\overline{\delta} \equiv \overline{\delta}(V)$, $\delta \equiv \delta(V)$ e $\Delta \equiv \Delta(V)$.

# 4 O método do grafo coluna-interseção

## 4.1 O grafo coluna-interseção

Na primeira modelagem utilizaremos o grafo coluna-interseção ($GCI$). Nesse grafo, cada nó representa uma coluna da matriz. Se duas colunas têm elementos não nulos em uma mesma linha, isto é, são estruturalmente não-ortogonais, os vértices respectivos são adjacentes. Para ilustrar, tomemos novamente a jacobiana $\widehat{J}$ da Figura 1. O $GCI$ associado a $\widehat{J}$ é ilustrado na Figura 3. O nó $c_5$, por exemplo, é ligado a $c_1$ porque $j_{11} \neq 0$ e $j_{15} \neq 0$. Ou seja, o nó $c_i$ é ligado ao nó $c_j$ se $j_{ki} \neq 0 \neq j_{kj}$ para algum $k$.




Coloração

| nó | cor |
|---|---|
| $c_1$ | azul |
| $c_2$ | verde |
| $c_3$ | vermelho |
| $c_4$ | azul |
| $c_5$ | vermelho |

Figura 3: O grafo coluna-interseção $\widehat{G}$ associado à matriz $\widehat{J}$ da Figura 1

## 4.2 A coloração distância-1

Colorir um grafo consiste em atribuir alguma cor a cada um de seus nós. Dizemos que uma coloração é distância-1 se nós que são vizinhos têm cores diferentes. Então, o problema de achar o mínimo número de grupos estruturalmente ortogonais de uma matriz pode ser formulado como achar uma coloração distância-1 do $GCI$ associado a esta matriz que utilize o menor número possível de cores. Repare que a coloração do grafo $\widehat{G}$ exibida na Figura 3 é de distância-1. Coleman e Moré [2] foram os primeiros a formular o problema de particionamento das colunas como uma coloração distância-1 do $GCI$ correspondente.

O menor número de cores necessário para uma coloração distância-1 é chamado de número cromático do grafo, denotado por $\chi(G)$. Um subgrafo completo de $G$ é chamado de clique. A cardinalidade de uma clique é igual ao seu número de vértices. O número clique de um grafo $G$ é a cardinalidade da maior clique contida no grafo, denotada por $\omega(G)$.

Claramente, são necessárias $k$ cores para colorir uma clique $k$ nós, o que implica a desigualdade

$$\omega(G) \leq \chi(G).$$

Além disso, temos o resultado de Brooks [1]:

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

sendo que $\chi(G) = \Delta + 1$ se e somente se ou $\Delta \neq 2$ e $G$ contém uma clique de cardinalidade $\Delta + 1$, ou $\Delta = 2$ e $G$ possui um ciclo ímpar.

O limite inferior para $\chi(G)$ permite concluir, por exemplo, que a coloração exibida na Figura 3 é ótima, pois $\widehat{G}$ contém cliques de cardinalidade três, como, por exemplo, o subgrafo induzido pelos nós $c_1$, $c_2$ e $c_5$.

Alguns cliques de um *GCI* são facilmente detectados a partir da matriz correspondente. Quando identificamos, por exemplo, três não nulos na primeira linha isso indica que as colunas correspondentes são estruturalmente não ortogonais entre si, formando então um clique de cardinalidade três no grafo *GCI*. Seja $\rho_{max}$ o maior número de não nulos numa mesma linha. Então,

$$\rho_{max} \leq \omega(G) \leq \chi(G),$$

e $\rho_{max}$ é um limite inferior fácil de obter.

Como o problema de coloração distância-1 é NP-difícil, é improvável a existência de um algoritmo polinomial para o particionamento das colunas de uma matriz em um número mínimo de subconjuntos estruturalmente ortogonais. Isto justifica a pesquisa de heurísticas para o problema.

A coloração distância-1, apresentada na Figura 3, dos nós do *GCI* $\widehat{G}$, correspondente à matriz $\widehat{J}$ da Figura 1, pode ser apresentada diretamente na matriz, como na Figura 4 (a). Se somarmos as colunas de mesma cor, encontramos a versão compacta de $\widehat{J}$ já obtida anteriormente, apresentada agora em cores na Figura 4 (b).

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

(a) Coloração distância-1 de $\widehat{J}$

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

(b) Versão compacta de $\widehat{J}$ induzida pela coloração em (a)

Figura 4: Coloração distância-1 do grafo $\widehat{G}$ aplicada a $\widehat{J}$

# 5 O método do grafo bipartido

## 5.1 O modelo grafo bipartido

Em um grafo bipartido, o conjunto $V$ de nós pode ser particionado em dois subconjuntos de tal maneira que nós pertencentes a um mesmo subconjunto não sejam adjacentes. Devido a esse fato, usa-se a notação $G = (V_1, V_2, E)$ para o grafo bipartido, onde $V_1$ e $V_2$ são os dois subconjuntos de nós mencionados. Um outro modelo para o problema de particionamento de colunas é o modelo grafo bipartido, onde $V_1$ está associado às colunas da matriz jacobiana e $V_2$ às suas linhas. Uma aresta liga o nó (coluna) $c_i \in V_1$ ao nó (linha) $\ell_k \in V_2$ se o elemento $j_{ki}$ é não nulo. Na Figura 5 temos o grafo bipartido associado à jacobiana $\widehat{J}$ da Figura 1. Observe na Figura 5 que $\Delta(V_1) = 2$ e $\Delta(V_2) = 3$.

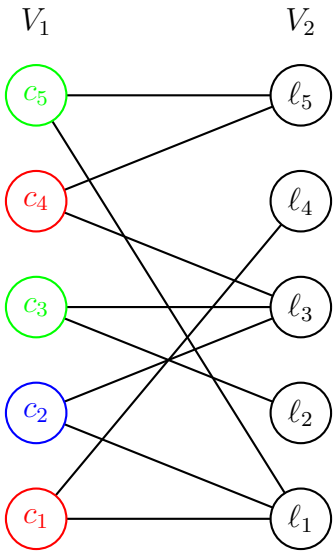



Figura 5: Grafo bipartido $\widehat{G_b}$ associado à matriz $\widehat{J}$ da Figura 1

É interessante notar que a estrutura de esparsidade da matriz jacobiana é completamente

herdada pelo grafo bipartido associado, ou seja, existe um elemento não nulo na linha $k$ e coluna $i$ se, e somente se, o nó $c_i$ é adjacente ao nó $\ell_k$. Isto proporciona uma relação um-a-um entre uma matriz e o grafo bipartido correspondente, enquanto que várias matrizes de estruturas distintas no que se refere ao padrão de elementos não nulos podem levar ao mesmo grafo coluna-interseção.

## 5.2 A coloração distância-2

Caminhos entre nós pertencentes a um mesmo subconjunto de um grafo bipartido, digamos $V_1$, alternam necessariamente entre nós de $V_1$ e $V_2$, e possuem, portanto, um número par de arestas. Relembrando, os nós $c_i$ e $c_j$ são adjacentes no *GCI* correspondente a uma matriz $J$ se existe $k$ tal que $j_{ki} \neq 0 \neq j_{kj}$. Este fato, por outro lado, implica que os nós $c_i$ e $c_j$ do grafo bipartido associado à mesma matriz são adjacentes ao nó $\ell_k$, e estão portanto à distância-2. Ou seja, os nós que estão à distância-1 no modelo *GCI* de uma matriz são precisamente os nós em $V_1$ do seu modelo bipartido, que estão à distância-2. Então o problema de achar o mínimo número de grupos estruturalmente ortogonais de uma matriz pode ser formulado como o problema de coloração parcial (somente dos nós em $V_1$) distância-2 do grafo bipartido associado à matriz.

Se fizermos uma coloração dos nós em $V_1$ de tal forma que dois nós à distância-2 são de cores distintas, dizemos que esta é uma coloração distância-2 parcial. A Figura 5 mostra uma coloração distância-2 parcial da matriz jacobiana $\widehat{J}$. Repare que a coloração dos nós colunas da Figura 5 coincide com a coloração do *GCI* exibido na Figura 3.

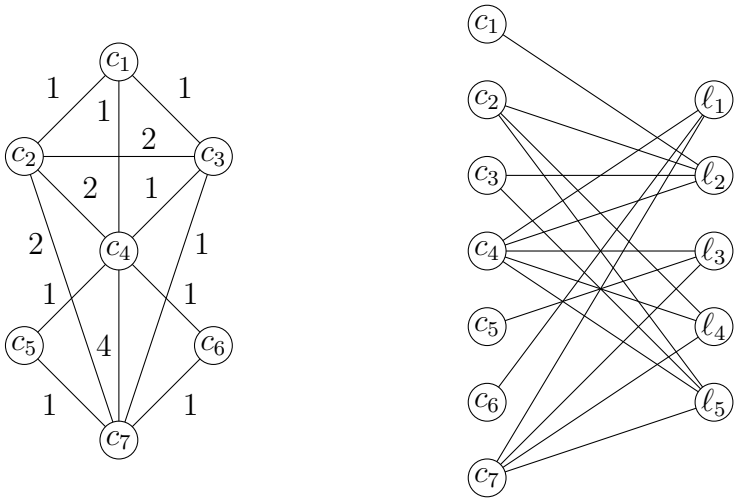
Gebremedhin, Manne e Potheny [5] elaboraram esse modelo de coloração distância-2 parcial do *GB*, e o defendem como uma representação mais flexível e robusta do problema de particionamento das colunas da matriz jacobiana.

Observe que $\Delta(V_2)$ é justamente o máximo número de não nulos por linha ($\rho_{max}$), consequentemente $\Delta(V_2)$ é um limite inferior para o número de cores a serem usadas em uma coloração distância-2.

# 6 Comparações teóricas entre os modelos

O *GB* permite identificar a estrutura de esparsidade da matriz, o que não é possível no *GCI*. Isto equipa este modelo com uma flexibilidade maior. Podemos, por exemplo, usar o *GB* para fazer uma partição das linhas, colunas ou uma combinação dessas em grupos estruturalmente ortogonais fazendo uma coloração distância-2 dos nós-linhas, nós-colunas ou uma combinação dos dois. Já o *GCI* serve apenas para que seja feita uma coloração

distância-1 das colunas.




$$\begin{bmatrix} 0 & 0 & 0 & a_{14} & 0 & a_{16} & a_{17} \\ a_{21} & a_{22} & a_{23} & a_{24} & 0 & 0 & 0 \\ 0 & 0 & 0 & a_{34} & a_{35} & 0 & a_{37} \\ 0 & a_{42} & 0 & a_{44} & 0 & 0 & a_{47} \\ 0 & a_{52} & a_{53} & a_{54} & 0 & 0 & a_{57} \end{bmatrix}$$

Figura 6: Uma matriz com o seu grafo coluna-interseção com pesos à esquerda e seu grafo bipartido à direita

Em relação ao número de nós dos grafos, não há mistério: o *GCI* tem tantos nós quanto colunas, ou seja, $n$ nós. O *GB* tem tantos nós quanto a soma dos colunas e linhas, ou seja, $n + m$. O número de arestas do *GB* é simplesmente o número de não nulos da matriz, enquanto o número de arestas do *GCI* depende da estrutura de esparsidade da matriz. Então, mesmo com o tamanho da matriz e o número de elementos não nulos, não podemos determinar *a priori* o número de arestas do *GCI*. Entretanto, podemos estabelecer uma relação entre o número de arestas de um grafo e o de outro.

