Sparse Matrix Methods
Chapter 7 lecture notes
Ordering methods

Tim Davis

2011

Ordering methods

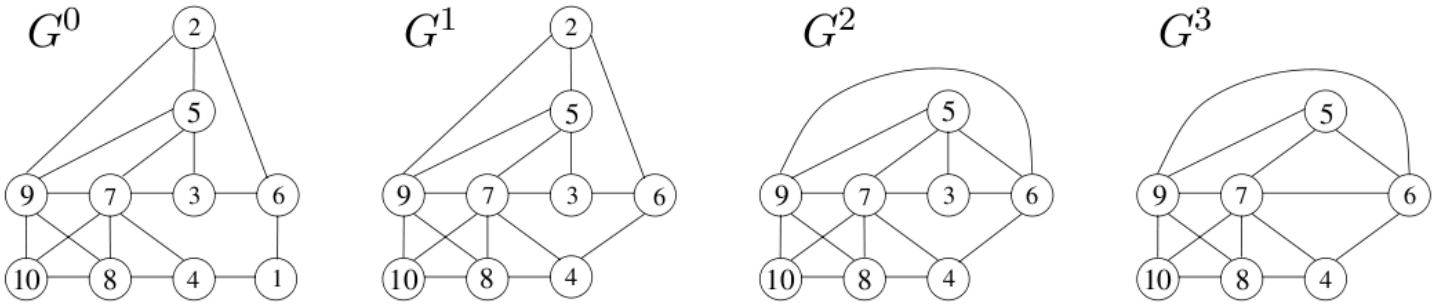
- find P to minimized $|L|$, where $PAP^T = LL^T$
- NP-hard problem
- minimum degree methods
- nested dissection methods

Minimum degree

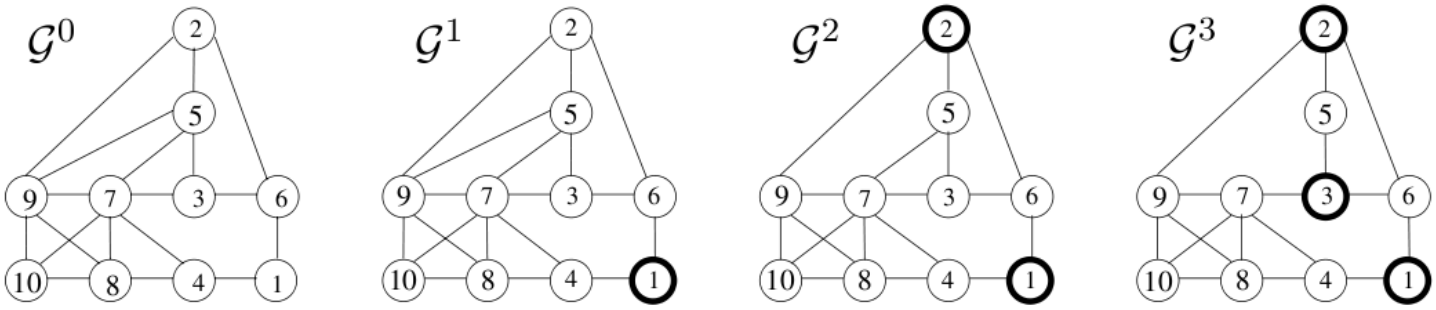
```
for k = 1:n
    L (k,k) = sqrt (A (k,k)) ;
    L (k+1:n,k) = A (k+1:n,k) / L (k,k) ;
    A (k+1:n,k+1:n) = A (k+1:n,k+1:n) - L (k+1:n,k) * L (k+1:n,k)' ;
end
```

- $G^{[k]}$: the graph after nodes 1 to k eliminated
- $\mathcal{A}_i$: the adjacency of node i in $G^{[k]}$
- $\mathcal{L}_j$: the nonzero pattern of `L(:,j)`
- $\mathcal{G}^{[k]}$: the *quotient* graph: a collection of cliques
- $\mathcal{E}_k$: clique created when node k eliminated

