Métodos de Ordenação e Armazenamento de Matrizes Esparsas

Antonio J. A. Simões Costa e Katia C. de Almeida

1. Introdução

O número de barras existente em um sistema de potência pode ser muito elevado (por exemplo, o sistema Sul-Sudeste brasileiro possui cerca de 1000 barras). Num sistema de potência típico existem em média três linhas conectadas a cada barra. Como conseqüência, as matrizes presentes em vários modelos matemáticos utilizados nos estudo destes sistemas são de grande dimensão e muito esparsas (isto é, com poucos elementos não nulos em cada linha ou coluna). Devido à sua grande dimensão, se torna praticamente impossível armazenar todos os elementos destas matrizes. É necessário o uso de técnicas apropriadas para sua armazenagem e manipulação. Estas técnicas são conhecidas como técnicas de esparsidade.

No estudo de técnicas de esparsidade, é útil se estabelecer correspondências entre algumas noções de Teoria de Grafos e matrizes simétricas esparsas. Porisso, iniciaremos este estudo com uma revisão de noções básicas de Teoria de Grafos. Segue-se o estudo de três estruturas de dados freqüentemente utlizados para o armazenamento de matrizes esparsas:

- Tabela de Conexão;
- Listas de Adjacências;
- Listas Encadeadas.

De posse destas estruturas, estaremos aptos a elaborar algoritmos para realizar algumas operações elementares envolvendo matrizes e vetores esparsos.

Em seguida, abordaremos o problema da fatoração LU de sistemas lineares esparsos. Ficará evidente a necessidade de se utilizar esquemas de ordenação que buscam minimizar a criação de novos elementos não nulos (*fill-ins*) nos fatores triangulares. Será então introduzido o método do Mínimo Grau (ou esquema "Tinney II") e serão feitas aplicações a sistemas de potência.

2. Matrizes Simétricas e Grafos de Adjacência

Inicialmente será suposto que as matrizes a serem manipuladas são simétricas e positivas definidas. Se uma matriz A $(N \times N)$ é simétrica e definida positiva, então $a_{ii} > 0, i = 1, ..., N$.

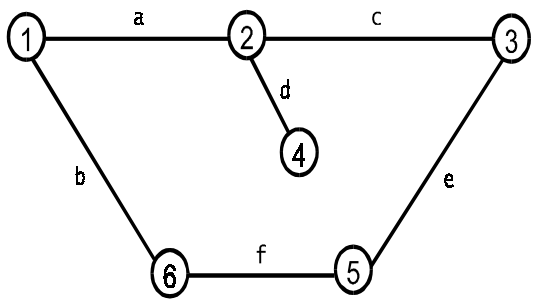
Um Grafo $G = (X, E)$ consiste de um conjunto finito X de *nós* ou *vértices* e um conjunto E de *arestas ou ramos*, que são pares não ordenados de vértices.

É possível se construir um grafo de adjacência de A. Este grafo contém N vértices, denotados por $\{x_1, x_2, ..., x_N\}$, um para cada linha/coluna de A e seu conjunto E de arestas se compõe de todos os pares $\{x_i, xj\}$ tais que $a_{ij} = a_{ji} \neq 0$.

Exemplo 1. *A figura abaixo apresenta um exemplo de uma matriz esparsa* 6×6 *e o grafo de adjacência associado.*

$$\begin{bmatrix} (\mathbf{1}) & \mathbf{a} & & & & \mathbf{b} \\ \mathbf{a} & (\mathbf{2}) & \mathbf{c} & \mathbf{d} & & \\ & \mathbf{c} & (\mathbf{3}) & & \mathbf{e} & \\ & \mathbf{d} & & (\mathbf{4}) & & \\ & & \mathbf{e} & & (\mathbf{5}) & \mathbf{f} \\ \mathbf{b} & & & & \mathbf{f} & (\mathbf{6}) \end{bmatrix}$$

Exemplo de matriz esparsa



Grafo correspondente à matriz ao lado

Quando dois vértices são conectados por uma aresta, dizemos que são *adjacentes*. O conjunto adjacente de um vértice x_i é definido como o conjunto de vértices diferentes de x_i, adjacentes a x_i. Da mesma forma, o conjunto adjacente de um subconjunto Y, do conjunto de vértices X, é o conjunto de todos os vértices adjacentes aos vértices contidos em Y mas que não estão em Y.

Exemplo 2. *No grafo do Exemplo 1, podemos facilmente verificar que* $Adj(x_2) = \{x_1, x_3, x_4\}$ *e* $Adj(\{x_2, x_3\}) = \{x_1, x_4, x_5\}$.

O *Grau* de um subconjunto Y de X é o número de elementos em $Adj(Y)$.

Define-se como um *caminho* de comprimento $l \geq 0$, de um vértice x a um vértice y, como um conjunto ordenado de $l + 1$ vértices distintos $(v_1, v_2, ..., v_{l+1})$ tal que:

$$v_1 = x$$
$$v_{i+1} \in Adj(v_i), i = 1, ..., l$$
$$v_{l+1} = y$$

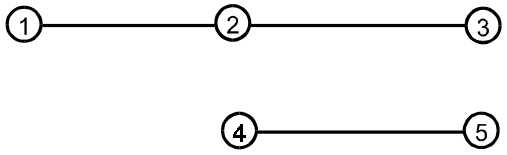
Um grafo G é conexo se existe um caminho entre quaisquer dois vértices de G.

Observação 1. *Note que a uma matriz bloco-diagonal corresponde um grafo desconexo.*

Exemplo 3. *A figura abaixo mostra uma matriz esparsa bloco-diagonal e o grafo de adjacência correspondente.*

$$\begin{bmatrix} (\mathbf{1}) & * & & & & \\ * & (\mathbf{2}) & * & & & \\ & * & (\mathbf{3}) & & & \\ & & & (\mathbf{4}) & * & \\ & & & * & (\mathbf{5}) \end{bmatrix}$$

Matriz esparsa bloco-diagonal



Grafo correspondente à matriz ao lado.

3. Estruturas de Dados para Representação de Grafos e Matrizes Esparsas

3.1. Tabelas de Conexão

Para uma matriz $(N \times N)$ esta tabela terá N linhas e m colunas, sendo

$$m = \max \{Grau(x) \mid x \in X\} \tag{3.1}$$

Uma tabela com as mesmas dimensões que a anterior pode ser construída para armazenar os valores da matriz A. As tabelas que armazenam os índices de coluna dos elementos não-nulos em cada linha de A (ou, equivalentemente, os vértices adjacentes a cada vértice do grafo de A) e seus valores numéricos serão denotadas por **VIZ** e **VAL**, respectivamente. Um arranjo de comprimento N pode ser usado para armazenar os elementos diagonais de A.

Exemplo 4. *A tabela de conexão para a matriz/grafo do Exemplo 1 é mostrada abaixo.*

Nó	***VIZ***			***VAL***		
1	***2***	***6***	***-***	***a***	***b***	***-***
2	***1***	***3***	***4***	***a***	***c***	***d***
3	***2***	***5***	***-***	***c***	***e***	***-***
4	***2***	***-***	***-***	***d***	***-***	***-***
5	***3***	***6***	***-***	***e***	***f***	***-***
6	***1***	***5***	***-***	***b***	***f***	***-***

É fácil verificar que o esquema de armazenamento baseado em tabelas de conexão é ineficiente se um número significativo de vértices tém grau muito menor do que m.

3.2. Lista de Adjacências

Formada pelos arranjos:

- **ADJ**: contém as listas de adjacências para todos os vértices do grafo de A de forma ordenada; tem comprimento igual a $2 \times (\textit{número de arestas})$;
- **XADJ**: contém apontadores para o começo de cada lista de adjacências em ADJ; tem comprimento igual a $N + 1$;
- **VAL**: arranjo paralelo a **ADJ** contendo os valores numéricos dos elementos de A fora da diagonal;
- **DIAG**: de comprimento N, contém os elementos diagonais de A.