Denotemos por $G = (V_1, E_c)$ e $G_b = (V_1, V_2, E_b)$ o *GCI* e o *GB* associados a uma matriz. Para cada par de colunas $\{c_i, c_j\}$ atribuímos um peso $w(c_i, c_j)$ igual à quantidade de linhas em que ambas as colunas têm elementos não nulos. As arestas do grafo coluna-interseção $G$ têm, portanto, pesos positivos, iguais ou superiores a 1. Denotaremos por $\overline{w}$ a média aritmética dos pesos das arestas apenas. Em particular, decorre que $\overline{w} \geq 1$. Segue da definição do grafo bipartido, que $w(c_i, c_j)$ é também o número de caminhos distância-2 entre $c_i$ e $c_j$ em $G_b$. Por outro lado, se $c_i$ tem elemento não nulo na linha $\ell_k$, $d(\ell_k) - 1$ é o número de caminhos distância-2, que passam por $\ell_k$, entre $c_i$ e outros nós-coluna no grafo bipartido.

Assim, temos que

$$\sum_{c_j \in V_1} w(c_i, c_j) = \sum_{\ell_k \in N(c_i)} (d(\ell_k) - 1).$$

Segue então que

$$\sum_{\{c_i, c_j\} \in E_c} w(c_i, c_j) = \sum_{c_i \in V_1} \sum_{\ell_k \in N(c_i)} (d(\ell_k) - 1).$$

Como $\Delta(V_2)$ é um limite superior para $d(\ell_k)$, para todo $\ell_k$, temos que

$$\sum_{c_i \in V_1} \sum_{\ell_k \in N(c_i)} (d(\ell_k) - 1) \leq (\Delta(V_2) - 1) \sum_{c_i \in V_1} \sum_{\ell_k \in N(c_i)} 1 = (\Delta(V_2) - 1) \sum_{c_i \in V_1} d(c_i) = (\Delta(V_2) - 1)|E_b|.$$

Portanto,

$$|E_c| 2\overline{w} = \sum_{\{c_i, c_j\} \in E_c} w(c_i, c_j) \leq (\Delta(V_2) - 1)|E_b|,$$

ou seja,

$$|E_c| \leq \frac{\Delta(V_2) - 1}{2\overline{w}} |E_b|.$$

Temos então um limite superior para o número de arestas do *GCI*. Com base em testes empíricos, os autores de [4] afirmam que $|E_c|$ chega próximo ao limite superior $\frac{\Delta(V_2)-1}{2\overline{w}}|E_b|$, e que $\frac{\Delta(V_2)-1}{2\overline{w}} > 1$ na maioria dos grafos coluna-interseção provendo de matrizes esparsas, e, consequentemente $|E_c| > |E_b|$.

E se, ao invés de obter um limite superior, admitimos a aproximação:

$$\sum_{v \in V_1} \sum_{z \in N(v)} (d(z) - 1) \approx \sum_{v \in V_1} \sum_{z \in N(v)} (\overline{\delta}(V_2) - 1) = (\overline{\delta}(V_2) - 1)|E_b|,$$

Temos que $|E_c|$ deve ser aproximadamente $|E_b|\overline{\delta}(V_2)/2\overline{w}$.

Na execução da heurística associada ao *GCI*,[2] dezesseis dos vinte grafos coluna-interseção apresentaram $\overline{\delta}(V_2)/2\overline{w} > 1$ indicando que esses *GCI* provavelmente possuem mais arestas que o correspondente *GB*.

Antes de começar a construção do *GCI* e do *GB*, já sabemos quantos nós terá cada um e o tempo de meramente alocar esses nós é trivial em comparação ao tempo de calcular quais arestas terão, e alocá-las. Então tendo mais arestas que não nulos (onde número de não nulos é igual $|E_b|$), prejudica muito o tempo de construção do *GCI*.

# 7 Implementação

Os algoritmos apresentam uma estrutura básica que envolve: leitura dos dados, montagem do grafo, coloração do grafo e liberação do grafo, conforme fluxograma na Figura 7.

[2]Calculamos $\overline{w}$ durante a construção do *GCI*, enquanto $\delta(V_2)$ é uma propriedade da matriz, a média de não nulos por linha.

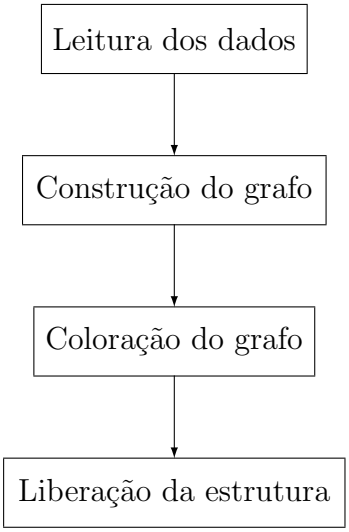



Figura 7: Arcabouço dos algoritmos

## 7.1 Construção do *GCI*

Na construção, o algoritmo obtém um conjunto de vizinhos por vez, ou seja, colunas que não são estruturalmente ortogonais entre si, e constrói o grafo de forma gradual a partir desses conjuntos. A construção possui duas fases distintas:

1. Encontrar um conjunto de vizinhos.
2. Inserir esse conjunto de vizinhos na estrutura de dados, isto é, inserir no grafo.

Para encontrar um conjunto de vizinhos, lemos uma linha da matriz. Ao encontrar $k$ elementos não nulos nessa mesma linha, sabemos que as $k$ colunas desses elementos são estruturalmente não ortogonais entre si. Armazenamos esse conjunto de vizinhos num vetor e depois inserimos na estrutura de dados. Esse procedimento é repetido para cada linha. Apresentamos o pseudo-código dessa rotina no Algoritmo 1.

## 7.2 Montagem do *GB*

A montagem do grafo bipartido é muito simples. Inicializamos um grafo com todos os nós colunas e nós linhas. Para cada não nulo $a_{ik}$ na matriz, sabemos que o nó $c_k$ e o nó $\ell_i$ são adjacentes e os ligamos por uma aresta. O pseudo-código encontra-se a seguir no Algoritmo 2.

**Seja** $m$ o nº de linhas e $n$ o nº de colunas de uma matriz $A$.
**Inicialize** *Vizinhos*, o vetor de vizinhos.
**Inicialize** A estrutura de dados com $n$ colunas.
**Para** $i$ de 1 até $m$
    **Para** $j$ de 1 até $n$
        se $a_{ij} \neq 0$, $Vizinhos \leftarrow Vizinhos \cup c_j$
    **Insira** *Vizinhos* na estrutura. Esvazia *Vizinhos*.

Algoritmo 1: Montagem do grafo coluna-interseção

**Seja** $m$ o nº de linhas e $n$ o nº de colunas de uma matriz $A$.
**Inicialize** a estrutura de dados com $m$ linhas e $n$ colunas.
**Para** $i$ de 1 até $m$
    **Para** $j$ de 1 até $n$
        se $a_{ij} \neq 0$, ligamos $c_j$ e $\ell_i$ por uma aresta

Algoritmo 2: Montagem do grafo bipartido

## 7.3 Coloração distância-1

As cores serão representadas por números naturais, portanto afirmações como "encontre a menor cor disponível" são pertinentes. Na solução gulosa para o problema de fazer uma coloração distância-1, nós são coloridos sequencialmente, sendo percorridos em alguma ordem pré-determinada.

**Seja** $c_1, c_2, \ldots, c_n$ uma ordenação dos nós pertencentes a $V_1$.
**Incializa** um vetor *forbiddenColors* de tamanho $|V_1|$ com $a \notin V_1$
**Para** $i$ de 1 até $n$
    **Para** cada $w$ colorido $\in N(c_i)$
        $forbiddenColors[cor(w)] \leftarrow c_i$
    $cor[c_i] \leftarrow \min\{c > 0 : forbiddenColors[c] \neq c_i\}$

Algoritmo 3: Coloração distância-1 do grafo coluna-interseção

Suponha que as colunas de índice inferior a $i$ já foram coloridas. Mantemos um vetor *cor* indexado de acordo com os nós, onde cada índice guarda a cor do nó. Para colorir $c_i$, criamos uma lista de cores proibidas, denominada *forbiddenColors*, da seguinte forma: percorremos o

conjunto $N(c_i)$ e, para cada $w \in N(c_i)$ já colorido com a cor $cor(w)$, marcamos $c_i$ na posição $cor(w)$ do *forbiddenColors*. Após percorrer todos os vizinhos, colorimos $v$ com a menor cor disponível, ou seja, a menor posição que não esteja marcada com $c_i$ no *forbiddenColors*. No Algoritmo 3 exibimos o pseudo-código.

## 7.4 Coloração distância-2

Novamente, na solução gulosa para coloração distância-2, os nós de $V_1$ são coloridos sequencialmente. Mantemos um vetor *cor* indexado de acordo com os nós, onde cada índice guarda a cor do nó. Se $c_i$ é o próximo a ser colorido, criamos inicialmente uma lista de cores proibidas para $c_i$ denominada *forbiddenColors*. Percorremos então o conjunto $N_2(c_i)$ e, para cada $w \in N_2(c_i)$ já colorido com a cor $cor(w)$, marcamos $c_i$ na posição $cor(w)$ do *forbiddenColors*. Após percorrer todos os vizinhos, colorimos $c_i$ com a menor cor disponível, ou seja, a menor posição que não esteja marcada com $c_i$ no *forbiddenColors*. O pseudo-código segue abaixo no Algoritmo 4.

**Seja** $c_1, c_2, \ldots, c_n$ uma ordenação dos nós pertencentes a $V_1$.
**Incializa** um vetor *forbiddenColors* de tamanho $|V_1|$ com $a \notin V_1$.
**Para** $i$ de 1 até $n$
    **Para** cada $w$ colorido $\in N_2(c_i)$
        $forbiddenColors[cor(w)] \leftarrow v_i$
    $cor[c_i] \leftarrow \min\{c > 0 : forbiddenColors[c] \neq c_i\}$

Algoritmo 4: Coloração distância-2 do grafo bipartido

Dois nós-colunas que estão à distância-1 no grafo coluna-interseção estão à distância-2 no grafo bipartido e vice-versa. Se executamos uma coloração distância-2 dos nós-colunas na mesma ordem que fazemos uma coloração distância-1 dos nós-colunas no grafo coluna-interseção, obtemos a mesma coloração dos nós-colunas.

