CS 267 Applications of Parallel Computers

Lecture 15:

Graph Partitioning - II

James Demmel

http://www.cs.berkeley.edu/~demmel/cs267_Spr99

Outline of Graph Partitioning Lectures

- **Review of last lecture**
- **Partitioning without Nodal Coordinates - continued**
  - Kernighan/Lin
  - Spectral Partitioning
- **Multilevel Acceleration**
  - BIG IDEA, will appear often in course
- **Available Software**
  - good sequential and parallel software availble
- **Comparison of Methods**
- **Applications**

Review Definition of Graph Partitioning

° **Given a graph $G = (N, E, W_N, W_E)$**

- N = nodes (or vertices), E = edges
- W_N = node weights, W_E = edge weights

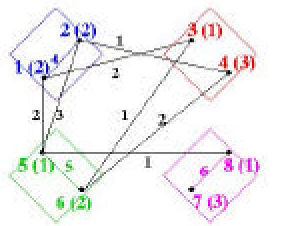
° Ex: N = {tasks}, W_N = {task costs}, edge (j,k) in E means task j sends W_E(j,k) words to task k

° **Choose a partition $N = N_1 \cup N_2 \cup \ldots \cup N_P$ such that**

- The sum of the node weights in each N_j is "about the same"
- The sum of all edge weights of edges connecting all different pairs N_j and N_k is minimized

° Ex: balance the work load, while minimizing communication

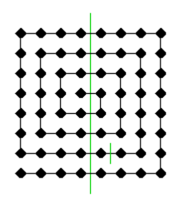
° **Special case of $N = N_1 \cup N_2$: Graph Bisection**



Review of last lecture

° **Partitioning with nodal coordinates**

- Rely on graphs having nodes connected (mostly) to "nearest neighbors" in space
- Common when graph arises from physical model
- Algorithm very efficient, does not depend on edges!
- Can be used as good starting guess for subsequent partitioners, which do examine edges
- Can do poorly if graph less connected:



° **Partitioning without nodal coordinates**

- Depends on edges
- No assumptions about where "nearest neighbors" are
- Began with Breadth First Search (BFS)

Partitioning without nodal coordinates - Kernighan/Lin

° Take a initial partition and iteratively improve it

- Kernighan/Lin (1970), cost = O(|N|^3) but easy to understand
- Fiduccia/Mattheyses (1982), cost = O(|E|), much better, but more complicated

° Let G = (N,E,W_E) be partitioned as N = A U B, where |A| = |B|

° T = cost(A,B) = S {W(e) where e connects nodes in A and B}

° Find subsets X of A and Y of B with |X| = |Y| so that swapping X and Y decreases cost:

  - newA = A - X U Y   and   newB = B - Y U X
  - newT = cost(newA , newB) < cost(A,B)
  - Keep choosing X and Y until cost no longer decreases

° Need to compute newT efficiently for many possible X and Y, choose smallest

Kernighan/Lin - Preliminary Definitions

° T = cost(A, B),   newT = cost(newA, newB)

° Need an efficient formula for newT; will use

- E(a) = external cost of a in A = S {W(a,b) for b in B}
- I(a) = internal cost of a in A = S {W(a,a') for other a' in A}
- D(a) = cost of a in A = E(a) - I(a)
  - Moving a from A to B would decrease T by D(a)
- E(b), I(b) and D(b) defined analogously for b in B

° Consider swapping X = {a} and Y = {b}

- newA = A - {a} U {b},   newB = B - {b} U {a}

° newT = T - (D(a) + D(b) - 2*w(a,b)) = T - gain(a,b)

- gain(a,b) measures improvement gotten by swapping a and b

° Update formulas, after a and b are swapped

- newD(a') = D(a') + 2*w(a',a) - 2*w(a',b)   for a' in A, a' != a
- newD(b') = D(b') + 2*w(b',b) - 2*w(b',a)   for b' in B, b' != b