Note que o arranjo **XDAJ** possui $N+1$ posições para que o final da lista seja indicado.

Exemplo 5. *A lista de adjacências para a matriz/grafo do Exemplo 1 é mostrada abaixo.*

XADJ	=	*1*	*2*	*3*	*4*	*5*	*6*	*7*
		1	***3***	***6***	***8***	***9***	***11***	***13***

ADJ	=	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*
		2	***6***	***1***	***3***	***4***	***2***	***5***	***2***	***3***	***6***	***1***	***5***

VAL	=	**a**	**b**	**a**	**c**	**d**	**c**	**e**	**d**	**e**	**f**	**b**	**f**

DIAG	=	$\mathbf{a}_{11}$	$\mathbf{a}_{22}$	$\mathbf{a}_{33}$	$\mathbf{a}_{44}$	$\mathbf{a}_{55}$	$\mathbf{a}_{66}$

Para examinar todos os vértices adjacentes de um determinado vértice k a partir da lista de adjacências, o seguinte segmento de código pode ser usado:

```
⋮
x_inic = XADJ(k)
x_fim = XADJ(k + 1) − 1
se (x_inic ≤ x_fim)
    para j = x_inic : x_fim
        viz = ADJ(j)
        val = VAL(j)
        escrever viz, val
    fim
senão
    escrever "vértice k está isolado"
fim
⋮
```

3.3. Lista Encadeada

Os esquemas anteriores têm a desvantagem de exigir o conhecimento prévio do grau de cada vértice. Caso apareça um novo elemento não nulo na matriz (por exemplo, durante a fatoração LU), será difícil a sua inserção nas listas. Esta dificuldade é contornada introduzindo-se um arranjo de ligação **PROX**. A estrutura resultante é chamada de *Lista Encadeada* e corresponde aos seguintes arranjos.

- **PRIM**: aponta para o primeiro elemento da lista de vértices adjacentes a um dado vértice;
- **VIZ**: contém os nós adjacentes ao indexador de **PRIM**;
- **PROX**: aponta para o próximo nó adjacente em **VIZ**;
- **VAL**: guarda os valores numéricos, sendo paralelo a **VIZ** e a .**PROX**.

Além destes arranjos, defini-se o arranjo **DIAG** para armazenar os elementos da diagonal de A.

Exemplo 6. *A lista encadeada para a matriz/grafo do Exemplo 1 é mostrada abaixo.*

	PRIM
1)	***10***
2)	***6***
3)	***1***
4)	***12***
5)	***8***
6)	***5***

	VIZ	***VAL***	***PROX***
1)	***2***	***c***	***11***
2)	***6***	***f***	***0***
3)	***2***	***a***	***0***
4)	***3***	***c***	***7***
5)	***1***	***b***	***9***
6)	***1***	***a***	***4***
7)	***4***	***d***	***0***
8)	***3***	***e***	***2***
9)	***5***	***f***	***0***
10)	***6***	***b***	***3***
11)	***5***	***e***	***0***
12)	***2***	***d***	***0***

Para examinar os vértices adjacentes de um determinado vértice k a partir da lista encadeada, o seguinte segmento de código pode ser usado:

$$
\begin{array}{l}
\vdots \\
\ell = PRIM(k) \\
\textbf{enquanto } \ell \neq 0 \\
\quad viz = VIZ(\ell) \\
\quad val = VAL(\ell) \\
\quad \textbf{escrever } viz, val \\
\quad \ell = PROX(\ell) \\
\textbf{fim} \\
\vdots
\end{array}
$$

Observação 2. *Se houver espaço suficiente nos arranjos* ***VIZ****,* ***PROX*** *e* ***VAL****, novas arestas podem ser facilmente adicionadas à lista encadeada.*

Exemplo 7. *Para adicionar a aresta* $\{3, 6\}$ *com* $a_{36} = g$ *nos arranjos do exemplo anterior faz-se as seguintes modificações:*

$$
\begin{array}{l}
PROX(13) = 1,\ VIZ(13) = 6,\ VAL(13) = g,\ PRIM(3) = 13 \\
PROX(14) = 5,\ VIZ(14) = 3,\ VAL(14) = g,\ PRIM(6) = 14
\end{array}
$$

Um segmento de código a ser usado para fazer a inserção de uma novo elemento na posição $novo\_viz$ da linha ℓ, de valor $novo\_val$ é dado abaixo:

$$
\begin{array}{l}
\vdots \\
temp = PRIM(\ell) \\
PRIM(\ell) = idisp \\
VIZ(idisp) = novo\_viz \\
VAL(idisp) = novo\_val \\
PROX(idisp) = temp \\
\vdots
\end{array}
$$

Observação 3. *Nas aplicações de listas encadeadas em operações envolvendo matrizes esparsas, é normalmente necessário que a lista de vértices adjacentes a cada vértice esteja ordenada de forma crescente ou decrescente. Mesmo quando a lista encadeada original está assim ordenada, a inclusão de novos elementos em geral exige que sejam realizadas reordenações. Para isso, rotinas eficientes de reordenação da lista encadeada devem ser utilizadas.*

4. Esquemas de Armazenamento Compacto para Vetores e Matrizes Retangulares

A utilização de um grafo não-orientado para representar a estrutura de uma matriz é possível apenas no caso de matrizes quadradas simétricas. Matrizes quadradas assimétricas podem ser representadas por grafos ordenados, porém matrizes retangulares, incluindo vetores-linha e vetores-coluna, não podem ter suas estruturas associadas a grafos. Apesar disso, as estruturas de dados vistas anteriormente permanecem válidas, com pequenas alterações. Estas, na verdade, se resumem ao fato de que os elementos a_{ij}, para os quais $i = j$ são também armazenados nos arranjos compactos das estruturas de armazenamento, e não mais em um arranjo separado como o arranjo **DIAG** que mencionamos anteriormente.

4.1. Esquema de armazenamento Compacto para Vetores

Um vetor $\mathbf{v} \in \Re^n$ [1] é descrito de forma não ambígua através da especificação:

- do número de linhas n;
- do índice da linhas onde estão localizados os elementos não nulos;
- dos valores numéricos dos elementos.

A dimensão do vetor é um número inteiro, as localizações dos elementos não nulos podem ser feita em arranjos de números inteiros e os valores podem ser armazenados em arranjos de números reais. A maneira mais simpels de se guardar as posições dos elementos não nulos e os correspondentes valores neméricos é através de dois arranjos, **lin_v**, e **val_v**. Este tipo de armazenamento é chamado de *Esquema Primitivo de Armazenamento*. Como exemplo deste esquema temos:

Vetor denso

$$\mathbf{v} = \begin{bmatrix} 0. \\ 3. \\ 0. \\ 2. \\ 0. \\ 0. \\ 6. \end{bmatrix}$$

Esquema primitivo

	lin_v	**val_v**
1)	**2**	**3.**
2)	**4**	**2.**
3)	**7**	**6.**

nnz = 3

O esquema primitivo de armazenamento pode ser convertido facilmente para o esquema de listas de adjacências e de listas encadeadas. Para efetuar a conversão, supomos aqui que o número de elementos não nulos do vetor, nnz é conhecido. Esta informação não é imprescindível, porém pode ser obtida com facilidade e ajuda na montagem das listas de adjacências e listas encadeadas.

[1] Neste texto estaremos nos referindo sempre a vetores coluna. Os arranjos e algoritmos podem ser facilmente estendidos para vetores linha.