A montagem do grafo grafo bipartido é quase trivial por haver uma relação biunívoca entre cada nó do grafo bipartido e um elemento não nulo da matriz, enquanto a montagem do grafo coluna-interseção é um processo mais elaborado. Em compensação, a coloração distância-1 é uma rotina mais simples que a coloração distância-2 cuja dificuldade está embutida na determinação de $N_2(c_i)$.

# 8 As matrizes de teste

Optamos por usar o mesmo banco de matrizes esparsas que os autores de [4], o *University of Florida Sparse Matrix Collection* [6], ao invés de matrizes geradas aleatoriamente. De acordo com Tim Davis [6], o pesquisador da universidade da Flórida que elaborou o sítio e a coleção, matrizes aleatórias são nitidamente diferentes das matrizes que aparecem em problemas reais. Matrizes aleatoriamente geradas têm um preenchimento catastrófico, enquanto matrizes reais sempre exibem alguma estrutura e os preenchimentos podem ser reduzidos aproveitando essa estrutura. Além disso, matrizes aleatórias possuem uma alta probabilidade de serem não singulares e tendem a ser bem condicionadas, diferentemente de matrizes que provêm de problemas reais.

## 8.1 Formato dos dados

No banco de dados do [6], cada matriz está disponível em três formatos: MATLAB mat-file, Rutherford-Boeing e Matrix Market. Nesses formatos, somente os elementos não nulos da matriz são armazenados. Dado que nossos algoritmos serão implementados na linguagem C++, o formato para MATLAB não é conveniente. O mais compacto é o Rutherford-Boeing, porém o formato Matrix Market é de fácil manipulação e compacto o suficiente para nossos propósitos, razão pela qual optamos por este último. Neste caso, cada não nulo da matriz é representada por uma tripla $(i, j, v)$ que indica a linha $(i)$, a coluna $(j)$ e o valor $(v)$ do elemento não nulo da matriz. Todas as triplas estão amanezadas num arquivo no formato texto. O sítio do Matrix Market [8] fornece um programa escrito em linguagem C que lê esse arquivo e devolve três vetores: $I$, $J$ e $Val$. A $k$-ésima posição desses vetores representa um não nulo, cuja linha, coluna e valor são $I_k$, $J_k$ e $Val_k$. As triplas estão em ordem crescente de colunas, ou seja, o vetor $J$ está em ordem crescente. A Figura 8 traz um exemplo de uma matriz neste formato após execução do programa de leitura.

$$\begin{pmatrix} 0.75 & 0.30 & 0 \\ 0.50 & 0 & 0 \\ 0 & 0.25 & 0 \end{pmatrix} \qquad \begin{array}{rcccc} I: & 2 & 1 & 3 & 1 \\ J: & 1 & 1 & 2 & 2 \\ Val: & 0.50 & 0.75 & 0.25 & 0.30 \end{array}$$

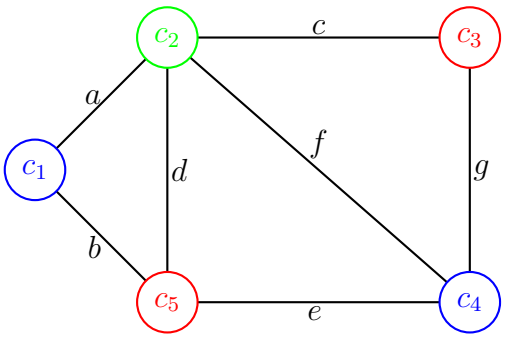
Figura 8: Matriz (esquerda) e os três vetores que a representam (direita) no formato Matrix Market

# 9 A estrutura de dados

Para escolhermos uma estrutura de dados apropriada à representação de um grafo, consideramos o tempo de execução de algumas operações básicas e a memória disponível.

## 9.1 Considerações sobre espaço de memória

Uma representação típica de grafo é através de uma matriz de adjacência ou incidência. Na matriz de incidência *nó* $\times$ *arco* as linhas são associadas aos nós e as colunas às arestas. O elemento na linha $i$ e coluna $j$ é 1 se o nó $i$ é incidente no arco $j$, e 0 caso contrário. A matriz de adjacência tem linhas e colunas associadas aos nós. O elemento na linha $i$ e coluna $j$ é o número de arestas que têm $i$ e $j$ como extremidades. Na Figura 9 temos as matrizes de adjacência e de incidência do *GCI* da matriz $\widehat{J}$ da Figura 1. As matrizes de teste são esparsas, consequentemente os grafos também o serão, ou seja, cada nó terá poucos vizinhos relativamente ao número de não nulos, portanto, cada $N(v)$ terá poucos elementos. Isto implica que as matrizes de adjacência e incidência serão também esparsas. A estrutura escolhida deve explorar esta esparsidade, armazenando informações apenas sobre nós e arestas de fato existentes.




$$\begin{array}{c} \\ c_1 \\ c_2 \\ c_3 \\ c_4 \\ c_5 \end{array} \begin{array}{c} \begin{array}{ccccc} c_1 & c_2 & c_3 & c_4 & c_5 \end{array} \\ \left[\begin{array}{ccccc} 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 \end{array}\right] \end{array} \qquad \begin{array}{c} \\ c_1 \\ c_2 \\ c_3 \\ c_4 \\ c_5 \end{array} \begin{array}{c} \begin{array}{ccccccc} a & b & c & d & e & f & g \end{array} \\ \left[\begin{array}{ccccccc} 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 0 & 0 \end{array}\right] \end{array}$$

Figura 9: A matriz de adjacência e incidência do *GCI* da $\widehat{J}$

## 9.2 Considerações sobre tempo de acesso

Durante a montagem e coloração dos grafos utilizaremos diversas ordens de percorrer os nós, além da ordem natural. Por isso é vantajoso que os nós sejam representados por um vetor. Um vetor tem uma vantagem fundamental no tempo de acesso de um elemento. Cada elemento do vetor é alocado sequencialmente na memória virtual.[3] Com isso, posições do vetor podem ser acessadas diretamente, com tempo assintótico constante — $O(1)$.

Para a implementação da subrotina de coloração distância-1, é preciso checar as cores dos nós vizinhos. Logo, ter acesso rápido ao conjunto dos vizinhos é conveniente. Mas qual a melhor estrutura para esses conjuntos de vizinhos? Listas ligadas ou vetores? Diferentemente do que ocorre com um vetor, para encontrar o $k$-ésimo nó de uma lista ligada é necessário percorrer nó por nó, começando no nó inicial, até chegarmos à $k$-ésima posição. São necessárias $k$ avaliações para se acessar o $k$-ésimo elemento, o que implica em um tempo assintótico de $O(n)$. Adotamos uma lista para representar os vizinhos de um nó por dois motivos:

- Durante a construção dos grafos, o tamanho de cada lista de vizinhos precisa ser redimensionado, pois os algoritmos adicionam os nós vizinhos ao encontrá-los. Realocar um vetor consome mais tempo e pode gerar dificuldades.
- Ao acessar o conjunto dos vizinhos de um determinado nó $v$, *a priori* não sabemos quantos nem quais nós estão no conjunto $N(v)$. Descobrimo-los atravessando o conjunto numa ordem arbitrária. Isso elimina a vantagem de um vetor ao invés de uma lista, porque com um vetor posso acessar qualquer posição em tempo $O(1)$ mas, dado que atravessaria o vetor na ordem natural[4], tanto na lista como num vetor tem-se tempo $O(n)$. Logo uma lista é suficiente.

Por isso optamos pela estrutura de um vetor de listas ligadas esquematizada na Figura 10, correspondente ao grafo da Figura 9. A sequência de caixas é o vetor de nós. No caso do *GCI* esses nós serão os nós colunas, enquanto no *GB* os primeiros $n$ nós serão os nós colunas, e, os seguintes $m$ nós, os nós linhas. Cada posição $v$ do vetor tem uma lista conectada, representada pelos círculos dispostos verticalmente, que contém o conjunto de vizinhos de $v$. Então, por exemplo, o nó 1 do grafo da Figura 9 é representado pela primeira caixa de etiqueta '1' na Figura 10. A lista conectada verticalmente à caixa 1 é uma lista dos nós adjacentes ao nó 1 do grafo, os nós 2 e 5.

[3]Para um vetor de $n$ inteiros, dado que cada inteiro tem, em geral, um tamanho de 4 bytes, será alocado um bloco de $4n$ bytes. Para acessar o $k$-ésimo elemento de um vetor $W$, a posição inicial do vetor é localizada na memória virtual e ao avançar $4k$ bytes, chegamos no $k$-ésimo elemento.

[4]Onde natural é a ordem em que os dados foram armazenados.

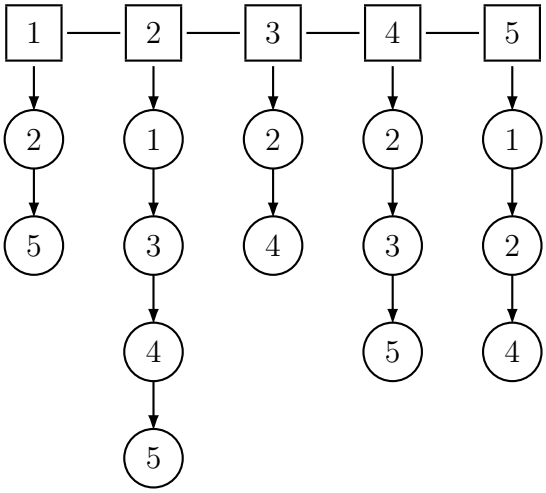


Figura 10: A estrutura de dados

## 9.3 Liberação do dados

O Algoritmo 5 contém a rotina de liberação utilizada.

**Seja** $V$ o vetor de nós e $|V|$ o tamanho do vetor
  **Para** $j$ de 1 até $|V|$
    **Se** existir uma lista anexada a $V_j$
      **Seja** $n_1$ o primeiro nó da lista
      **Enquanto**existir $v$ vizinho de $n_1$.
        **Conecta** o $n_1$ ao vizinho de $v$*.
        **Libera** o $n_1 \rightarrow v$.
      **Libera** $n_1$
    **Libera** $V_j$
  **Libera** $V$

*Se $v$ não possui vizinho, $n_1$ é conectado ao NULL

Algoritmo 5: A liberação da estrutura de dados

Antes da liberação de um nó $n_1$, temos um procedimento para garantir que a lista ligada continua conectada. Conectamos o nó $n_1$ ao vizinho de seu vizinho. Se não tomamos essa precaução, ao liberar um nó da lista perdemos o caminho até os próximos nós da lista.

