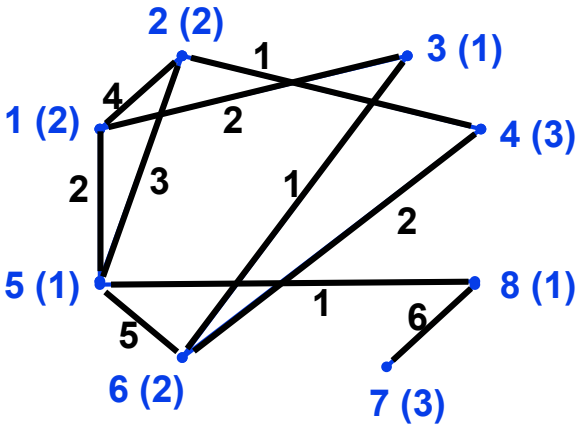
Graph Partitioning and Domain decomposition

Outline of Graph Partitioning Lecture

- Review definition of Graph Partitioning problem
- Overview of heuristics
- Partitioning with Nodal Coordinates
  - Ex: In finite element models, node at point in (x,y) or (x,y,z) space
- Partitioning without Nodal Coordinates
  - Ex: In model of WWW, nodes are web pages
- Multilevel Acceleration
  - BIG IDEA, appears often in scientific computing
- Comparison of Methods and Applications
- Beyond Graph Partitioning: Hypergraphs

Definition of Graph Partitioning

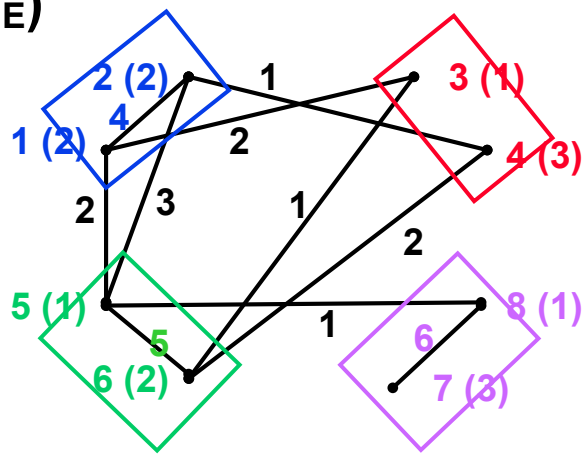
- Given a graph G = (N, E, W_N, W_E)
  - N = nodes (or vertices),
  - W_N = node weights (between brackets)
  - E = edges
  - W_E = edge weights



- 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 U N_2 U … U 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 U N_2: Graph Bisection

Definition of Graph Partitioning

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



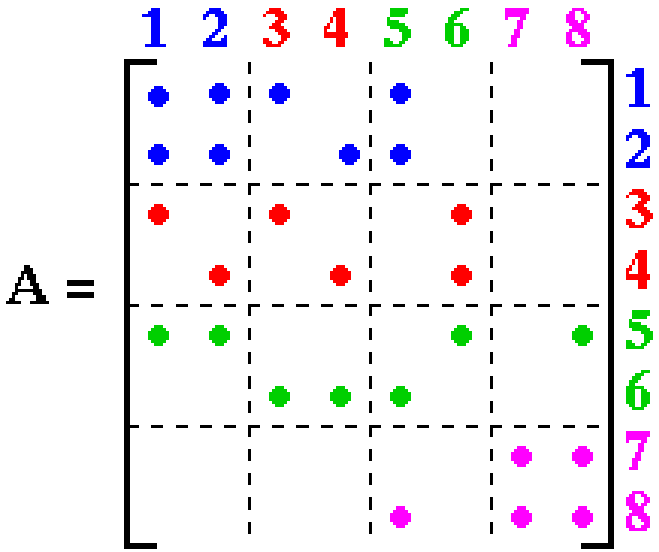
- Ex: N = {tasks}, W_N = {task costs}, edge (j,k) in E means task j sends W_E(j,k) words to task k
- Choose a partition N = N_1 U N_2 U … U N_P such that
  - The sum of the node weights in each N_j is "about the same"
  - The sum of all edge weights of edges connecting all different pairs N_j and N_k is minimized (shown in black)
- Ex: balance the work load, while minimizing communication
- Special case of N = N_1 U N_2: Graph Bisection