4.1.1. Listas de Adjacências para vetores Esparsos

A montagem das listas de adjacências para vetores esparsos é trivial uma vez que o arranjo **xadj_v** terá apontadores apenas para o começo e fim do arranjo **adj_v** (onde estão localizados as posições dos elementos não nulos).

Exemplo 8. *Para o vetor* **v** *do exemplo anterior temos então o seguinte grafo e as seguintes listas de adjacências:*

$\mathbf{xadj\_v}(1) = 1$
$\mathbf{xadj\_v}(nnz) = nnz + 1$

xadj	=	[	^{1} **1**	^{2} **4**	]	
lin_v	=	[	^{1} **2**	^{2} **4**	^{3} **7**	]
val_v	=	[	**3.**	**2.**	**6.**	]

Nota-se que, se **lin_v** e **val_v** estão adequadamente ordenados em forma crescente, estes dois arranjos do esquema primitivo de armazenamento são aproveitados nas listas de adjacências. Precisamos portanto montar apenas o arranjo **xadj_v**, o que é trivial.

4.1.2. Listas Encadeadas para Vetores Esparsos

Estas consistem de um apontador para a localização do primeiro elemento não nulo de **v** e um arranjo de apontadores para as localizações dos elementos não nulos seguintes. Estes apontadores são chamados de prim_v e **prox_v**.

Exemplo 9. *As listas encadeadas para o vetor do exemplo acima são:*

	prim_v		**lin_v**	**val_v**	**prox**
		1)	**2**	**3.**	**2**
1)	**1**	2)	**4**	**2.**	**3**
		3)	**7**	**6.**	**0**

Nota-se que **lin_v** e **val_v** fazem parte do esquema primitivo de armazenamento, portanto, a criação da lista encadeada consiste na montagem de **prim_v** e **prox_v**, o que pode ser feito pelo segmento de código abaixo.

$prim\_v = 1$
para $k = 1 : nnz - 1$
    $prox\_v(k) = k + 1$
fim
$prox\_v(nnz) = 0$

Para recuperar os elementos não nulos de um vetor esparso pode ser usado o seguinte algoritmo:

$k = prim\_v$
$loc = 0$
enquanto $k \neq 0$
    $loc = loc + 1$
    $\ell(loc) = lin\_v(k)$
    $v(loc) = val\_v(k)$
    $k = prox\_v(k)$
fim

4.2. Esquema de Armazenamento Compacto para Matrizes Retangulares

Uma matriz $A \in \Re^{m \times n}$ é descrita de forma inequívoca especificando-se:

- seu número de linhas, m, e de colunas, n;
- o número da linha e da coluna de cada elemento não nulo;
- os valores numéricos dos elementos não nulos.

Estas informações podem ser armazenadas no *Esquema Primitivo de Armazenamento para Matrizes* constituído pelos arranjos **lin_A**, **col_A** e **val_A**.

Matriz densa

$$\mathbf{A} = \begin{bmatrix} 3. & 0 & 2. & 0 \\ 0 & 4. & 0 & 0 \\ 0 & 0 & 0 & 7. \\ 5. & 0 & 8. & 9. \\ 0 & 0 & 0 & 6. \end{bmatrix}$$

Esquema primitivo

	lin_A	**col_A**	**val_A**
1)	1	1	3.0
2)	1	3	2.0
3)	2	2	4.0
4)	3	4	7.0
5)	4	1	5.0
6)	4	3	8.0
7)	4	4	9.0
8)	5	4	6.0

4.2.1. Listas de Adjacências para Matrizes Retangulares

Podem ser feitas por linhas ou por colunas (significando que o primeiro número a ser acessado poderá corresponder a uma linha ou a uma coluna). Aquí derivaremos o esquema de adjacências por linhas. O esquema por colunas pode ser construído com pequenas modificações nos algoritmos descritos.

A lista de adjacências por linhas é constituída pelos seguintes arranjos: **xadj_A**, **adj_A** e **v_A**.

		1	2	3	4	5	6	
xadj_A	=	**1**	**3**	**4**	**5**	**8**	**9**	$\leftarrow n^o$ *da linha*

		1	2	3	4	5	6	7	8
adj_A	=	**1**	**3**	**2**	**4**	**1**	**3**	**4**	**4**

v_A	=	**3.0**	**2.0**	**4.0**	**7.0**	**5.0**	**8.0**	**9.0**	**6.0**

O segmento de código abaixo pode ser usado para montar as listas de adjacências por linha para uma matriz retangular $m \times n$. Este algoritmo considera que os elementos primitivos estejam em ordem aleatória e, portanto, **adj_A** e **v_A** devem ser re-ordenadas.

```
xadj_A(1) = 1
para i = 1 : m
    j = 0
    para k = 1 : nnz
        ℓ = lin_A(k)
        se (ℓ = i)
            j = j + 1
            adj_A(j) = col_A(k)
            v_A(j) = val_A(k)
        fim
    fim
    xadj_A(i + 1) = xadj_A(i) + j
fim
```

4.2.2. Listas Encadeadas para Matrizes Retangulares

A estrutura de listas encadeadas pode também ser aplicada a linhas ou a colunas de uma matriz esparsa dependendo se um arranjo de apontadores iniciais é indexado por linhas ou por colunas para esta matriz. Supondo que a lista seja encadeada por linhas, os arranjos correpondentes são: **lprim_A**, **col_A**, **val_A** e **lprox_A**.

Exemplo 10. *Para a matriz A dos exemplos anteriores, as listas encadeadas por linhas podem ser escritas:*

	lprim_A
1)	**2**
2)	**3**
3)	**4**
4)	**7**
5)	**8**

	col_A	val_A	lprox_A
1)	**1**	**3.0**	**0**
2)	**3**	**2.0**	**1**
3)	**2**	**4.0**	**0**
4)	**4**	**7.0**	**0**
5)	**1**	**5.0**	**0**
6)	**3**	**8.0**	**5**
7)	**4**	**9.0**	**6**
8)	**4**	**6.0**	**0**

Um algoritmo de conversão para lista encadeada por linhas é mostrado a seguir. Este algoritmo supõe que os elementos no esquema primitivo podem estar em ordem aleatória o que significa que, ao final, **col_A**, e **val_A** devem ser re-ordenados.

lprim_A $= zeros(1:m)$
$nloc = 0$
$k = 1$
enquanto (**lin_A**$(k) \neq 0$)
  $i =$ **lin_A**(k)
  $nloc = nloc + 1$
  $temp =$ **lprim_A**(i)
  lprim_A$(i) = nloc$
  lprox_A$(nloc) = temp$
  $k = k + 1$
fim

5. Algumas Operações Elementares com Vetores Esparsos

5.1. Adição de um Vetor Esparso a um Vetor Denso

Considere a operação:

$$w = y + \alpha\, x$$

onde y é um vetor denso e x é um vetor esparso de dimensão n, e α é um escalar. O vetor resultante w é obviamente denso. Os únicos elementos de w que diferem dos elementos de y são aqueles correspondentes às posições não-nulas de x. O algoritmo a seguir implementa a operação, considerando que o vetor esparso x

está armazenado na lista encadeada ($prim\_x$, lin_x, val_x, $prox\_x$) .