Kernighan/Lin Algorithm

```
Compute  T = cost(A,B) for initial A, B                    … cost = O(|N|^2)
Repeat
     … One pass greedily computes |N|/2 possible X,Y to swap, picks best
     Compute costs D(n) for all n in N                          … cost = O(|N|^2)
     Unmark all nodes in N                                      … cost = O(|N|)
     While there are unmarked nodes                             … |N|/2 iterations
         Find an unmarked pair (a,b) maximizing gain(a,b)         … cost = O(|N|^2)
         Mark a and b (but do not swap them)                      … cost = O(1)
         Update D(n) for all unmarked n,
               as though a and b had been swapped               … cost = O(|N|)
     Endwhile
         … At this point we have computed a sequence of pairs
         …  (a1,b1), … , (ak,bk)   and gains gain(1),…., gain(k)
         … where k = |N|/2, numbered in the order in which we marked them
     Pick m maximizing Gain = Σ_{k=1 to m} gain(k)                … cost = O(|N|)
         … Gain is reduction in cost from swapping (a1,b1) through (am,bm)
     If Gain > 0 then   … it is worth swapping
         Update newA = A - { a1,…,am } U { b1,…,bm }            … cost = O(|N|)
         Update newB = B - { b1,…,bm } U { a1,…,am }            … cost = O(|N|)
         Update T = T - Gain                                    … cost = O(1)
     endif
 Until Gain <= 0
```


Comments on Kernighan/Lin Algorithm

- Most expensive line show in red
- Some gain(k) may be negative, but if later gains are large, then final Gain may be positive
    - can escape “local minima” where switching no pair helps
- How many times do we Repeat?
    - K/L tested on very small graphs (|N|<=360) and got convergence after 2-4 sweeps
    - For random graphs (of theoretical interest) the probability of convergence in one step appears to drop like $2^{-|N|/30}$

Partitioning without nodal coordinates - Spectral Bisection

° Based on theory of Fiedler (1970s), popularized by Pothen, Simon, Liou (1990)

° Motivation, by analogy to a vibrating string

° Basic definitions

° Vibrating string, revisited

° Motivation, by using a continuous approximation to a discrete optimization problem

° Implementation via the Lanczos Algorithm

- To optimize sparse-matrix-vector multiply, we graph partition
- To graph partition, we find an eigenvector of a matrix associated with the graph
- To find an eigenvector, we do sparse-matrix vector multiply
- No free lunch ...

Motivation for Spectral Bisection: Vibrating String

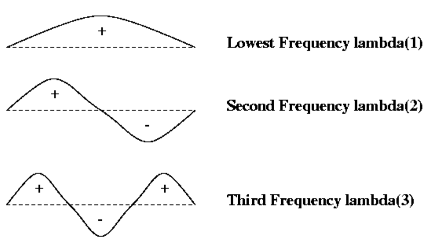
° Think of G = 1D mesh as masses (nodes) connected by springs (edges), i.e. a string that can vibrate

° Vibrating string has modes of vibration, or harmonics

° Label nodes by whether mode - or + to partition into N- and N+

° Same idea for other graphs (eg planar graph ~ trampoline)

Modes of a Vibrating String



Basic Definitions

- *Definition*: The **incidence matrix In(G)** of a graph G(N,E) is an |N| by |E| matrix, with one row for each node and one column for each edge. If edge e=(i,j) then column e of In(G) is zero except for the i-th and j-th entries, which are +1 and -1, respectively.

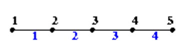
- Slightly ambiguous definition because multiplying column e of In(G) by -1 still satisfies the definition, but this won't matter...