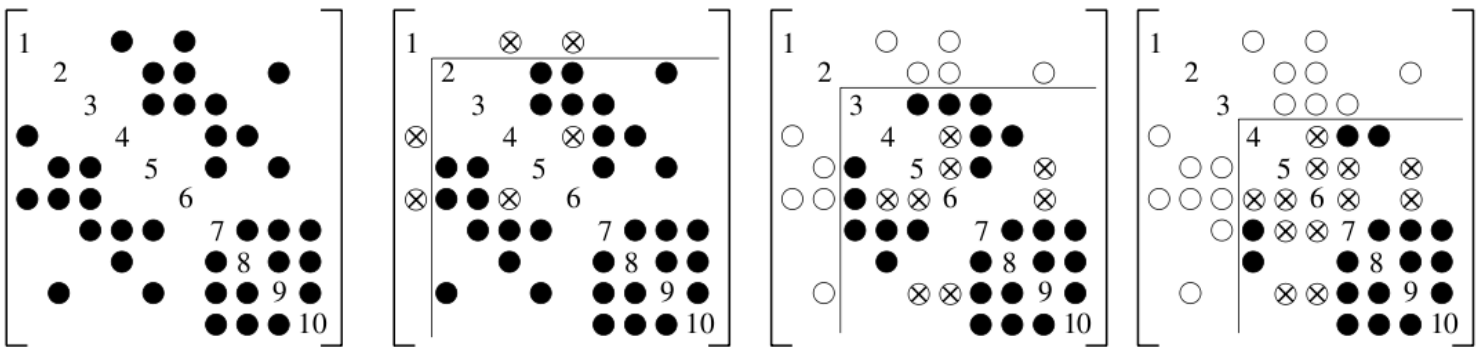
Minimum degree



(a) Elimination graph



(b) Quotient graph



(c) Factors and active submatrix

- In the quotient graph, the nonzero pattern of row/column i is:

$$\left(\mathcal{A}_i \cup \bigcup_{e \in \mathcal{E}_i} \mathcal{L}_e \right) \setminus \{i\}$$

- key ideas:
  - element absorption
  - indistinguishable nodes
  - mass elimination
  - approximate degree

Minimum degree

- exact degree:

$$d_i = \left| \left(\mathcal{A}_i \cup \bigcup_{e \in \mathcal{E}_i} \mathcal{L}_e \right) \setminus \{i\} \right|$$

- cheap bound:

$$d_i < |\mathcal{A}_i| + \sum_{e \in \mathcal{E}_i} |\mathcal{L}_e|$$

- better bound:

$$d_i < |\mathcal{A}_i| + |\mathcal{L}_k \setminus \{i\}| + \sum_{e \in \mathcal{E}_i \setminus \{k\}} |\mathcal{L}_e \setminus \mathcal{L}_k|$$

Minimum degree

function *scan1*

assume $w(e) < 0$ for all $e = 1, \ldots, n$

for each node $i \in \mathcal{L}_k$ **do**

for each element $e \in \mathcal{E}_i$ **do**

if $(w(e) < 0)$ **then** $w(e) = |\mathcal{L}_e|$

$w(e) = w(e) - |i|$

Maximum matching

- permutation PA to find a zero-free diagonal
- model as a bipartite graph
- algorithm based on finding *alternating augmenting paths*