$w(1:n) = y(1:n)$
$k = prim\_x$
enquanto $(k \neq 0)$
    $i = lin\_x(k)$
    $w(i) = w(i) + \alpha * val\_x(k)$
    $k = prox\_x(k)$
fim

5.2. Adição de Dois Vetores Esparsos

Neste caso, a operação considerada é:

$$z = x + \alpha\, y$$

onde x e y são vetores esparsos de dimensão n e α é um escalar. Neste caso o vetor z será em geral esparso. Há entretanto uma dificuldade adicional com respeito à operação anterior: não é possível saber de antemão a estrutura esparsa do vetor resultante z. Esta estrutura deve ser determinada mesclando-se as estruturas compactas de x e y. Contudo, temos que verificar se x e y têm elementos comuns, já que um elemento não-nulo não pode aparecer mais do que uma vez em uma lista encadeada ou de adjacência. Em casos como esse, o procedimento usual é separar a operação em duas etapas:

- *Adição simbólica,* em que se utiliza apenas os apontadores dos elementos das estruturas esparsas dos operandos x e y para determinar inicialmente a estrutura esparsa de z;
- *Adição numérica,* em que os valores numéricos dos operandos x e y são processados para determinar os valores numéricos de z.

5.2.1. Algoritmo da Adição Simbólica

As etapas básicas do algoritmo são as seguintes:

1. Criar inicialmente um apontador $aptr$, de dimensão n, para armazenar a estrutura de elementos não-nulos de x. Este apontador é definido como:

   $aptr(\ell) = 0 \Rightarrow$ elem. ℓ da linha k é nulo;
   $aptr(\ell) \neq 0 \Rightarrow$ elem. ℓ da linha $k \neq 0$.

2. Copiar a estrutura compacta de x em z;
3. Mesclar as estruturas compactas de x e y. Para isso:

1. usar vetor *aptr* para indicar os elementos não-nulos de x, já copiados em z;
2. copiar na estrutura compacta de z apenas os elementos não-nulos de y que não coincidem com os de x.

Na prática, as etapas 1 e 2 acima são implementadas simultaneamente. O algoritmo detalhado, considerando que listas encadeadas ($prim\_x$, lin_x, $prox\_x$) e ($prim\_y$, lin_y, $prox\_y$) são as estruturas compactas utilizadas para x e y, é apresentado na Fig. 5.1. A estrutura esparsa resultante para z é armazenada na lista encadeada ($prim\_z$, lin_z, $prox\_z$).

Adição Simbólica

```
{Inicialização}
k = prim_x
prim_z = k
aptr(1 : n) = 0
npos_z = 0
{copiar lista encad. de x em z e construir aptr}
enquanto (k ≠ 0)
    npos_z = npos_z + 1
    lin_z(k) = lin_x(k)
    prox_z(k) = prox_x(k)
    aptr(lin_x(k)) = k
    k = prox_x(k)
fim
{Adicionar valores da lista encad. de y à lista encad. de z}
k = first_y
enquanto (k ≠ 0)
    i = lin_y(k)
    se (aptr(i) = 0)
        npos_z = npos_z + 1
        temp = prim_z
        prim_z = npos_z
        lin_z(npos_z) = i
        prox_z(npos_z) = temp
    fim
    k = prox_y(k)
fim
```

Figura 5.1: Algoritmo para adição simbólica de dois vetores esparsos

5.2.2. Algoritmo da Adição Numérica

Consiste das seguintes etapas:

1. Carregar os valores numéricos de x, que estão originalmente no arranjo val_x, em um vetor denso temporário que chamaremos z_denso;

2. Adicionar os valores numéricos de y, val_y, a z_denso;

3. Copiar os valores não-nulos em z_denso no arranjo compacto val_z, cujos apontadores $prim\_z$, lin_z e $prox\_z$ foram determinados na etapa simbólica.

O algoritmo detalhado é mostrado na Fig. 5.2.

Adição Numérica

$z\_denso(1:n) = 0$
$k = prim\_x$
{Copiar val_x em z_denso}
enquanto $(k \neq 0)$
    $z\_denso(lin\_x(k)) = val\_x(k)$
    $k = prox\_x(k)$
fim
{Adicionar elems. em val_y a z_denso}
$k = prim\_y$
enquanto $(k \neq 0)$
    $i = lin\_y(k)$
    $z\_denso(i) = zdenso(i) + \alpha * val\_y(k)$
    $k = prox\_y(k)$
fim
{Copiar elems. não-nulos de z_denso em val_z}
$k = prim\_z$
enquanto $(k \neq 0)$
    $val\_z(k) = z\_denso(lin\_z(k))$
    $k = prox\_z(k)$
fim

Figura 5.2: Algoritmo para adição numérica de dois vetores esparsos

5.3. Multiplicação de uma Matriz Esparsa por um Vetor Denso

Se A é uma matriz esparsa $m \times n$ e x é um vetor $n \times 1$, desejamos implementar o produto:

$$y = Ax$$

em que o vetor y é $m \times 1$ cujos elementos y_i são dados por:

$$y_i = \sum_{j=1}^{n} A_{ij}\, x_j$$

Este produto resultará em um vetor denso, exceto se A contiver linhas nulas. Como supõe-se que x é denso, haverá uma contribuição $A_{ij}\, x_j$ para y_i sempre que $A_{ij} \neq 0$.

Supondo que A está armazenada sob a forma de lista encadeada *por linhas* ($lprim\_A$, col_A, val_A, $lprox\_A$), o algoritmo abaixo forma y por linhas.

```
para i = 1 : m
    y(i) = 0.
    k = lprim_A(i)
    enquanto (k ≠ 0)
        j = col_A(k)
        y(i) = y(i) + val_A(k) * x(j)
        k = lprox_A(k)
    fim
fim
```

6. Solução de Sistemas Triangulares Esparsos

6.1. Fatoração LU

Nesta seção, discutiremos a fatoração LU apenas de *matrizes simétricas ou estruturalmente simétricas*, já que os problemas típicos de sistemas de potência tipicamente envolvem matrizes com estas características.

Estamos interessados na solução do sistema linear:

$$A\, x = b \tag{6.1}$$

onde A é uma matriz $n \times n$ não-singular. Neste caso, A pode ser fatorada como

$$A = L\, U$$

onde L é uma matriz triangular inferior e U é uma matriz triangular superior unitária (isto é, os elementos diagonais u_{ii} de U são iguais à unidade; por outro

lado, o mesmo não se exige dos elementos diagonais de L, l_{ii}). Supondo que a fatoração LU foi realizada, a solução da Eq. (6.1) envolve as seguintes etapas:

$$L\,\bar{x} = b \tag{6.2}$$

e

$$U\,x = \bar{x} \tag{6.3}$$

A etapa dada pela Eq. (6.2) é chamada *substituição direta*, enquanto que a Eq. (6.3) é a *substituição inversa.* Nesta subseção abordaremos o problema da fatoração LU. As etapas de substituição direta e inversa serão abordados nas seções seguintes.

Um algoritmo bem conhecido para implementar a fatoração LU no caso em que A *é uma matriz densa* é dado abaixo. A matriz A com as propriedades descritas acima e a ordem de A constituem os dados de entrada, enquanto que, na saída, os fatores triangulares L e U são escritos sobre A.

para $k = 1 : n$
  para $i = 1 : k-1$
    $A(k,k) = A(k,k) - A(k,i) * A(i,k)$
    para $j = k+1 : n$
      $A(k,j) = A(k,j) - A(k,i) * A(i,j)$
      $A(j,k) = A(j,k) - A(j,i) * A(i,k)$
    fim
  fim
  para $j = k+1 : n$
    $A(k,j) = A(k,k)^{-1} * A(k,j)$
  fim
fim

Examinemos a criação de novos elementos não-nulos (isto é, o *enchimento)* nos fatores triangulares de A em decorrência da fatoração. Observe que estes elementos podem surgir em decorrência de operações do tipo:

$$a'_{kj} = a_{kj} - a_{ki} * a_{ij}$$

Se, antes da operação, a_{kj} era zero mas o produto $a_{ki} * a_{ij}$ é não-nulo, então cria-se um enchimento na posição (k, j) em decorrência da operação. Para o fator triangular inferior, esta propriendade pode ser assim enunciada: se $l_{ki} \neq 0$ e $l_{ij} \neq 0$, então $l_{kj} \neq 0$. Do mesmo modo, para U, se $u_{ki} \neq 0$ e $u_{ij} \neq 0$, então $u_{kj} \neq 0$. A Fig. 6.1 ilustra a criação de enchimento nos fatores triangulares.