- *Definition*: The **Laplacian matrix L(G)** of a graph G(N,E) is an |N| by |N| symmetric matrix, with one row and column for each node. It is defined by
  - L(G) (i,i) = degree of node I (number of incident edges)
  - L(G) (i,j) = -1 if i != j and there is an edge (i,j)
  - L(G) (i,j) = 0 otherwise

Example of In(G) and L(G) for 1D and 2D meshes

Incidence and Laplacian Matrices

Graph G

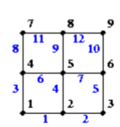


Incidence Matrix In(G)

	1	2	3	4
1	-1			
2	1	-1		
3		1	-1	
4			1	-1
5				1

Laplacian Matrix L(G)

	1	2	3	4	5
1	1	-1			
2	-1	2	-1		
3		-1	2	-1	
4			-1	2	-1
5				-1	1



	1	2	3	4	5	6	7	8	9	10	11	12
1	-1		1									
2	1	-1		1								
3		1			1							
4			-1			-1		1				
5				-1		1	-1		1			
6					-1		1			1		
7								-1			-1	
8									-1		1	-1
9										-1		1

	1	2	3	4	5	6	7	8	9
1	2	-1		-1					
2	-1	3	-1		-1				
3		-1	2			-1			
4	-1			3	-1		-1		
5		-1		-1	4	-1		-1	
6			-1		-1	3			-1
7				-1			2	-1	
8					-1		-1	3	-1
9						-1		-1	2

Nodes numbered in black

Edges numbered in blue

Properties of Incidence and Laplacian matrices

° ***Theorem 1:*** **Given G, In(G) and L(G) have the following properties** (proof on web page)

- L(G) is symmetric. (This means the eigenvalues of L(G) are real and its eigenvectors are real and orthogonal.)
- Let e = $[1,\ldots,1]^T$, i.e. the column vector of all ones. Then L(G)*e=0.
- In(G) * $(\mathrm{In}(G))^T$ = L(G). This is independent of the signs chosen for each column of In(G).
- Suppose L(G)*v = l*v, v != 0, so that v is an eigenvector and l an eigenvalue of L(G). Then

$$l = \|\, \mathrm{In}(G)^T * v \,\|^2 \,/\, \|\, v \,\|^2 \qquad \ldots \ \|x\|^2 = S_k\, x_k^2$$
$$= S \{ (v(i)-v(j))^2 \text{ for all edges } e=(i,j) \} \,/\, S_i\, v(i)^2$$

- The eigenvalues of L(G) are nonnegative:
  - $0 = l_1 \le l_2 \le \ldots \le l_n$
- The number of connected components of G is equal to the number of l_i equal to 0. In particular, l_2 != 0 if and only if G is connected.

° ***Definition*****:** $l_2(L(G))$ **is the algebraic connectivity of G**

Spectral Bisection Algorithm

° **Spectral Bisection Algorithm:**

- Compute eigenvector v_2 corresponding to $l_2(L(G))$
- For each node n of G
  - if $v_2(n) < 0$ put node n in partition N-
  - else put node n in partition N+

° **Why does this make sense? First reasons...**

° ***Theorem 2 (Fiedler, 1975):*** **Let G be connected, and N- and N+ defined as above. Then N- is connected. If no $v_2(n) = 0$, then N+ is also connected.** (proof on web page)

° **Recall $l_2(L(G))$ is the algebraic connectivity of G**

° ***Theorem 3 (Fiedler):*** **Let $G_1(N,E_1)$ be a subgraph of G(N,E), so that G_1 is "less connected" than G. Then $l_2(L(G_1)) \le l_2(L(G))$, i.e. the algebraic connectivity of G_1 is less than or equal to the algebraic connectivity of G.** (proof on web page)

Motivation for Spectral Bisection: Vibrating String

° Vibrating string has modes of vibration, or harmonics

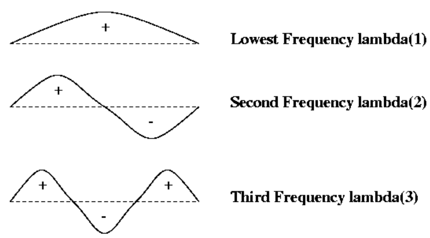
° Modes computable as follows

- Model string as masses connected by springs (a 1D mesh)
- Write down F=ma for coupled system, get matrix A
- Eigenvalues and eigenvectors of A are frequencies and shapes of modes