# 10 O algoritmo do grafo coluna-intercessão

Agora que determinamos a estrutura de dados, podemos entrar em mais detalhes de implementação. Primeiramente consideramos o algoritmo que particiona as colunas em grupos estruturalmente ortogonais por meio de uma coloração distância-1 do grafo coluna-interseção, o *AGCI* (Algoritmo Grafo Coluna-interseção).

## 10.1 Montando o grafo

Nesse etapa, ocorrem duas ações distintas: (i) encontramos um conjunto de vizinhos mútuos, e (ii) inserimo-los na estrutura de dados.

### 10.1.1 Encontrando os vizinhos

A matriz é representada por três vetores $I$, $J$ e *Val*, conforme explicado na seção 8, onde *Val* pode ser descartado. Somente a posição nos interessa e não o valor de cada não nulo. Antes de começar a procura pelos vizinhos, ordenamos os pares[5] $(I_k,J_k)$ de acordo com a linha. Aqui está um exemplo típico após essa ordenação:

$$\begin{array}{cc} I & J \\ (1 & 3) \\ (1 & 7) \\ (1 & 6) \\ (2 & 1) \\ (2 & 2) \\ (3 & 9) \\ \vdots & \vdots \end{array}$$

Percebemos neste exemplo que existem elementos não nulos $j_{13}, j_{17}, j_{16}$ na linha um e, consequentemente, as colunas 3, 7 e 6 não são colunas estruturalmente ortogonais. Identificamos assim um conjunto $\{3, 7, 6\}$ de nós que são vizinhos dois a dois. Logo, para cada linha, encontramos um conjunto de vizinhos, armazenando todos os valores das colunas dos não nulos que estão nessa linha. A seguir explicitamos o algoritmo que monta o *GCI* no Algoritmo 6.

[5]Para ordenação usamos sempre o algoritmo *quicksort*.

**Seja** $nnz$ e $n$ o nº de não nulos e de colunas da matriz.
**Seja** $I$ e $J$ os vetores das linhas e colunas dos não nulos.
**Inicialize** *Vizinhos*, o vetor de vizinhos.
**Inicialize** o grafo $G$.
$Vizinhos = Vizinhos \cup J_1$
**Para** $k$ de 2 até $nnz$
    **Se** $I_k = I_{k-1}$
        $Vizinhos \leftarrow Vizinhos \cup J_k$
    **Senão** insira *Vizinhos* em $G$
        Esvazia *Vizinhos*
        $Vizinhos \leftarrow Vizinhos \cup J_k$
**Retorne** $G$

Algoritmo 6: A montagem do *GCI*

### 10.1.2 Inserindo os vizinhos na estrutura de dados, G

Após achar um vetor de *Vizinhos*, acessamos o primeiro elemento do vetor, $Vizinhos_1$. Seja $j$ o índice da coluna contido em $Vizinhos_1$. Então, todos os outros elementos desse vetor são vizinhos de $c_j$, e devem ser inseridos na lista dos vizinhos de $c_j$, $N(c_j)$. Se $N(c_j)$ já contém alguns elementos, é preciso cuidado para não inserir múltiplas cópias do mesmo nó na lista de vizinhos. Para evitar repetições, executamos uma fusão do vetor *Vizinhos* e com o vetor $N(c_j)$, ou seja, intercalamos os dois, de forma que o resultado da fusão é o novo $N(c_j)$. Nessa fusão, quando elementos iguais são encontrados, um deles é eliminado. Para esse fim, mantemos a lista $N(c_j)$ e o vetor de vizinhos ordenados. Com isso, facilmente evitamos repetições em $N(Cj)$ e reduzimos o tempo de inserção. Esse processo é repetido para todos os elementos do vetor. O pseudo-código encontra-se no Algoritmo7.

**Seja** $k$ o tamanho do vetor *Vizinhos*.
**Para** $i$ de 1 até $k$
    $N(Vizinhos_i) \leftarrow$ fusão de $N(Vizinhos_i)$ e *Vizinhos*

Algoritmo 7: A inserção dos vizinhos

## 10.2 Coloração distância-1

A coloração distância-1 consiste em encontrar uma cor para cada nó, diferente das cores dos seus vizinhos. A estrutura de dados de vetor de listas de vizinhos é conveniente pois é fácil acessar os vizinhos de um nó. O pseudo código já foi apresentado na Seção 7.3.

# 11 O algoritmo do grafo bipartido

Usaremos a sigla $AGB$ para designar o algoritmo associado ao grafo bipartido.

## 11.1 Montando o grafo

Após a leitura da matriz, são alocados os vetores $I$ e $J$. Cada $(I_i, J_i)$ representa as coordenadas de um elemento não nulo, logo sabemos que a linha $I_i$ e coluna $J_i$ serão ligadas por uma aresta no grafo bipartido. Então inserimos em ordem a linha $I_i$ na lista de vizinhos da coluna $J_i$ e inserimos em ordem a coluna $J_i$ na lista de vizinhos da linha $I_i$. A seguir no Algoritmo 8, o pseudo-código dessa construção.

**Seja** $m$ o nº de linhas, $n$ o de nº colunas, e $nnz$ o número de
não nulos da matriz $A$.
**Inicialize** a estrutura de dados com $m$ nós-linhas e $n$ nós-colunas.
**Para** $i$ de 1 até $nnz$
$N(\ell_{I_i}) = N(\ell_{I_i}) \cup c_{J_i}$
$N(c_{J_i}) = N(c_{J_i}) \cup \ell_{I_i}$

Algoritmo 8: A montagem do $GB$

## 11.2 Coloração distância-2

Atravessamos os nós-colunas do conjunto $V_1$ do grafo bipartido na ordem natural e para cada nó $v_i \in V_1$, percorremos os nós $w \in N_2(v_i)$ e, para cada $w$ colorido, fazemos *forbiddenColors*[cor(w)]= $v_i$. Dada a nossa estrutura de dados, para percorrer os nós $w \in N_2(v)$ precisamos percorrer o $N_1(y)$ de cada $y \in N_1(v_i)$. Então, para todo $w \in N_1(y)$ colorido, fazemos *forbiddenColors*[cor(w)]= $v_i$. Após repetir esse processo para todos os vizinhos distância-2, colorimos $v_i$ com a menor posição que não esteja marcada com $v_i$ no *forbiddenColors*. Como podemos ver no Algoritmo 9, essa rotina possui *loop* a mais em relação ao rotina de coloração distância-1 (Algoritmo 3).

**Seja** $\{v_1, v_2, \cdots, v_n\}$ o vetor de nós colunas, ou seja $v_i \in V_1$
**Inicializa** uma lista *forbiddenColors*, onde cada $forbiddenColors_i = a$, $a \notin V$
**Para** $i$ de 1 até $n$
    **Para** cada $y \in N_1(v_i)$
        **Para** cada $w$ colorido $\in N_1(y)$
            $forbiddenColors[cor(w)] \leftarrow v_i$
    $cor[v_i] \leftarrow \min\{c > 0 : forbiddenColors[c] \neq v_i\}$

Algoritmo 9: O algoritmo de coloração distância-2

# 12 Resultados computacionais

Procuramos resumir nesta seção os resultados de nossos testes computacionais. Para tanto apresentamos tabelas que resumem as características das 23 matrizes utilizadas nos testes, dos grafos correspondentes e gráficos que indicam o desempenho dos algoritmos. Comparamos a performance dos dois algoritmos e também nossos resultados em relação aos resultados reportados no artigo [4].

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, enquanto que em [4] as implementações foram feitas em linguagem C e os programas executados em um Intel Pentium 4, 2.53 GHz com 1 GB memória RAM no sistema Linux 2.4.20/RedHat 8.0.

As matrizes provieram do mesmo banco de matrizes esparsas que os autores de [4], o *University of Florida Sparse Matrix Collection* [6], com exceção das matrizes com sufixo "_MM", que foram tiradas do sítio do Matrix Market [8], 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.

A Tabela 1 contém dados relativos às matrizes. As colunas $m$, $n$ e $nnz$ contêm o número de linhas, colunas e elementos não nulos. O máximo, mínimo e média de não nulos por coluna são denotados por $\kappa_{max}$, $\kappa_{min}$ e $\overline{\kappa}$, respectivamente. Os mesmos números para as linhas são designados por $\rho_{\max}$, $\rho_{\min}$ e $\overline{\rho}$, respectivamente.

Nas Tabelas 2 e 3, na parte separada à esquerda, estão as características dos grafos. $|V|$ e $|E|$ são os números de vértices e arestas. Denotaremos a média aritmética, o mínimo e o máximo graus dos nós de $\overline{\delta}$, $\delta$ e $\Delta$. Os símbolos no cabeçalho da Tabela 3 são definidos como segue: $\Delta = \max\{\Delta(V_1), \Delta(V_2)\}$, $\delta = \min\{\delta(V_1), \delta(V_2)\}$ e $\overline{\delta} = (\overline{\delta}(V_1) + \overline{\delta}(V_2))/2$.

Tabela 1: Dados das Matrizes de teste

| Matriz | $m$ | $n$ | $nnz$ | $\kappa_{max}$ | $\kappa_{min}$ | $\overline{\kappa}$ | $\rho_{\max}$ | $\rho_{\min}$ | $\overline{\rho}$ |
|---|---|---|---|---|---|---|---|---|---|
| lp_cre_a | 3,516 | 7,248 | 18,168 | 14 | 1 | 2.51 | 360 | 0 | 5.17 |
| lp_dfl001 | 6,071 | 12,230 | 35,632 | 14 | 1 | 2.91 | 228 | 2 | 5.87 |
| lp_ken_11 | 14,694 | 21,349 | 49,058 | 3 | 1 | 2.30 | 122 | 1 | 3.34 |
| lp_stocfor3 | 16,675 | 23,541 | 76,473 | 18 | 1 | 3.25 | 15 | 1 | 4.59 |
| lp_ken_13 | 28,632 | 42,659 | 97,246 | 3 | 1 | 2.28 | 170 | 1 | 3.40 |
| lp_pds_10 | 16,558 | 49,932 | 107,605 | 3 | 1 | 2.16 | 96 | 1 | 6.50 |
| lp_maros_r7 | 3,136 | 9,408 | 144,848 | 46 | 1 | 15.4 | 48 | 5 | 46.2 |
| lhr10 | 10,672 | 10,672 | 232,633 | 36 | 1 | 21.8 | 63 | 1 | 21.8 |
| lp_pds_20 | 33,874 | 108,175 | 232,647 | 3 | 1 | 2.15 | 96 | 0 | 6.87 |
| lp_cre_d | 8,926 | 73,948 | 246,614 | 13 | 1 | 3.33 | 808 | 0 | 27.6 |
| lp_cre_b | 9,648 | 77,137 | 260,785 | 14 | 1 | 3.38 | 844 | 0 | 27.0 |
| e30r2000 | 9,661 | 9,661 | 306,356 | 62 | 8 | 31.7 | 62 | 8 | 31.7 |
| lhr14 | 14,270 | 14,270 | 307,858 | 36 | 1 | 21.6 | 63 | 1 | 21.6 |
| lp_ken_18 | 105,127 | 154,699 | 358,171 | 3 | 1 | 2.31 | 325 | 1 | 3.40 |
| af23560 | 23,560 | 23,560 | 484,256 | 21 | 10 | 20.6 | 21 | 11 | 20.6 |
| e40r0100 | 17,281 | 17,281 | 553,956 | 62 | 8 | 32.1 | 62 | 8 | 32.1 |
| cage11 | 39,082 | 39,082 | 559,722 | 31 | 3 | 14.3 | 31 | 3 | 14.3 |
| lhr34 | 35,152 | 35,152 | 764,014 | 36 | 1 | 21.7 | 63 | 1 | 21.7 |
| lhr71c | 70,354 | 70,304 | 1,528,092 | 36 | 1 | 21.7 | 63 | 1 | 21.7 |
| cage12 | 130,228 | 130,228 | 2,032,536 | 33 | 5 | 15.6 | 33 | 5 | 15.6 |
| lp_osa_07 | 1,118 | 25,067 | 144,812 | 6 | 1 | 5.78 | 17,613 | 18 | 130 |
| lp_fit2d | 25 | 10,524 | 129,042 | 17 | 1 | 12.3 | 10,500 | 1,427 | 5,162 |