Maximum matching

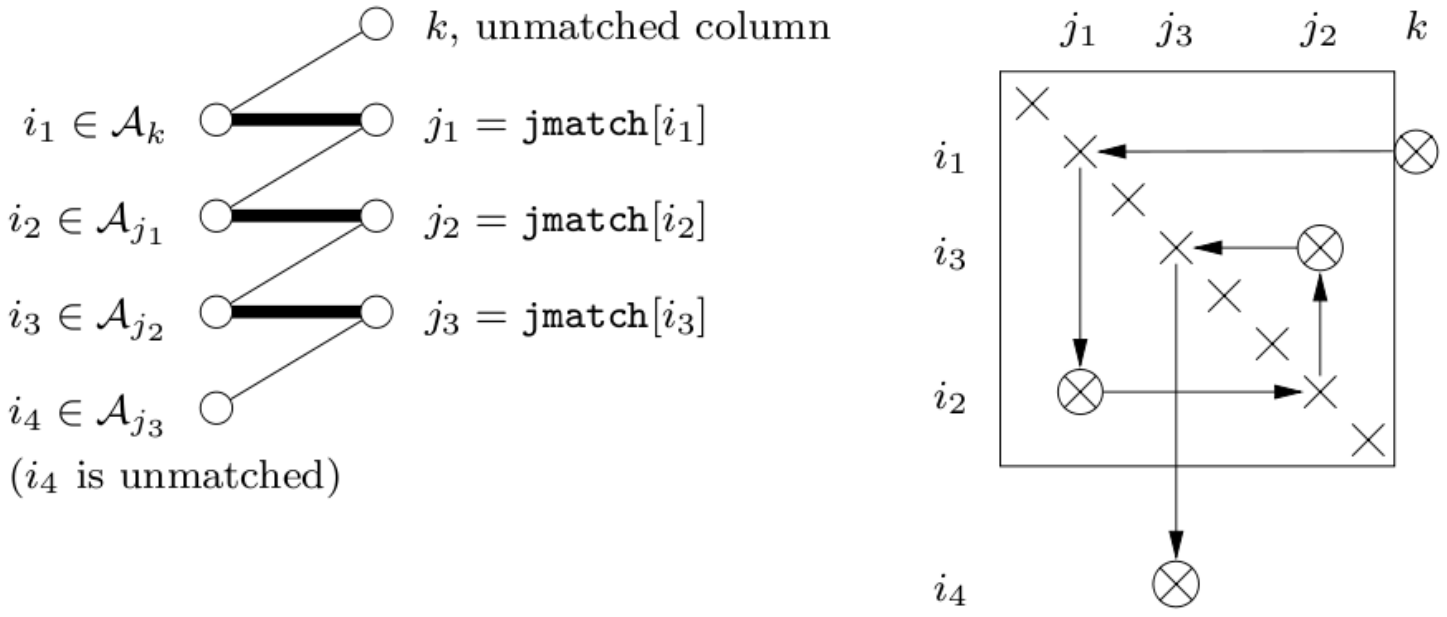


Figure 7.2. *An alternating augmenting path*

Maximum matching

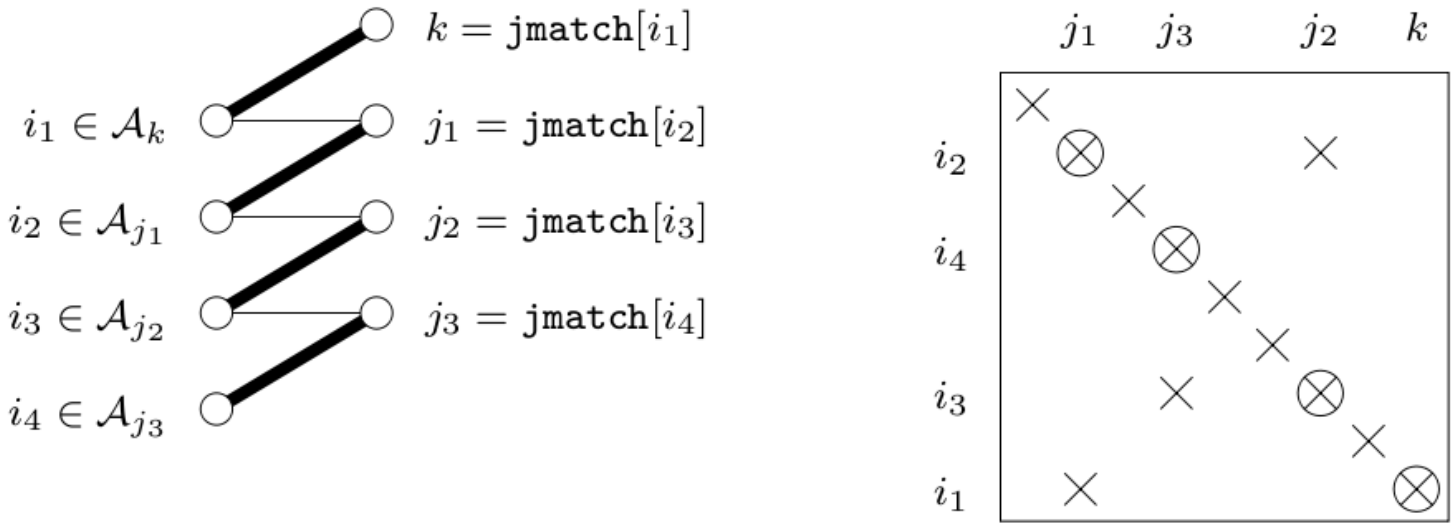


Figure 7.3. *Matching extended via an alternating augmenting path*

```
int *maxtrans (cs *A)   /* returns jmatch [0..m-1] */
{
    int i, j, k, n, m, *Ap, *jmatch, *w, *cheap ;
    if (!A) return (NULL) ;                          /* check inputs */
    n = A->n ; m = A->m ; Ap = A->p ;
    jmatch = cs_malloc (m, sizeof (int)) ;           /* allocate result */
    w = cs_malloc (2*n, sizeof (int)) ;              /* allocate workspace */
    if (!w || !jmatch) return (cs_idone (jmatch, NULL, w, 0)) ;
    cheap = w + n ;
    for (j = 0 ; j < n ; j++) cheap [j] = Ap [j] ;   /* for cheap assignment */
    for (j = 0 ; j < n ; j++) w [j] = -1 ;           /* all columns unflagged */
    for (i = 0 ; i < m ; i++) jmatch [i] = -1 ;      /* no rows matched yet */
    for (k = 0 ; k < n ; k++) augment (k, A, jmatch, cheap, w, k) ;
    return (cs_idone (jmatch, NULL, w, 1)) ;
}
```

```
int augment (int k, cs *A, int *jmatch, int *cheap, int *w, int j)
{
    int found = 0, p, i = -1, *Ap = A->p, *Ai = A->i ;
    /* --- Start depth-first-search at node j ------------------------------ */
    w [j] = k ;                                 /* mark j as visited for kth path */
    for (p = cheap [j] ; p < Ap [j+1] && !found ; p++)
    {
        i = Ai [p] ;                            /* try a cheap assignment (i,j) */
        found = (jmatch [i] == -1) ;
    }
    cheap [j] = p ;                             /* start here next time for j */
    /* --- Depth-first-search of neighbors of j ---------------------------- */
    for (p = Ap [j] ; p < Ap [j+1] && !found ; p++)
    {
        i = Ai [p] ;                            /* consider row i */
        if (w [jmatch [i]] == k) continue ; /* skip col jmatch [i] if marked */
        found = augment (k, A, jmatch, cheap, w, jmatch [i]) ;
    }
    if (found) jmatch [i] = j ;                 /* augment jmatch if path found */
    return (found) ;
}
```