6.2. Fatoração LU Esparsa

No caso de matrizes estruturalmente simétricas, os fatores triangulares L e U terão elementos não-nulos:

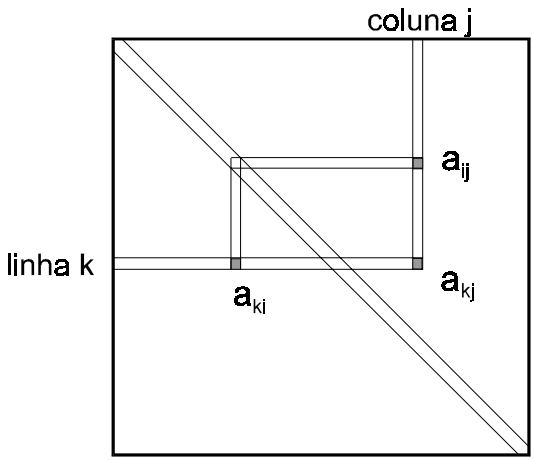


Figura 6.1: Ilustração do enchimento na fatoração LU.

- – Nas posições em que A tiver elementos não-nulos, e
  – Se $a_{kj} = 0$ mas $a_{ki} * a_{ij} \neq 0$, cria-se enchimento nas posições:
    * (k, j) de U e
    * (j, k) de L.

Como antes de se calcular os valores numéricos dos elementos de L e U é necessário saber quais elementos destas matrizes são não-nulos, a fatoração envolve duas etapas:

- – *Fatoração simbólica*, através da qual se determina em que posições de L e U ocorrem elmentos não-nulos;
  – *Fatoração numérica:* cálculo dos valores numéricos dos elementos não-nulos de L e U, a partir da alocação de memória feita no estágio numérico.

6.2.1. Fatoração Simbólica

Entrada:

- – Listas encadeadas *por linha e por coluna* da matriz estruturalmente simétrica, **A**;
  – Ordem de **A**, n;
  – Número de elementos $\neq 0$ em **A**, nnz.

Saída:

- – Listas encadeadas de **A** são sobrescritas pelas listas de **L** e **U**;

– nnz conterá o número de elementos $\neq 0$ nos fatores triangulares.

Equações básicas:

$$A_{k\,j} = A_{k\,j} - A_{k\,i} * A_{i\,j}$$
$$A_{j\,k} = A_{j\,k} - A_{j\,i} * A_{i\,k}$$

O algoritmo está descrito na Fig. 6.2. Observe que, no algoritmo, o índice j é suposto variável sobre os elementos não-nulos da linha i.

Fatoração Simbólica Esparsa

para $k = 2 : n$
  construir apontador *aptr* para linha k :
    $aptr(\ell) = 0 \Rightarrow$ elem. ℓ da linha k é nulo;
    $aptr(\ell) \neq 0 \Rightarrow$ elem. ℓ da linha $k \neq 0$;
  {Percorrer os elementos $\neq 0$ da linha k}
  para cada coluna i da linha k onde $\exists\ A_{ki} \neq 0$:
    {variar j sobre a linha i}
    Seja j o próximo elemento $\neq 0$ na linha i (A_{ij})
    se $(aptr(j) = 0)$
      {enchimento nas posições kj e jk}
      inserir novo elemento $\neq 0$ na col. j da linha k;
      inserir novo elemento $\neq 0$ na linha k da col. j.
      atualizar *aptr*
      incrementar nnz
    fim
  fim
fim

Figura 6.2: Algoritmo para fatoração simbólica esparsa.

6.2.2. Fatoração Numérica

Entrada:

- – Listas encadeadas *por linha e por coluna* para os fatores triangulares determinadas pela fatoração simbólica;
  – Ordem de **A**, n;

Saída:

- – Arranjos com os valores numéricos de **L**, **U** indexados pelas listas encadeadas atualizadas na fatoração simbólica;
  – Arranjo com os elementos diagonais de **A** é sobrescrito pelos elementos diagonais de **L**.

O algoritmo de fatoração numérica é esboçado na Fig. 6.3.

Fatoração Numérica Esparsa

para $k = 1 : n$
  construir apontador *aptr* para linha k a partir das listas encadeadas de **L** e **U** obtidas na fatoração simbólica:
    $aptr(\ell) = 0 \Rightarrow$ elem. ℓ da linha k é nulo;
    $aptr(\ell) \neq 0 \Rightarrow$ elem. ℓ da linha $k \neq 0$;
  {Percorrer os elementos $\neq 0$ da linha k}
  enquanto $\exists$ um elem. $i \neq 0$ na linha k:
    $\beta = l_{ki}$
    $\alpha = u_{ik}$
    $diag(k) = diag(k) - \beta * \alpha$
    enquanto $\exists$ um elem. $\neq 0$ j na linha i:
      $u_{k\,j} = u_{k\,j} - l_{k\,i} * u_{i\,j}$
      $l_{j\,k} = l_{j\,k} - l_{j\,i} * u_{i\,k}$
    fim
  fim
  enquanto $\exists$ um sucessor j de i na lista de elementos não-nulos da col. k
    $u_{k\,j} = diag(k)^{-1} * u_{k\,j}$
  fim
fim

Figura 6.3: Algoritmo para fatoração numérica esparsa.

7. Substituições Direta e Inversa

À fatoração LU da matriz de coeficientes de um sistema linear esparso, seguem-se os estágios de substituição direta e inversa, conforme dado pelas Eqs. 6.2 e 6.3, que fornecem a solução desejada para o problema linear. Tanto no caso denso quanto no caso esparso, diferentes algoritmos podem ser utilizados para implementar os estágios de substituição direta e inversa, dependendo se o acesso aos elementos dos fatores L e U se faz por linhas ou por colunas.

No caso da substituição direta, apresentaremos algoritmos que acessam o fator triangular inferior L *por colunas*. O algoritmo para o caso de matrizes densas é bem conhecido, e está apresentado na coluna da esquerda do quadro abaixo. Vê-se que o laço interno do algoritmo de fato acessa os elementos fora da diagonal de L por colunas, enquanto o laço externo acessa os elementos diagonais de L. A solução desejada sobrescreve os elementos do vetor do lado direito b. A Figura 7.1 mostra como os elementos de L são acessados durante o processo de solução.