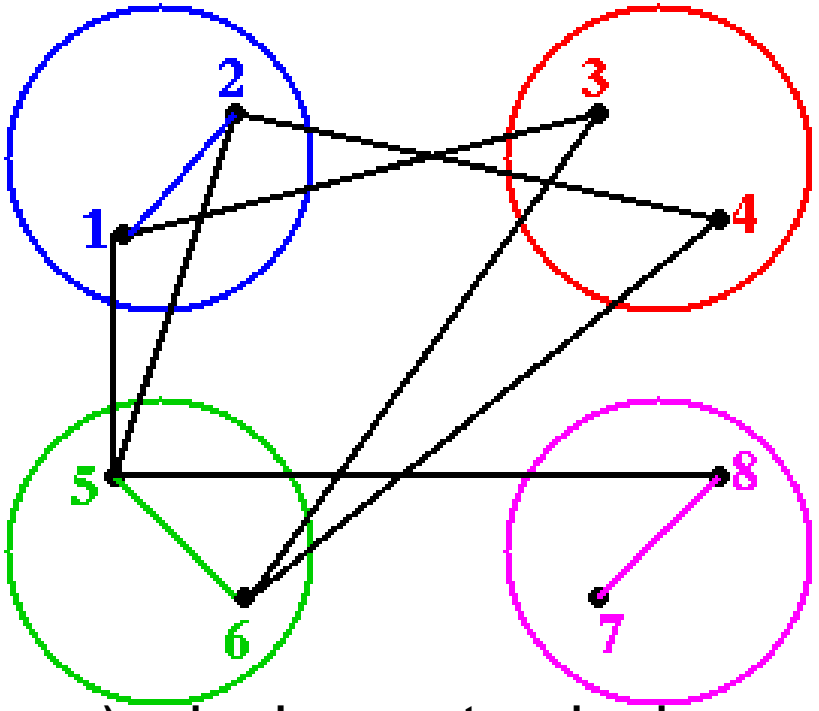
Some Applications

- Telephone network design
  - Original application, algorithm due to Kernighan
- Load Balancing while Minimizing Communication
- Sparse Matrix times Vector Multiplication (SpMV)
  - Solving PDEs
  - N = {1,…,n}, (j,k) in E if A(j,k) nonzero,
  - $W_N(j)$ = #nonzeros in row j, $W_E(j,k) = 1$
- VLSI Layout
  - N = {units on chip}, E = {wires}, $W_E(j,k)$ = wire length
- Sparse Gaussian Elimination
  - Used to reorder rows and columns to increase parallelism, and to decrease "fill-in"
- Data mining and clustering
- Physical Mapping of DNA
- Image Segmentation

Sparse Matrix Vector Multiplication y = y +A*x

Partitioning a Sparse Symmetric Matrix





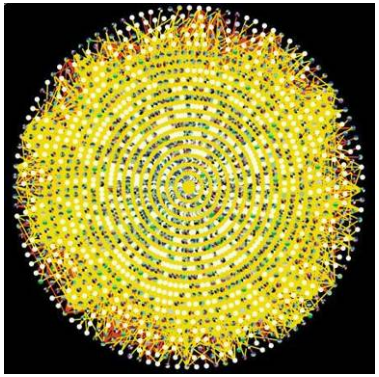

```
… declare A_local, A_remote(1:num_procs), x_local, x_remote, y_local
y_local = y_local + A_local * x_local
for all procs P that need part of x_local
        send(needed part of x_local, P)
for all procs P owning needed part of x_remote
        receive(x_remote, P)
        y_local = y_local + A_remote(P)*x_remote
```

Large-scale data analysis

- Graph abstractions are very useful to analyze complex data sets.
- Sources of data: petascale simulations, experimental devices, the Internet, sensor networks
- Challenges: data size, heterogeneity, uncertainty, data quality