Tabela 2: Os resultados do *AGCI*

| Características das matrizes | | | | | | Tempos de execução das heurísticas($s$) | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | Os tempos reportados em [4]($s$) | | | Os tempos do *AGCI*($s$) | | |
| Matriz | $\frac{\lvert V_1\rvert}{\lvert E_c\rvert}$ | $\Delta$ | $\delta$ | $\overline{\delta}$ | $K(\rho_{\max})$ | $T_{Gc}$ | $T_{col}$ | $T_{tot}$ | $*T_{Gc}$ | $*T_{col}$ | $*T_{tot}$ |
| cre_a | $\frac{7,248}{253,411}$ | 454 | 1 | 69,9 | 360(360) | 0 | 0,01 | 0,01 | 0,3 | 0,01 | 0,31 |
| dfl001 | $\frac{12,23}{250,976}$ | 423 | 2 | 41 | 228(228) | 4 | 0,01 | 4,01 | 0,28 | 0,01 | 0,29 |
| ken_11 | $\frac{21,349}{459,921}$ | 138 | 1 | 43,1 | 130(122) | 2 | 0,01 | 2,01 | 0,5 | 0,03 | 0,53 |
| stocfor3 | $\frac{23,541}{125,969}$ | 39 | 1 | 10,7 | 16(15) | 0 | 0,01 | 0,01 | 0,12 | 0 | 0,12 |
| ken_13 | $\frac{42,659}{1,158,664}$ | 186 | 2 | 54,3 | 176(170) | 10 | 0,01 | 10 | 1,37 | 0,07 | 1,44 |
| pds_10 | $\frac{49,932}{594,681}$ | 106 | 1 | 23,8 | 96(96) | 9 | 0,01 | 9,01 | 0,51 | 0,04 | 0,55 |
| maros_r7 | $\frac{9,408}{610,76}$ | 314 | 4 | 129 | 74(48) | 9 | 0,01 | 9,01 | 1,25 | 0,04 | 1,29 |
| lhr10 | $\frac{10,672}{431,411}$ | 101 | 1 | 80,8 | 65(63) | 6 | 0,02 | 6,02 | 0,71 | 0,03 | 0,74 |
| pds_20 | $\frac{108,175}{1,325,891}$ | 115 | 1 | 24,5 | 96(96) | 12 | 0,02 | 12 | 0,51 | 0,04 | 0,55 |
| cre_d | $\frac{73,948}{21,347,885}$ | 1,124 | 2 | 577 | 813(808) | 78 | 0,25 | 78,2 | 25,79 | 1,23 | 27,02 |
| cre_b | $\frac{77,137}{20,852,569}$ | 1,168 | 2 | 540 | 845(844) | 270 | 0,75 | 271 | 25,59 | 1,2 | 26,79 |
| e30r2000_MM | $\frac{9,661}{688,848}$ | 209 | 42 | 143 | 65(62) | 14 | 0,02 | 14 | 1,03 | 0,03 | 1,06 |
| lhr14 | $\frac{14,27}{572,463}$ | 101 | 1 | 80,2 | 65(63) | 11 | 0 | 11 | 0,96 | 0,03 | 0,99 |
| ken_18 | $\frac{154,699}{8,412,174}$ | 340 | 1 | 109 | 330(325) | 0,51 | 0,15 | 51,2 | 13,31 | 0,53 | 13,84 |
| af23560 | $\frac{23,56}{1,210,004}$ | 107 | 41 | 103 | 32(21) | 14 | 0,02 | 14 | 1,76 | 0,09 | 1,85 |
| e40r0100_MM | $\frac{17,281}{1,254,328}$ | 209 | 42 | 145 | 66(62) | 17 | 0,01 | 17 | 2,2 | 0,06 | 2,26 |
| e40r0100 | $\frac{17,281}{1,254,328}$ | 209 | 42 | 145 | 95(62) | - | - | - | 1,85 | 0,07 | 1,92 |
| cage11 | $\frac{39,082}{1,887,384}$ | 340 | 7 | 96,6 | 81(31) | 20 | 0,02 | 20 | 3,07 | 0,17 | 3,24 |
| lhr34 | $\frac{35,152}{1,417,888}$ | 101 | 1 | 80,7 | 65(63) | 27 | 0,03 | 27 | 2,32 | 0,11 | 2,43 |
| lhr71c | $\frac{70,304}{2,835,968}$ | 101 | 1 | 80,7 | 65(63) | 47 | 0,04 | 47 | 4,65 | 0,18 | 4,83 |
| cage12 | $\frac{130,228}{7,550,823}$ | 400 | 12 | 116 | 96(33) | 72 | 0,11 | 72,1 | 1,37 | 0,71 | 2,08 |
| Total1 | $\frac{930,536}{73,242,018}$ | | | | 3,764 (3,573) | 673,5 | 1.51 | 675 | 89,45 | 4,68 | 94,13 |
| osa 07 | $\geq 10^8$ | | | | | | | | | | |
| fit2d | $\geq 10^8$ | | | | | | | | | | |

Tabela 3: Os resultados do $AGB$

| Características das matrizes | | | | | | Tempos de execução das heurísticas($s$) | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | Os tempos reportados em [4]($s$) | | | Os tempos do $AGB$($s$) | | |
| Matriz | $\frac{\lvert V_1+V_2\rvert}{\lvert E_b\rvert}$ | $\Delta$ | $\delta$ | $\overline{\delta}$ | $K(\rho_{\max})$ | $T_{Gc}$ | $T_{col}$ | $T_{tot}$ | $*T_{Gc}$ | $*T_{col}$ | $*T_{tot}$ |
| cre_a | 10,764<br>18,168 | 360 | 0 | 3,38 | 360(360) | 0 | 0,01 | 0,01 | 0 | 0,03 | 0,03 |
| d.001 | 18,301<br>35,632 | 228 | 1 | 3,89 | 228(228) | 0 | 0,01 | 0,01 | 0,04 | 0,01 | 0,05 |
| ken_11 | 36,043<br>49,058 | 122 | 1 | 2,72 | 130(122) | 1 | 0 | 1 | 0,05 | 0,03 | 0,08 |
| stocfor3 | 40,216<br>76,473 | 18 | 1 | 3,8 | 16(15) | 1 | 0,03 | 1,03 | 0,07 | 0,01 | 0,08 |
| ken_13 | 71,291<br>97,246 | 170 | 1 | 2,73 | 176(170) | 1 | 0,02 | 1,02 | 0,07 | 0,1 | 0,17 |
| pds_10 | 66,49<br>107,605 | 96 | 1 | 3,23 | 96(96) | 1 | 0,01 | 1,01 | 0,11 | 0,03 | 0,14 |
| maros_r7 | 12,544<br>144,848 | 48 | 1 | 23,1 | 74(74) | 1 | 0,07 | 1,07 | 0,17 | 0,17 | 0,34 |
| lhr10 | 21,344<br>232,633 | 63 | 1 | 21,8 | 65(63) | 1 | 0,1 | 1,1 | 0,26 | 0,28 | 0,54 |
| pds_20 | 142,049<br>232,647 | 96 | 0 | 3,28 | 96(96) | 2 | 0,03 | 2,03 | 0,14 | 0,12 | 0,26 |
| cre_d | 82,874<br>246,614 | 808 | 0 | 0,95 | 813(808) | 2 | 0,34 | 2,34 | 0,38 | 1,22 | 1,6 |
| cre_b | 86,785<br>260,785 | 844 | 0 | 6,01 | 845(844) | 2 | 0,34 | 2,34 | 1,6 | 1,21 | 2,81 |
| e30r2000_MM | 19,322<br>306,356 | 62 | 8 | 31,7 | 65(62) | 2 | 0,07 | 2,07 | 0,36 | 0,28 | 0,64 |
| lhr14 | 28,54<br>307,858 | 63 | 1 | 21,6 | 65(63) | 2 | 0,11 | 2,11 | 0,34 | 0,37 | 0,71 |
| ken_18 | 259,826<br>358,171 | 325 | 1 | 2,76 | 330(325) | 7 | 0,23 | 7,23 | 0,36 | 0,56 | 0,92 |
| af23560 | 47,12<br>484,256 | 21 | 10 | 20,6 | 32(21) | 3 | 0,09 | 3,09 | 0,48 | 0,23 | 0,71 |
| e40r0100_MM | 34,562<br>553,956 | 62 | 8 | 32,1 | 66(62) | 4 | 0,13 | 4,13 | 0,66 | 0,48 | 1,14 |
| e40r0100 | 34,562<br>553,956 | 62 | 8 | 32,1 | 95(62) | - | - | - | 0,67 | 0,52 | 1,19 |
| cage11 | 78,164<br>559,722 | 31 | 3 | 14,3 | 81(31) | 4 | 0,12 | 4,12 | 0,68 | 0,28 | 0,96 |
| lhr34 | 70,304<br>764,014 | 63 | 1 | 21,7 | 65(63) | 6 | 0,25 | 6,25 | 0,9 | 0,9 | 1,8 |
| lhr71c | 140,608<br>1,528,092 | 63 | 1 | 21,7 | 65(63) | 12 | 0,45 | 12,4 | 1,78 | 1,84 | 3,62 |
| cage12 | 260,456<br>2,032,536 | 33 | 5 | 15,6 | 96(33) | 15 | 0,37 | 15,4 | 2,75 | 1,75 | 4,5 |
| Total1 | 1,527,603<br>8,396,670 | 3,576 | 45 | 256,95 | 3,764(3573) | 67 | 2,78 | 69,76 | 11,2 | 9,9 | 21,1 |
| osa_7 | 26,185<br>144,812 | 77,613 | 1 | 11,1 | 17,613(17,613) | 0 | 5,07 | 5,07 | 5,75 | 25,6 | 31,35 |
| fit2d | 10,549<br>129,042 | 10,500 | 1 | 24,5 | 10,501(10,500) | 0 | 4,63 | 4,63 | 5,18 | 36,82 | 42 |
| Total2 | 36,734<br>273,854 | | | | 28,114(28,113) | 0 | 9,7 | 9,7 | 10,93 | 62,42 | 73,35 |