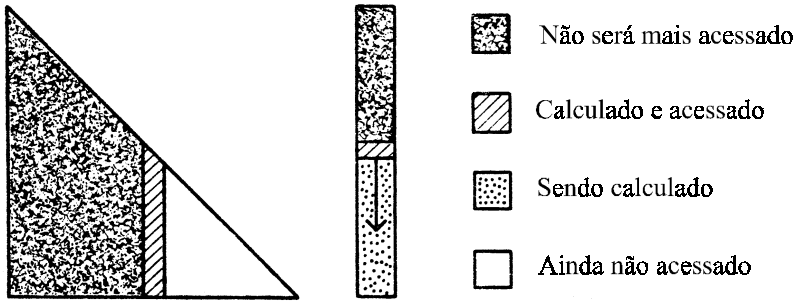


Figura 7.1: Acesso às colunas de L durante substituição direta.

para $i = 1 : n-1$
  $b(i) = L(i,i)^{-1} * b(i)$
  para $j = i+1 : n$
    $b(j) = b(j) - L(j,i) * b(i)$
  fim
fim
$b(n) = L(n,n)^{-1} * b(n)$

para $i = 1 : n-1$
  $b(i) = diag\_L(i)^{-1} * b(i)$
  $jp = cprim\_L(i)$
  enquanto $jp \neq 0$
    $j = lin\_L(jp)$
    $b(j) = b(j) - val\_L(jp) * b(i)$
    $jp = cprox\_L(jp)$
  fim
fim
$b(n) = diag\_L(n)^{-1} * b(n)$

A segunda coluna do quadro acima apresenta o algoritmo de substituição direta *esparsa*, considerando que os elementos fora da diagonal da matriz triangular esparsa L está armazenada *por colunas* através dos arranjos ($cprim\_L$, lin_L, val_L, $cprox\_L$), enquanto que os elementos diagonais de L estão contidos no arranjo $diag\_L$. Comparando as versões densa e esparsa do algoritmo, verificamos que, aparte as formas distintas de armazenamento de L, a única real diferença é a substituição do laço interno *"para"* do algoritmo denso por um laço *"enquanto"* que percorre a lista encadeada por colunas de L no algoritmo esparso.

As mesmas conclusões feitas acima para a substituição direta se aplicam, com as adaptações devidas, à substituição inversa. Consideraremos a seguir o caso de algoritmos que acessam o fator triangular superior U *por linhas, supondo que* U é

unitária (isto é, $u_{ii} = 1$). O lado direito do quadro abaixo mostra a versão densa da substituição inversa. Observe que os elementos de U são de fato acessados por linhas no laço interno do algoritmo. A versão esparsa é apresentada no lado direito do quadro, sendo os elementos fora da diagonal de U armazenados de forma compacta nos arranjos ($lprim\_U$, col_U, val_U, $lprox\_U$). Novamente, observe que a única diferença entre os dois algoritmos é a substituição do laço interno *"para"* do algoritmo denso por um laço *"enquanto"* que percorre a lista encadeada por linhas de U no algoritmo esparso.

para $i = n - 1 : -1 : 1$
  para $j = i + 1 : n$
    $b(i) = b(i) - U(i, j) * b(j)$
  fim
fim

para $i = n - 1 : -1 : 1$
  $jp = lprim\_U(i)$
  enquanto $jp \neq 0$
    $j = col\_U(jp)$
    $b(i) = b(i) - val\_U(jp) * b(j)$
    $jp = lprox\_U(jp)$
  fim
fim

8. Ordenação para Preservação da Esparsidade na Fatoração LU

8.1. Ordenação de Matrizes

Nesta seção, consideraremos apenas sistemas lineares cujas matrizes são estruturalmente simétricas.

Definimos uma *ordenação* de uma matriz estruturalmente simétrica A como uma permutação de suas linhas *e* colunas. A estrutura simétrica das matrizes consideradas implica que estas permutações têm que ser realizadas em pares linha-coluna, isto é, se a linha k for permutada com a linha j, o mesmo acontecerá com as colunas k e j.

Quando uma matriz é reordenada, verifica-se que o enchimento produzido durante a fatoração LU pode variar substancialmente. Faz sentido portanto buscar ordenações que minimizem o enchimento ou, se isto não for possível, o reduzam significativamente. Reordenações de matrizes estruturalmente simétricas são melhor ilustradas através dos grafos de adjacência introduzidos no início deste capítulo.

Exemplo 11. *Seja o sistema de potência representado na Fig. 8.1. A estrutura da matriz de admitância do sistema e seu grafo de adjacência associado são mostrados na Fig. 8.2, para duas ordenações diferentes:*

- A ordenação natural;

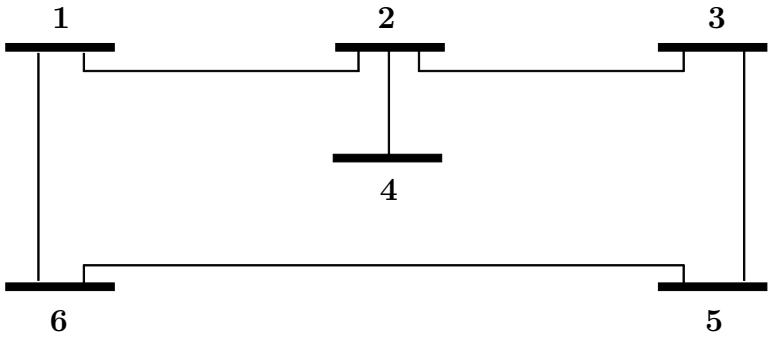


Figura 8.1: Sistema de Potência para exemplificar ordenação

- A ordenação $\{4, 1, 3, 5, 6, 2\}$ (isto é, o primeiro vértice considerado é o vértice originalmente numerado como 4, etc.).

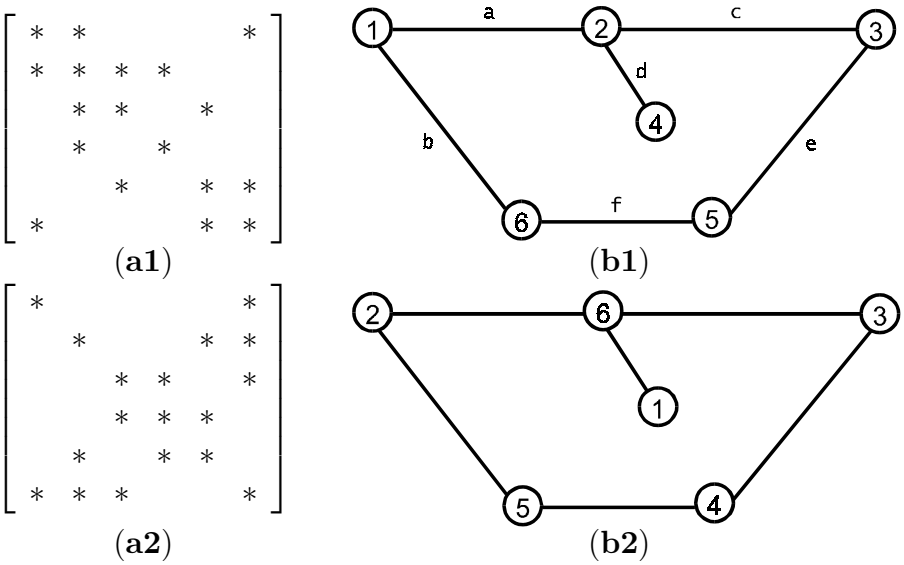


Figura 8.2: Estrutura da matriz A e grafo associado para: ($a1$, $b1$) ordenação natural e ($a2$, $b2$) ordenação $\{4, 1, 3, 5, 6, 2\}$.

8.2. Interpretação do Enchimento Usando Grafos Reduzidos

O processo de fatoração LU é tal que, quando a linha 1 e a coluna 1 da matriz $A = A(1 : n, 1 : n)$ são processadas, elas modificarão o restante da matriz A, isto é, $A(2 : n, 2 : n)$. Depois desta etapa, a linha 1 e a coluna 1 não serão mais acessadas. Em geral, no estágio k são modificados os elementos da submatriz $A(k + 1 : n, k + 1 : n)$. Depois deste ponto, as primeiras k linhas e k colunas não serão mais acessadas durante o processo de fatoração.

Lembremos que um novo elemento diferente de zero (isto é, um *enchimento*) na posição (k, j) será criado se $A(k, j)$ for inicialmente nulo e se existir pelo menos um par $A(k, i)$ e $A(i, j)$, ambos não nulos. Pode-se então estabelecer uma correspondência entre a operação que cria o enchimento e alterações no grafo de adjacência da matriz A. Assim, o processamento da primeira linha e primeira coluna de A provoca as seguintes mudanças no grafo de adjacência de A :

- O vértice x_1 e todas as arestas nele incidentes são eliminados do grafo;
- Novas arestas são criadas no grafo reduzido, tal que todos os vértices originalmente adjacentes a x_1 tornam-se mutuamente adjacentes.