Astrophysics: massive datasets, temporal variations

Bioinformatics: data quality, heterogeneity



Social Informatics: new analytics challenges, data uncertainty

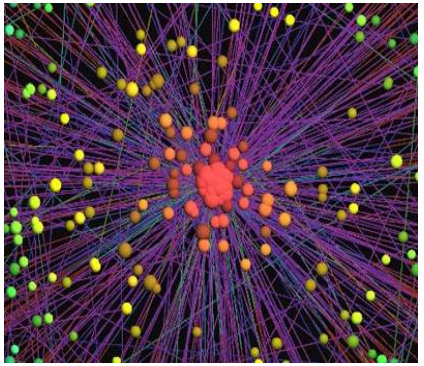
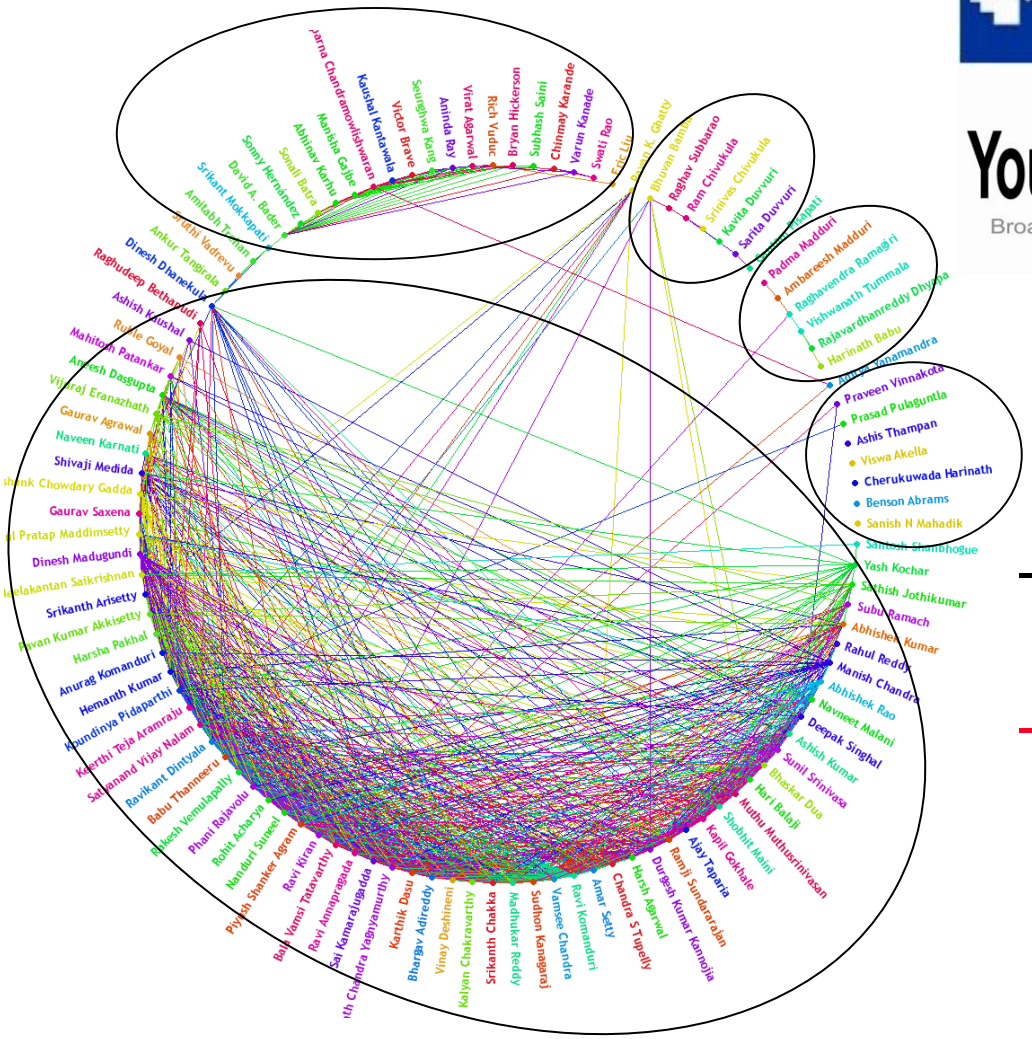


Image sources: (1) http://physics.nmt.edu/images/astro/hst_starfield.jpg (2,3) www.visualComplexity.com

Graph-theoretic problems in social networks



- Targeted advertising: clustering and centrality
- Studying the spread of information