Tabela 4: Os tempos de liberação das estruturas de dados

| Matriz | liberação($s$) | |
|---|---|---|
| | *GB* | *GCI* |
| cre_a | 0,01 | 0,2 |
| d.001 | 0,03 | 0,18 |
| ken_11 | 0,05 | 0,32 |
| stocfor3 | 0,08 | 0,11 |
| ken_13 | 0,80 | 0,82 |
| pds_10 | 0,10 | 0,45 |
| maros_r7 | 0,09 | 0,45 |
| lhr10 | 0,15 | 0,31 |
| pds_20 | 0,14 | 0,46 |
| cre_d | 0,19 | 16,09 |
| cre_b | 0,20 | 15,6 |
| e30r2000 | 0,19 | 0,45 |
| lhr14 | 0,20 | 0,4 |
| ken_18 | 0,34 | 6,06 |
| af23560 | 0,32 | 0,82 |
| e40r0100 | 0,36 | 0,84 |
| e40r0100_MM | 0,36 | 0,9 |
| cage11 | 0,42 | 1,4 |
| lhr34 | 0,50 | 1 |
| lhr71c | 1,02 | 2 |
| cage12 | 1,58 | 5,84 |
| Total1 | 7,13 | 54,7 |
| osa_7 | 25,60 | - |
| fit2d | 36,82 | - |
| Total2 | 62,42 | - |

Para termos uma noção do desempenho das heurísticas, colocamos os tempos de execução do artigo [4] nas tabelas 2 e 3. O tempo de construção do grafo, coloração e tempo total dos algoritmos do artigo [4] são dados em segundos por $T_{Gc}$, $T_{col}$ e $T_{tot}$. Os tempos correspondentes dos nossos programas vem precedido por um asterisco. Os autores de [4] não incluíram o tempo de liberação da estrutura de dados, logo, para facilitar a comparação, nosso tempo total ($*T_{tot}$) também é a soma do tempo de construção ($*T_{Gc}$) e de coloração ($*T_{col}$). Os tempos de liberação das estruturas são apresentados em separado, na Tabela 4.

O número de cores usadas está designado por $K$ e, entre parênteses, colocamos o número máximo de não nulos por linha $\rho_{\max}$ para termos uma idéia da qualidade da coloração, relembrando da seção 4.2 que

$$\rho_{\max} \leq \omega(G) \leq \chi(G).$$

Com $\rho_{\max}$ como limite inferior, será necessário no mínimo $\rho_{\max}$ cores para se obter uma coloração distância-1. Lembramos, conforme visto na Seção 7.4, que a coloração distância-1 do grafo coluna-interseção obtida pelo $AGCI$ coincide com a coloração do grafo bipartido obtida pelo $AGB$, assumindo que os nós-colunas são percorridos na mesma ordem em ambos os algoritmos. Logo as colunas "$K(\rho_{\max})$" das Tabelas 2 e 3 contêm os mesmo números.

Ao executar o $AGCI$ nos exemplos osa_07 e fitd2d foram encontrados cliques com mais

de 10,000 nós[6], resultando na geração de mais de 100 milhões de arestas na estrutura de dados. Consequentemente a implementação desta heurística sofreu um *overflow* nos dois exemplos. Por isso, a linha Total1 na Tabela 2 inclui o total de cores usadas e tempos de execução das matrizes, com exceção das matrizes osa_07 e fitd2d. Enquanto o algoritmo do grafo bipartido concluiu a coloração nessas duas matrizes, a execução foi muito lenta em comparação com os tempos relativos às outras matrizes. Por isso, na Tabela 3 temos o total de cores usadas e tempos de execução de osa_07 e fitd2d na linha Total2, e na linha Total1 os totais das outras matrizes. Essas duas matrizes são linearmente muito densas, ou seja, têm o maior valor de $\overline{\rho}$ da coleção, 130 e 5,162 respectivamente, o que faz com que cada coluna seja estruturalmente não ortogonal a muitas outras colunas, proporcionando grafos de coluna-interseção com arestas demais para a memória virtual. Dificultou também a coloração distância-2 do grafo bipartido, dado que cada nó-coluna tinha muitos vizinhos distância-2.

A Figura 11 ajuda a vizualizar a qualidade das colorações. As posições no eixo horizontal correspondem às matrizes, seguindo a ordem em que estão apresentadas nas tabelas, e o eixo vertical refere-se ao número de cores (lembramos que ambos os algoritmos obtém colorações com o mesmo número de cores, excetuando-se as matrizes osa_07 e fitd2d). As extremidades dos segmentos verticais correspondem aos limitantes inferiores e superiores para $\omega(G)$, e o losango vermelho está posicionado sobre o número de cores utilizado pelas heurísticas na coloração. Podemos ver que a maioria das colorações tiveram $K$ próximo de $\rho_{\max}$, se não exatamente igual.

Incluímos também a Figura 12, com os valores de $(K - \rho_{\max})/\rho_{\max}$, para ter uma noção das diferenças relativas entre $K$ e $\rho_{max}$. Nessa figura, há apenas quatro casos marcantes, onde o maior valor foi da matriz cage12, com 1,9 ou seja, nenhuma coloração usou mais que três vezes o limite inferior $\rho_{max}$. A grande maioria, 19 das 23 matrizes, tiveram valores inferiores a 0,07.

## 12.1 Divergência em e40r0100

Tivemos resultados idênticos aos de [4] em relação ao número de arestas, nós e cores usadas em cada algoritmo, exceto para a matriz e40r0100. Nessa matriz tanto o $AGCI$ e o $GB$ usaram 95 cores, enquanto os algoritmos de [4] usaram 66 cores. Além disso, não encontramos a matriz irmã dessa e30r2000 no sítio da "University of Florida Sparse matrix collection" [6]. Não havendo nada que distinguisse essa matriz das outras, e sendo improvável um mesmo erro nas duas heurísticas que provocasse diferença de resultados em apenas uma matriz,

[6]Isso já sabiamos dado que $\rho_{max}$ das matrizes osa_07 e fitd2d vale 17,613 e 10,500. E pela seção 4.2, $\rho_{max} \leq \omega(G)$

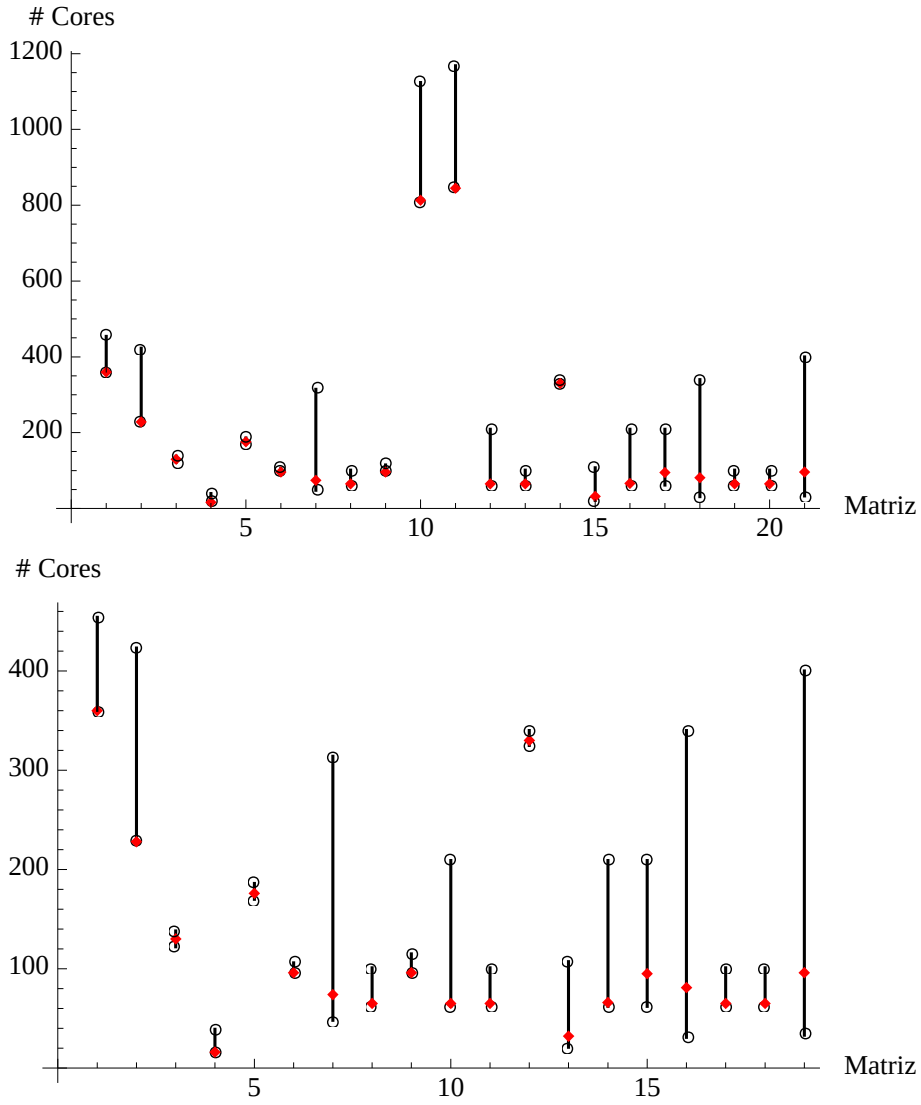



Figura 11: Limitantes inferior e superior, e número de cores utilizadas pelas heurísticas na coloração das matrizes-teste. Matrizes cre_b e cre_d excluídas do segundo gráfico, para melhorar visualização dos demais dados.