° Label nodes by whether mode - or + to get N- and N+

° Same idea for other graphs (eg planar graph ~ trampoline)

Modes of a Vibrating String



Details for vibrating string

° Force on mass j = k*[x(j-1) - x(j)] + k*[x(j+1) - x(j)]

= -k*[-x(j-1) + 2*x(j) - x(j+1)]

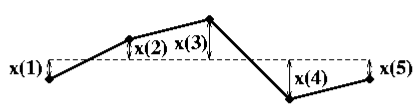
° F=ma yields m*x''(j) = -k*[-x(j-1) + 2*x(j) - x(j+1)] (*)

° Writing (*) for j=1,2,…,n yields

$$m * \frac{d^2}{dx^2} \begin{pmatrix} x(1) \\ x(2) \\ \dots \\ x(j) \\ \dots \\ x(n) \end{pmatrix} = -k* \begin{pmatrix} 2*x(1) - x(2) \\ -x(1) + 2*x(2) - x(3) \\ \dots \\ -x(j-1) + 2*x(j) - x(j+1) \\ \dots \\ 2*x(n-1) - x(n) \end{pmatrix} = -k* \begin{pmatrix} 2 & -1 & & & \\ -1 & 2 & -1 & & \\ & & \dots & & \\ & & -1 & 2 & -1 \\ & & & \dots & \\ & & & -1 & 2 \end{pmatrix} * \begin{pmatrix} x(1) \\ x(2) \\ \dots \\ x(j) \\ \dots \\ x(n) \end{pmatrix} = -k*L* \begin{pmatrix} x(1) \\ x(2) \\ \dots \\ x(j) \\ \dots \\ x(n) \end{pmatrix}$$

(-m/k) x'' = L*x

Vibrating Mass Spring System



Details for vibrating string - continued

° $-(m/k)\, x'' = L*x$, where $x = [x_1,x_2,\ldots,x_n\,]^T$

° Seek solution of form $x(t) = \sin(a*t) * x0$

- $L*x0 = (m/k)*a^2 * x0 = l * x0$
- For each integer i, get $l = 2*(1-\cos(i*p/(n+1)))$, $x0 = \begin{pmatrix} \sin(1*i*p/(n+1)) \\ \sin(2*i*p/(n+1)) \\ \ldots \\ \sin(n*i*p/(n+1)) \end{pmatrix}$
- Thus x0 is a sine curve with frequency proportional to i
- Thus $a^2 = 2*k/m *(1-\cos(i*p/(n+1)))$ or $a \sim \mathrm{sqrt}(k/m)*p*i/(n+1)$

° $L = \begin{pmatrix} 2 & -1 & & \\ -1 & 2 & -1 & \\ & \ldots. & & \\ & & -1 & 2 \end{pmatrix}$ not quite L(1D mesh), but we can fix that ...

A “vibrating string” for L(1D mesh)

° First equation changes to $m*x''(1) = -k*[-x(2)+\cancel{2}x(1)]$

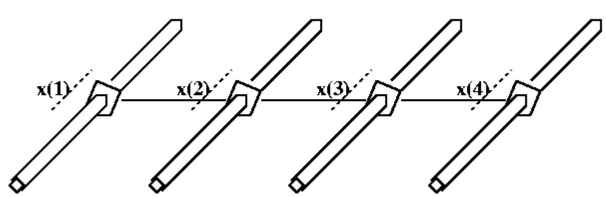
- First row of T changes from [2 -1 0 …] to [1 -1 0 …]

° Last equation changes to $m*x''(n)=-k*[-x(n-1) + \cancel{2}x(n)]$

- Last row of T changes from [… 0 -1 2] to [… 0 -1 1]