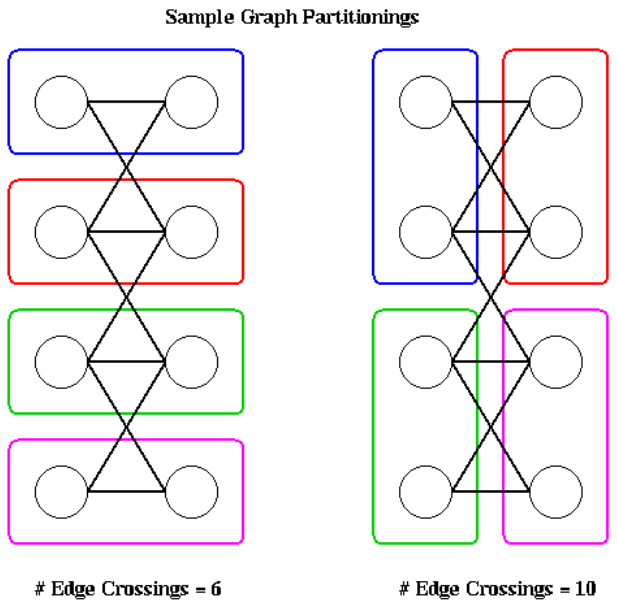
Image Source: Nexus (Facebook application)

Cost of Graph Partitioning

- Many possible partitionings to search
- Just to divide in 2 parts there are:

  n choose n/2 = $n!/((n/2)!)^2 \sim$

  $(2/(n\pi))^{1/2} * 2^n$ possibilities



- Choosing optimal partitioning is NP-complete
  - (NP-complete = can be proved it is a hard as other well-known hard problems in a class Nondeterministic Polynomial time)
  - Only known exact algorithms have cost = exponential(n)
- We need good heuristics

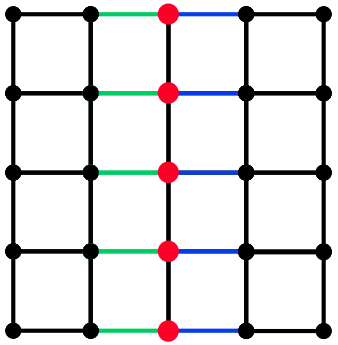
Outline of Graph Partitioning Lectures

- Review definition of Graph Partitioning problem
- Overview of heuristics
- **Partitioning with Nodal Coordinates**
  - **Ex: In finite element models, node at point in (x,y) or (x,y,z) space**
- **Partitioning without Nodal Coordinates**
  - **Ex: In model of WWW, nodes are web pages**
- **Multilevel Acceleration**
  - **BIG IDEA, appears often in scientific computing**
- **Comparison of Methods and Applications**
- **Beyond Graph Partitioning: Hypergraphs**

Edge Separators vs. Vertex Separators

- Edge Separator: E_s (subset of E) separates G if removing E_s from E leaves two ~equal-sized, disconnected components of N: N_1 and N_2
- Vertex Separator: N_s (subset of N) separates G if removing N_s and all incident edges leaves two ~equal-sized, disconnected components of N: N_1 and N_2

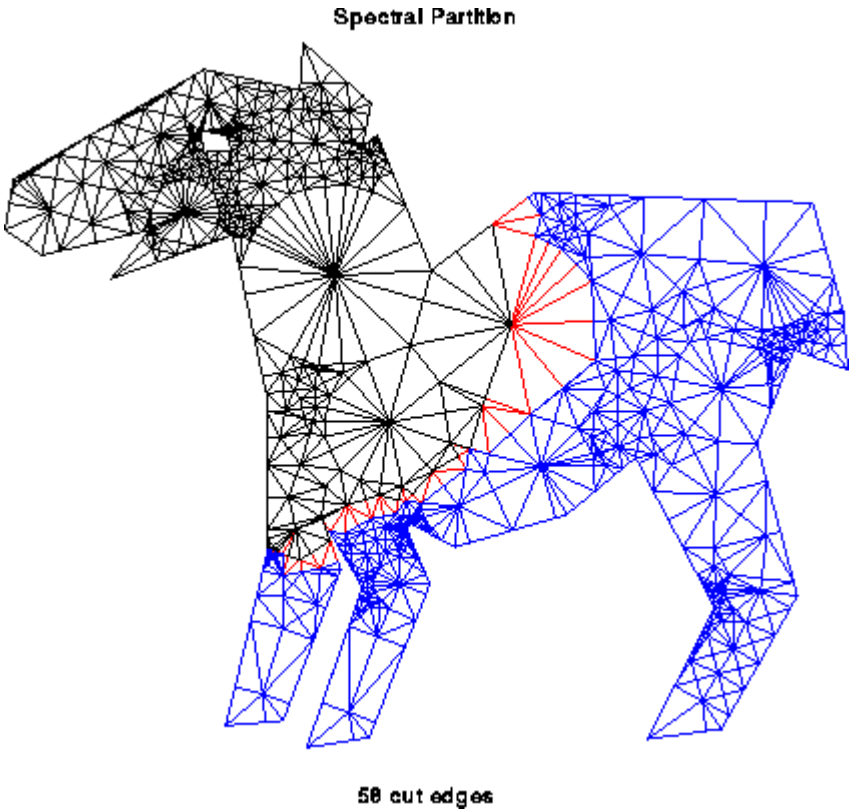
G = (N, E), Nodes N and Edges E
E_s = green edges or blue edges
N_s = red vertices