Deste modo, todo o processo de fatoração LU pode ser interpretado como uma seqüência de *grafos de adjacência reduzidos.* A Fig. 8.3 ilustra a geração de grafos reduzidos para a matriz de adjacência do sistema de potência da Fig. 8.1. As matrizes H_i representam as matrizes reduzidas $A(i+1:n, i+1:n)$, e as arestas mais grossas nos grafos reduzidos representam as arestas criadas durante o processo de fatoração. A cada uma delas corresponde um novo enchimento na matriz A. Por exemplo, a eliminação do vértice x_2 no grafo G_1 gera três novos enchimentos, que correspondem às arestas $\{x_3, x_4\}$, $\{x_4, x_6\}$ e $\{x_3, x_6\}$.

O conjunto de arestas adicionadas aos grafos reduzidos corresponde portanto ao conjunto de enchimentos criados durante a fatoração. Para o exemplo da Fig.8.3, a estrutura da matriz $L + L^T$ e o grafo correspondente, considerando todos os enchimentos produzidos durante a fatoração são mostrados na Fig. 8.4.

Esta visualização do enchimento em termos dos grafos de adjacência é muito útil na interpretação dos efeitos da ordenação durante o processo de fatoração da matriz A. Por exemplo, a segunda ordenação considerada no Exemplo 11 provocaria apenas dois enchimentos, um número bastante inferior aos 5 enchimentos produzidos se a ordenação natural for a utilizada (verifique!).

8.3. Algoritmos de Ordenação

Como observado na seção anterior, o enchimento provocado em uma matriz esparsa estruturalmente simétrica A durante a fatoração LU pode variar significativamente com a reordenação de suas linhas e colunas. A interpretação do processo de produção de enchimento utilizando os grafos reduzidos constitui-se em uma valiosa ferramenta para avaliação dos efeitos da reordenação das linhas e colunas de A sobre a produção de novos elementos não-nulos.

Constatados os efeitos da reordenação sobre a esparsidade dos fatores triangulares, é natural que seja levantada a seguinte questão: existe uma ordenação ótima que minimize o enchimento em L e U? Observe que este é um problema combinatório de grande dimensão: há n maneiras de se escolher a linha/coluna 1, $n-1$ maneiras de se escolher a linha/coluna 2, e assim por diante. De fato

nenhum método *ótimo* com este objetivo foi desenvolvido até hoje, e há evidências de que um método ótimo geral, aplicável a matrizes esparsas de estrutura genérica, não existe. Temos portanto que recorrer a algoritmos heurísticos para gerar ordenações que produzam um número pequeno aceitável, embora não necessariamente ótimo, de novos elementos não-nulos em L e U.

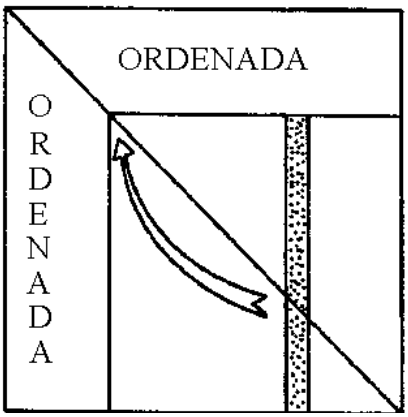
Abordaremos aqui três métodos heurísticos de ordenação. Ambos são baseados no conceito de *grau* dos vértices do grafo de adjacência. Como vimos na seção anterior, os vértices de menor grau tendem a criar um número menor de enchimentos. De fato, no caso da eliminação de um nó de grau 1, nenhum enchimento é criado. Os três algoritmos heurísticos são descritos na seqüência.

8.3.1. Algoritmo de Ordenação I

Também chamado *Esquema "Tinney-I"*, baseia-se na ordenação *a priori* (e portanto estática) dos vértices do grafo de adjacência em ordem crescente de grau. Sua implementação é a mais rápida, e costuma produzir resultados bastante razoáveis em aplicações a sistemas de potência. Sua implementação requer o cálculo dos graus de cada vértice do grafo de adjacência.

8.3.2. Algoritmo de Ordenação II

É conhecido como *Método do Mínimo Grau* ou *Esquema Tinney-II*. Baseia-se na evolução dos graus nodais nos grafos reduzidos que resultam da eliminação de vértices, conforme visto na seção anterior. Como a eliminação de um nó em um grafo reduzido pode provocar alteração no grau dos vértices adjacentes, o método tem características dinâmicas. A figura abaixo ilustra o método.



Suponha que as linhas/colunas 1 a $i-1$ (ou, equivalentemente, os nós do grafo de adjacência $\{x_1, \ldots, x_{i-1}\}$) tenham sido já ordenadas. O número de elementos não-nulos nestas linhas/colunas é fixo e não será mais alterado, já que elas não serão mais acessadas. Para reduzir o enchimento na coluna i é evidente que, na submatriz que ainda não foi fatorada, a coluna com o menor número de

elementos não-nulos deve ser deslocada para a posição i. Portanto, o esquema pode ser considerado como um método que reduz o enchimento através de uma minimização local no número de elementos da coluna i da matriz fatorada.

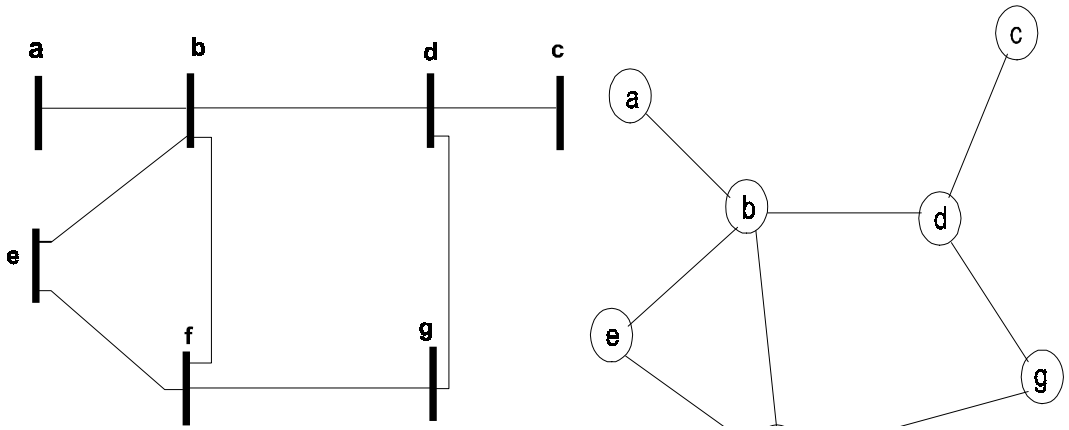
O algoritmo básico é melhor descrito em termos do grafo de adjacência e dos grafos reduzidos, conforme mostrado abaixo:

Algoritmo do Mínimo Grau

{Inicialização}
$i = 1$
enquanto $i \leq n$
    {Seleção do mínimo grau}
    Escolher um vértice x_i de mínimo grau no grafo reduzido G_{i-1}.
    {Redução do Grafo}
    Formar novo grafo reduzido G_i eliminando o vértice x_i de G_{i-1}.
    $i = i + 1$
fim

Exemplo 12. *Considere o sistema de potência e o respectivo grafo de adjacência da figura abaixo. A Fig. 8.5 apresenta as etapas de determinação da ordenação de mínimo grau. A ordenação encontrada é:*

$$\{a, c, d, e, b, f, g\}$$



8.3.3. Algoritmo de Ordenação III

Este esquema de ordenação é mais elaborado, pois implica na análise prévia da criação do enchimento para cada linha/coluna ainda não processada. Como no caso do Esquema II, o método é mais facilmente apresentado utilizando-se o grafo

de adjacência e o grafos reduzidos dele derivados:

Algoritmo de Ordenação III

{Inicialização}
$i = 1$
enquanto $i \leq n$
    {Seleção do mínimo grau}
    Escolher o vértice x_i no grafo reduzido G_{i-1} que, quando eliminado,
    produza o número mínimo de novas arestas no novo
    grafo reduzido G_i.
    {Redução do Grafo}
    Formar novo grafo reduzido G_i eliminando o vértice x_i de G_{i-1}.
    $i = i + 1$
fim

Verifica-se portanto a necessidade de simularmos a eliminação de todos os possíveis nós candidatos antes de tomar a decisão de qual nó deverá ser de fato o próximo a ser eliminado. Um possível atalho neste procedimento ocorre quando se verifica que um nó candidato não provoca nenhum enchimento: neste caso, este pode ser imediatamente selecionado, sem a necessidade de examinarmos todos os nós restantes.