° Component j of i-th eigenvector changes to $\cos((j-.5)*(i-1)*p/n)$

"Vibrating String" for Spectral Bisection

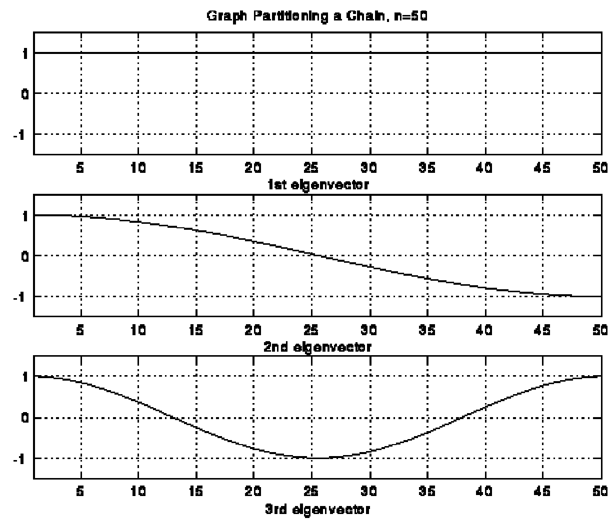


Eigenvectors of L(1D mesh)

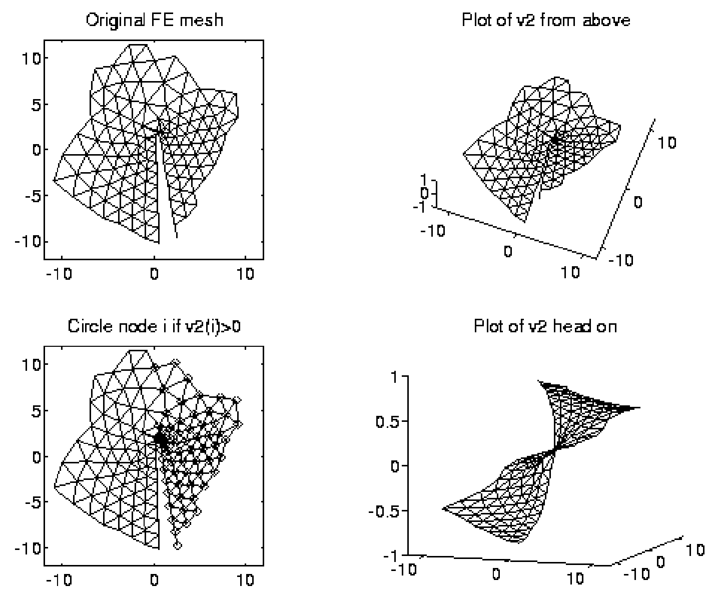
Eigenvector 1
(all ones)

Eigenvector 2

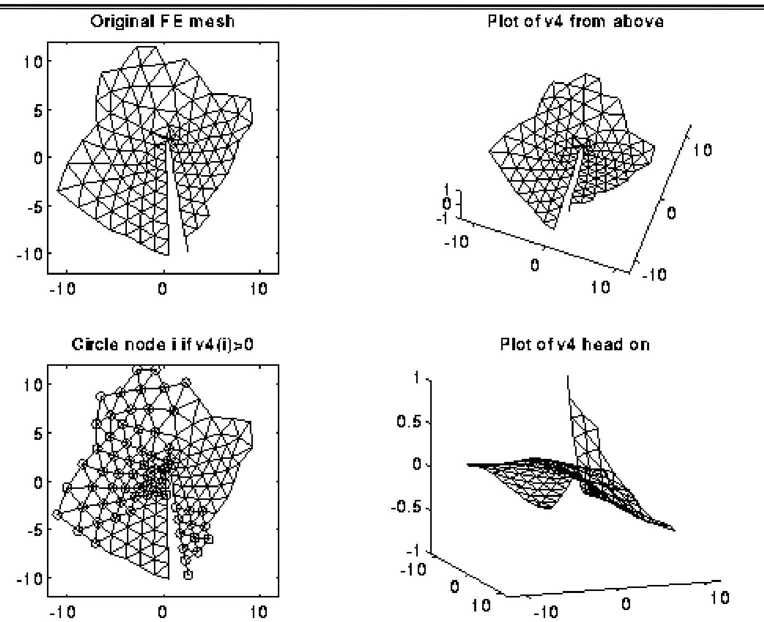
Eigenvector 3



2nd eigenvector of L(planar mesh)