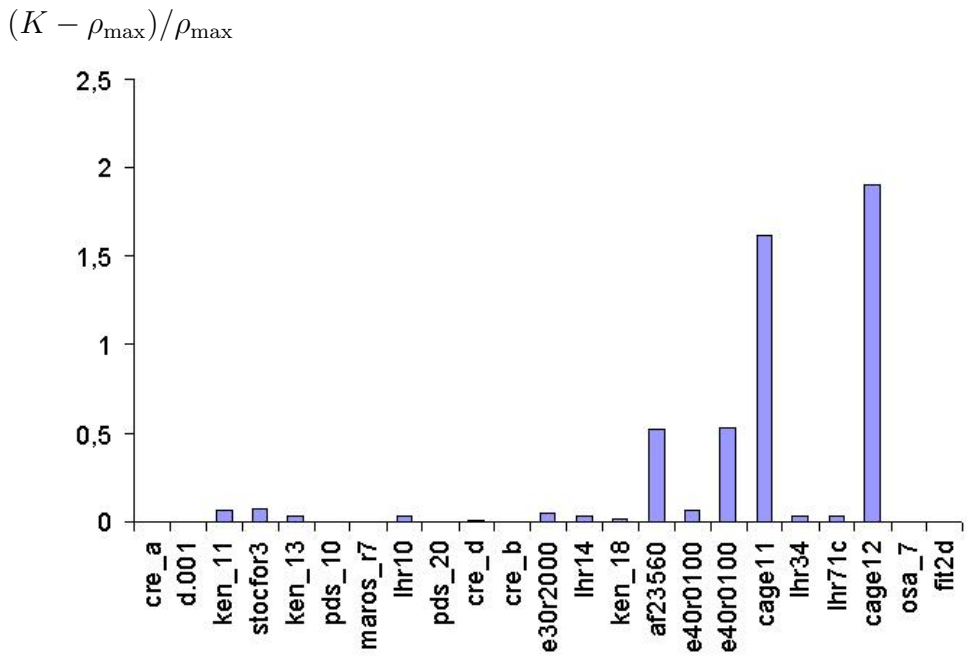



Figura 12: Os valores de $(K - \rho_{\max})/\rho_{\max}$ de cada matriz

levantamos a hipótese de que a e40r0100 tivesse sofrido alterações desde a publicação de [4] em 2005, e que e30r2000 tivesse sido removida do sítio. Entramos em contanto com o autor do banco de dados [6], e ele nos informou que a matriz e30r2000 nunca esteve na sua coleção e que e40r0100 não foi alterada. Com isso, e dado que nossos programas e os de [4] constróem grafos a partir de e40r0100 com o mesmo número de arestas e nós, concluímos que os autores de [4] obtiveram as matrizes e30r2000 e e40r0100 de outro local, e que a versão de e40r0100 utilizada pelos autores possivelmente possui uma ordenação das colunas diferente da versão em [6]. Isso proporcionaria grafos com o mesmo número de arestas e nós, mas ao executar a coloração dos nós-colunas, a ordem de atravessar os nós será outra, podendo resultar numa coloração diferente. Fomos à procura dessas matrizes na internet e as encontramos no sítio "MatrixMarket" [8]. Com estas últimas obtivemos resultados idênticos aos de [4]. Nas Tabelas 2 e 3, e40r0100 é a versão obtida do "University of Florida Sparse matrix collection", enquanto as matrizes e40r0100_MM e e30r2000_MM foram obtidas do MatrixMarket.

Na próxima parte desse projeto, faremos uso de outras ordens de atravessar os nós para coloração, e vamos verificar se conseguimos obter 66 cores na coloração da matriz e40r0100.

## 12.2 Comparações entre os resultados do *AGCI* e do Gebremedhin et al. [4]

Na Figura 14 apresentamos $T_{tot}$ e $*T_{tot}$ associado ao *GCI*. Em média nossa implementação desta heurística foi 9,3 vezes mais rápida do que a descrita em [4], onde o exemplo mais divergente foi a matriz cage12, tendo uma execução 34,7 vezes mais rápida. A discrepância

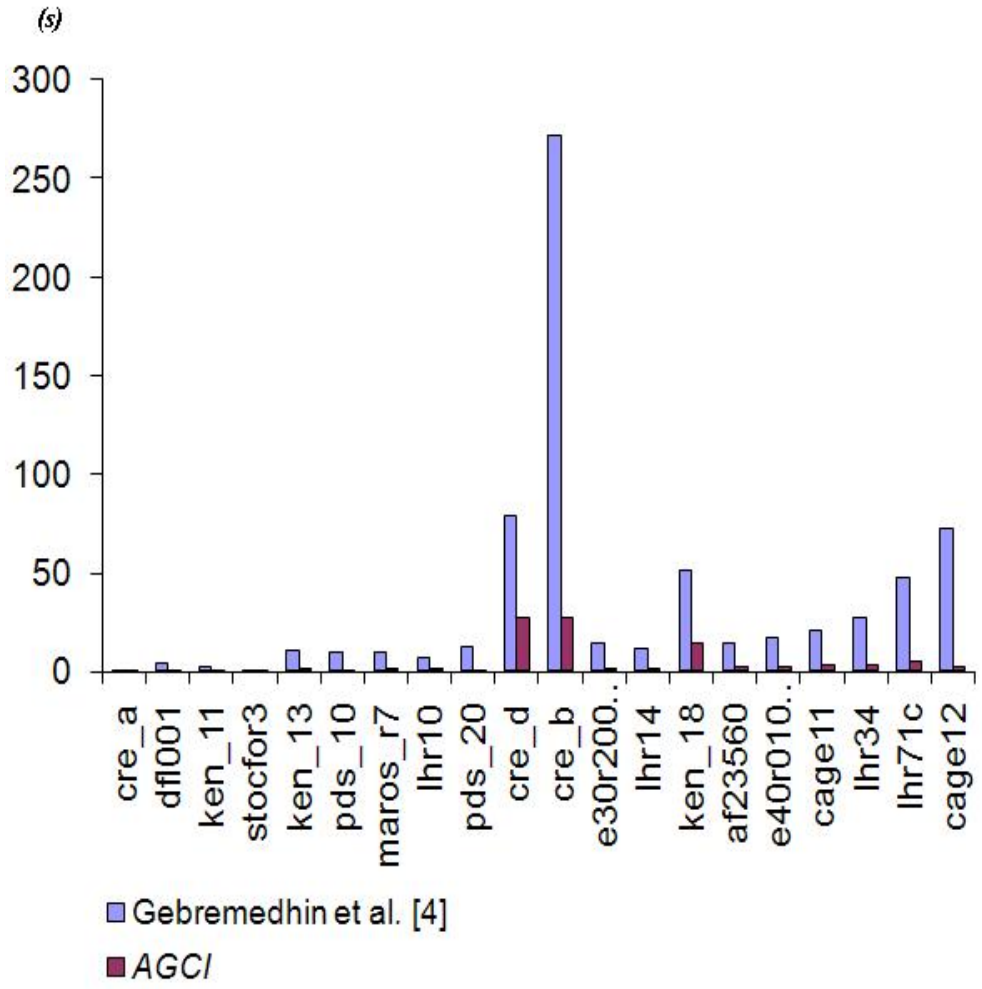



Figura 13: Tempos de execução de Gebremedhin et al. [4] e o *AGCI*

de tempos parece decorrer da etapa de construção do grafo. Nesta etapa nossa implementação foi em média 10,9 vezes mais veloz que a de [4], e o exemplo mais marcante foi novamente a matriz cage12, cuja construção foi 52,6 vezes mais rápida. O autores de [4] mencionam que constróem o grafo coluna-interseção a partir do grafo bipartido correspondente, usando o fato que todo vizinho distância-2 de um nó-coluna no grafo bipartido é um vizinho no grafo coluna-interseção, diferentemente do nosso algoritmo, que constrói o *GCI* diretamente da matriz. O desempenho da implementação de [4] decai muito nos grafos maiores, indicando

um tempo assintótico, em relação tamanho de grafo, maior do que o de nossa implementação. Mas, sem maiores detalhes de implementação e dado os diferentes sistemas operacionais e processadores, nossos resultados não são diretamente comparáveis aos de [4].

## 12.3 Comparações entre os resultados do *AGB* e do Gebremedhin et al. [4]

Na Figura 14 apresentamos $T_{tot}$ e $*T_{tot}$ associado ao *GB*. Como podemos ver na Figura 14, nossos tempos de execução e os de [4] apresentam uma distribuição consideravelmente diferente indicando implementações diferentes. Em média, nosso tempo total foi 1,2 vezes maior,

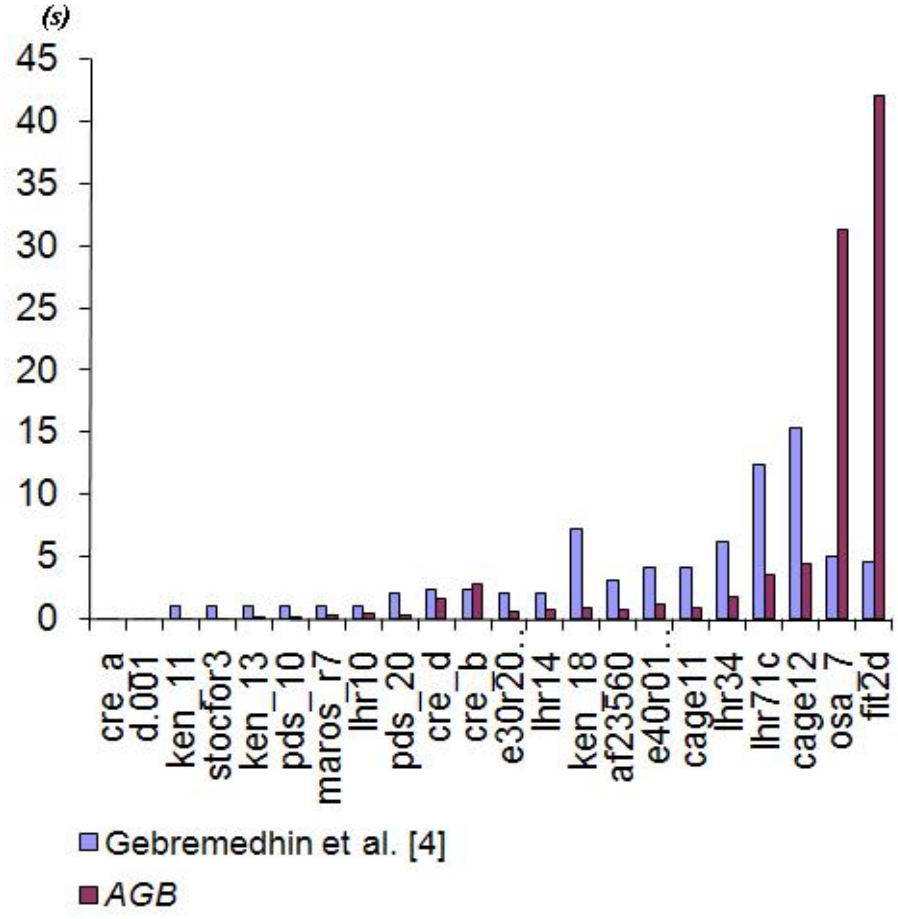



Figura 14: Tempos de execução de [4] e do *AGB*

apesar do fato que nossa implementação foi mais rápida em 20 das 23 matrizes. As duas matrizes que fizeram a diferença foram osa_7 e fit2d, onde a implementação de [4] foi muito mais eficiente do que a nossa, num claro contraste com os demais exemplos. Nossa maior deficiência foi no tempo de coloração, 5,8 vezes mais longo que os de [**?**]. Novamente, a maior

descrepância ocorre nas matrizes osa_7 e fit2d, cujos tempos de coloração são de 5 a 8 vezes maiores, respectivamente. Os autores [4] devem ter uma rotina mais eficiente para coloração distância-2, e isso fica mais evidente nos grafos maiores e mais densos.