As ordenações determinadas pelo Algoritmo III tendem a produzir um número menor de enchimentos que as fornecidas pelos demais algoritmos. Apesar disso, este algoritmo requer um tempo de processamento muito superior ao do Algoritmo II, sendo porisso geralmente preterido em relação àquele.

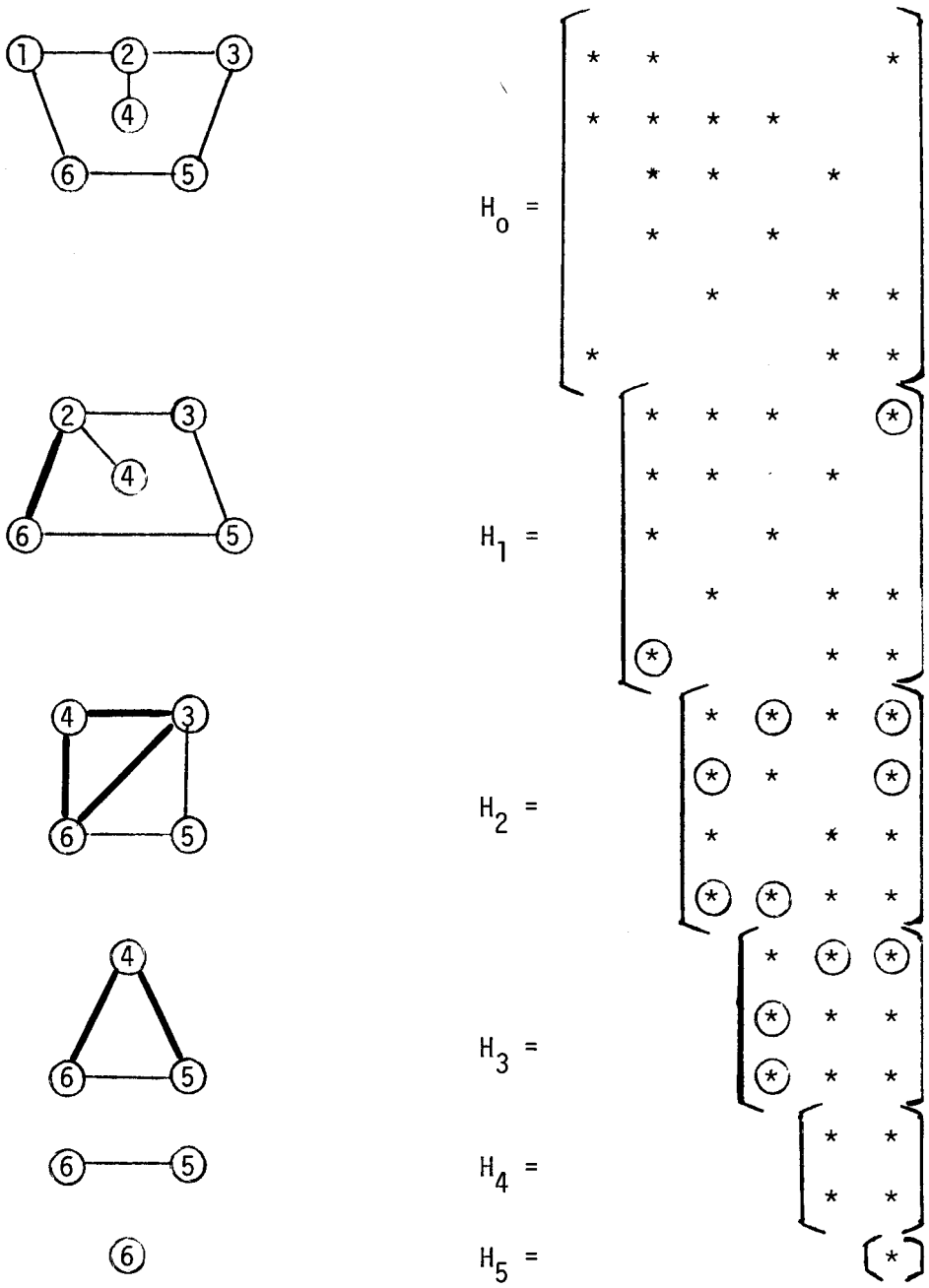


Figura 8.3: Interpretação do enchimento usando seqüência de grafos reduzidos.

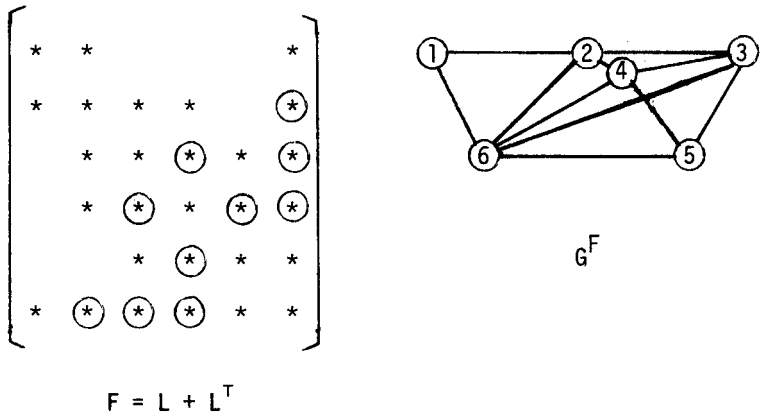


Figura 8.4: Enchimento total e grafo correspondente para o exemplo.

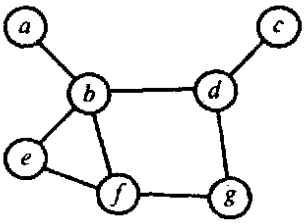
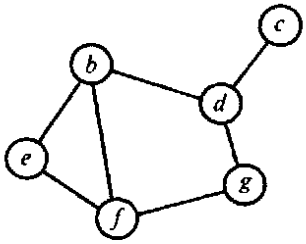
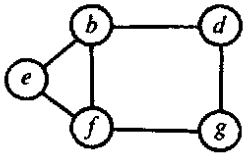
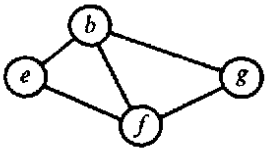
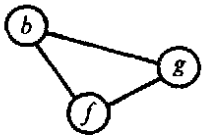
i	Grafo Reduzido $G(i\text{-}1)$	Nó Escolhido	Mínimo Grau
1		a	1
2		c	1
3		d	2
4		e	2
5		b	2
6		f	1
7		g	0

Figura 8.5: Exemplo de aplicação do algoritmo de mínimo grau.