4th eigenvector of L(planar mesh)



Motivation for Spectral Bisection: Continuous Approximation to a discrete optimization problem

° Use L(G) to count the number of edges from N- to N+

° *Lemma 1*: Let N = N- U N+ be a partition of G(N,E). Let x(j) = -1 if j is in N- and x(j) = +1 if j is in N+. Then
(proof on web page)

The number of edges connecting N- and N+
= .25 * x^T * L(G) * x
= .25 * $S_{i,k}$ x(i) * L(G)(i,k) * x(k)
= .25 * S { $(x(i) - x(k))^2$ for all edges (i,j) }

° Restate partitioning problem as finding vector x with entries +1 or -1 such that

- S_k x(k) = 0, i.e. |N+| = |N-|
- # edges connecting N+ to N- = .25*x^T*L(G)*x is minimized
- Put node j in N+ (or N-) if x(j) >=0 (or < 0)

Converting a discrete to a continuous problem

° **Discrete**: Find x with entries +1 or -1 such that

- S_k x(k) = 0, i.e. |N+| = |N-|
- # edges connecting N+ to N- = .25*x^T*L(G)*x is minimized
- Put node j in N+ (or N-) if x(j) >=0 (or < 0)

° **Continuous**: Find x with real entries such that

- S_k x(k) = 0 and S_k $(x(k))^2$ = |N| (set includes discrete one above)
- .25*x^T*L(G)*x is minimized
- Put node j in N+ (or N-) if x(j) >=0 (or < 0)

° ***Theorem 4*** (Courant/Fischer "minimax theorem"): x satisfying continuous problem is eigenvector v_2, for l_2. (proof on web page)

° ***Theorem 5:*** The minimum number of edges connecting N+ and N- in any partitioning with |N+|=|N-| is at least .25*|N|* l_2. (proof on web page)

- The larger the algebraic connectivity l_2, the more edges we need to cut to bisect the graph

Computing v_2 and l_2 of L(G) using Lanczos

° Given any n-by-n symmetric matrix A (such as L(G)) **Lanczos** computes a k-by-k "approximation" T by doing k matrix-vector products, k << n

```
Choose an arbitrary starting vector r
b(0) = ||r||
j=0
repeat
   j=j+1
   q(j) = r/b(j-1)           ... scale a vector
   r = A*q(j)                ... matrix vector multiplication, the most expensive step
   r = r - b(j-1)*v(j-1)     ... "saxpy", or scalar*vector + vector
  a(j) = v(j)^T * r          ... dot product
  r = r - a(j)*v(j)          ... "saxpy"
  b(j) = ||r||               ... compute vector norm
until convergence            ... details omitted
```

$$T = \begin{pmatrix} a(1) & b(1) & & & & \\ b(1) & a(2) & b(2) & & & \\ & b(2) & a(3) & b(3) & & \\ & & \dots & \dots & \dots & \\ & & & b(k-2) & a(k-1) & b(k-1) \\ & & & & b(k-1) & a(k) \end{pmatrix}$$

° Approximate A's eigenvalues/vectors using T's

References

- Details of all proofs on web page

- A. Pothen, H. Simon, K.-P. Liou, "Partitioning sparse matrices with eigenvectors of graphs", SIAM J. Mat. Anal. Appl. 11:430-452 (1990)
- M. Fiedler, "Algebraic Connectivity of Graphs", Czech. Math. J., 23:298-305 (1973)
- M. Fiedler, Czech. Math. J., 25:619-637 (1975)
- B. Parlett, "The Symmetric Eigenproblem", Prentice-Hall, 1980
- www.cs.berkeley.edu/~ruhe/lantplht/lantplht.html
- www.netlib.org/laso