## 12.4 Comparações entre o *AGCI* e o *AGB*

O tamanho total, $|V| + |E|$, do grafo coluna-interseção excedeu o grafo bipartido em todos os casos, sendo, em média, 7,3 vezes maior que o grafo bipartido, Figura 15.

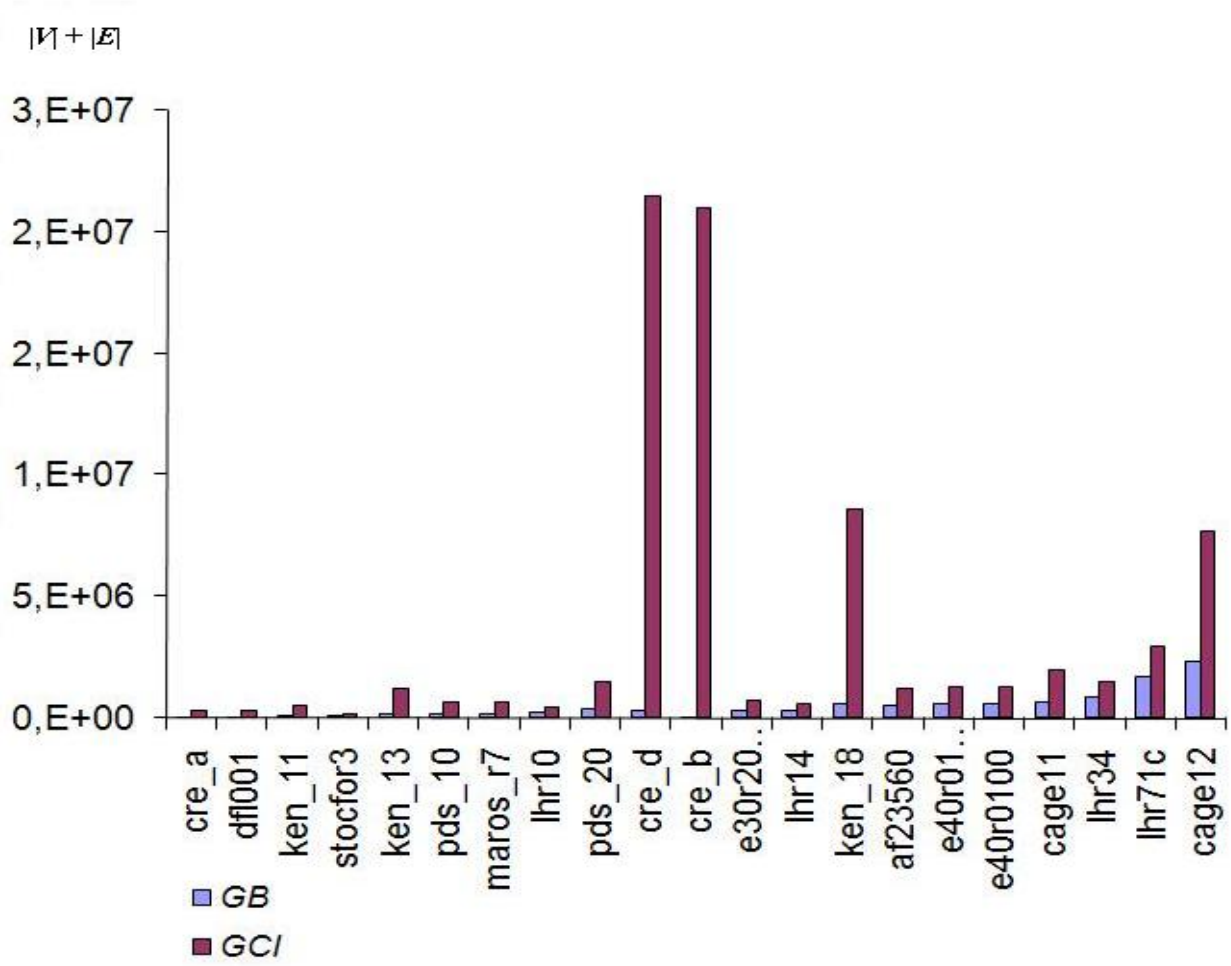



Figura 15: Os valores $|V| + |E|$ do grafo coluna-interseção e do grafo bipartido

Como consequência, mais tempo é gasto na construção e na liberação da estrutura de dados 4. O *AGCI* demorou em média 4,2 vezes mais do que o *AGB*, tendo apenas um caso em que o *AGCI* teve uma execução mais rápida, na coloração da matriz cage12, último par de barras na Figura 16.

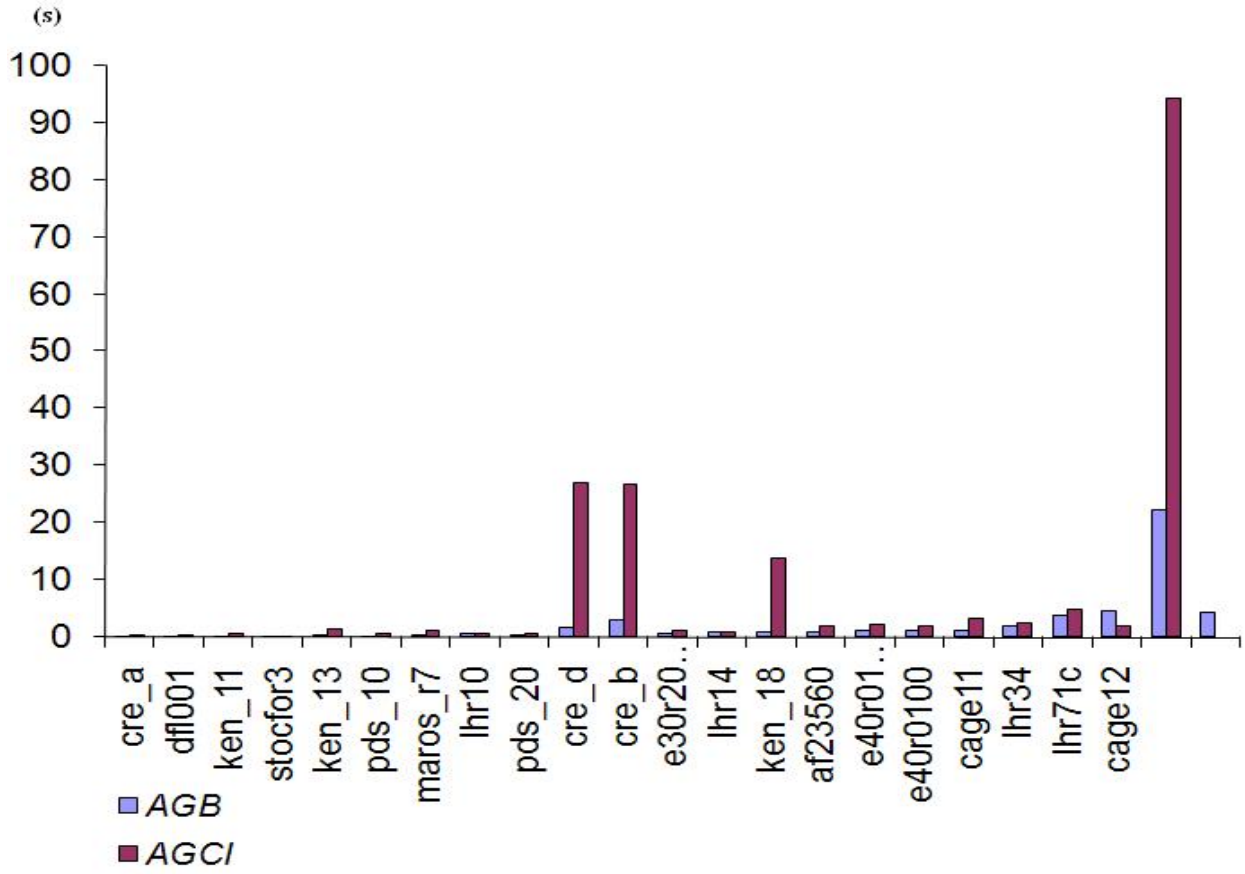



Figura 16: Tempos de execução do *AGCI* e do *AGB*

Então os resultados mostram que o *AGB* é superior, mas não tanto como nos resultados apresentados em [4], onde a sua implementação do algoritmo associado ao grafo bipartido era 10 vezes mais rápido em relação ao algoritmo associado ao grafo coluna-interseção.

# 13 Conclusões e trabalho futuro

Os resultados indicam que possivelmente formulamos uma rotina mais eficiente para a construção do grafo coluna-interseção. Entretanto, este fato isolado não foi suficiente para tornar a heurística correspondente mais atraente, dado que o método do grafo bipartido se mostrou superior em termos de flexibilidade (5.2), uso de memória e tempo de execução. Em termos de qualidade, os dois algoritmos usaram exatamente o mesmo número de cores, e, de fato, se atravessamos os nós-colunas do grafo bipartido e do grafo coluna-interseção na mesma ordem, a coloração é a mesma, como mostramos na Secção 5.2. E pelas Figuras 12 e 11 vimos que o número de cores efetivamente utilizado pelas heurísticas ficou próximo do limitante inferior, sendo igual (portanto, ótimo) em 26% dos casos.

O método associado ao grafo coluna-interseção está em uso desde a sua apresentação por

Coleman e Moré [2] em 1983, e é surpreendente o aparecimento de um método superior em 2002 [5] dada a importância de se calcular derivadas eficientemente. E apesar dos autores não apresentarem uma teoria concreta, os testes empíricos feitos em [4] e aqui confirmados, deixam claro o ganho de tempo ao usar o *GB*.

No próximo relatório, usaremos outras ordens de atravesar os nós para tentar melhorar a qualidade das colorações. Investigaremos uma rotina mais eficiente para a coloração distância-2, tentando obter um tempo de execução mais próximo do de [4].

# Referências


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

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

[3] A. R. Curtis, M. J. D. Powell, and J. K. Reid, "On the estimation of sparse Jacobian matrices", J. Inst. Math. Appl., 13 (1974), pp. 117119.

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

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

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

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

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