- Making an N_s from an E_s: pick one endpoint of each edge in E_s
  - $|N_s| \leq |E_s|$
- Making an E_s from an N_s: pick all edges incident on N_s
  - $|E_s| \leq d * |N_s|$ where d is the maximum degree of the graph
- We will find Edge or Vertex Separators, as convenient

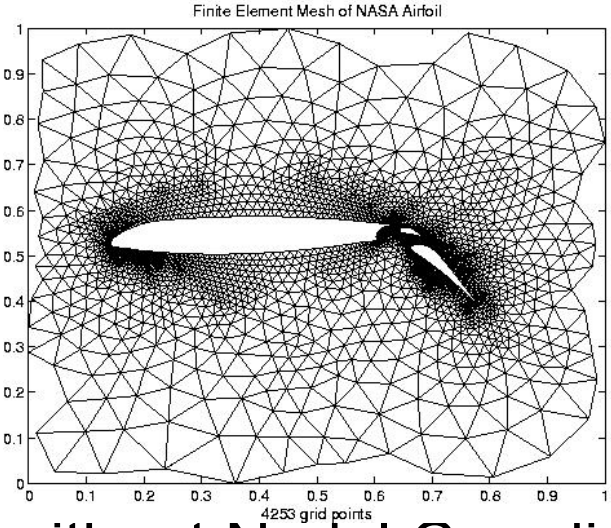
First Heuristic: Repeated Graph Bisection

- To partition N into 2^k parts
  - bisect graph recursively k times
- Henceforth discuss mostly graph bisection



Overview of Bisection Heuristics

- Partitioning with Nodal Coordinates
  - Each node has x,y,z coordinates → partition space



- Partitioning without Nodal Coordinates
  - E.g., Sparse matrix of Web documents
    - A(j,k) = # times keyword j appears in URL k
- Multilevel acceleration (BIG IDEA)
  - Approximate problem by "coarse graph," do so recursively

Outline of Graph Partitioning Lectures

- Review definition of Graph Partitioning problem
- Overview of heuristics
- **Partitioning with Nodal Coordinates**
  - **Ex: In finite element models, node at point in (x,y) or (x,y,z) space**
- **Partitioning without Nodal Coordinates**
  - **Ex: In model of WWW, nodes are web pages**
- **Multilevel Acceleration**
  - **BIG IDEA, appears often in scientific computing**
- **Comparison of Methods and Applications**
- **Beyond Graph Partitioning: Hypergraphs**

Nodal Coordinates: How Well Can We Do?

- A planar graph can be drawn in plane without edge crossings
- Ex: m x m grid of m^2 nodes: $\exists$ vertex separator N_s with $|N_s| = m = |N|^{1/2}$ (see earlier slide for m=5)
- *Theorem* (Tarjan, Lipton, 1979): If G is planar, $\exists$ N_s such that
  - $N = N_1 \cup N_s \cup N_2$ is a partition,
  - $|N_1| <= 2/3 |N|$ and $|N_2| <= 2/3 |N|$
  - $|N_s| <= (8 * |N|)^{1/2}$
- Theorem motivates intuition of following algorithms

Nodal Coordinates: Inertial Partitioning

- For a graph in 2D, choose line with half the nodes on one side and half on the other
  - In 3D, choose a plane, but consider 2D for simplicity
- Determine a line L, and then choose a line $L^{\perp}$ perpendicular to it, with half the nodes on either side

1. **Determine a line L through the points**

   L given by a*(x-xbar)+b*(y-ybar)=0,

   with $a^2+b^2=1$; (a,b) is unit vector $\perp$ to L

2. **Project each point to the line**

   For each $n_j = (x_j, y_j)$, compute coordinate

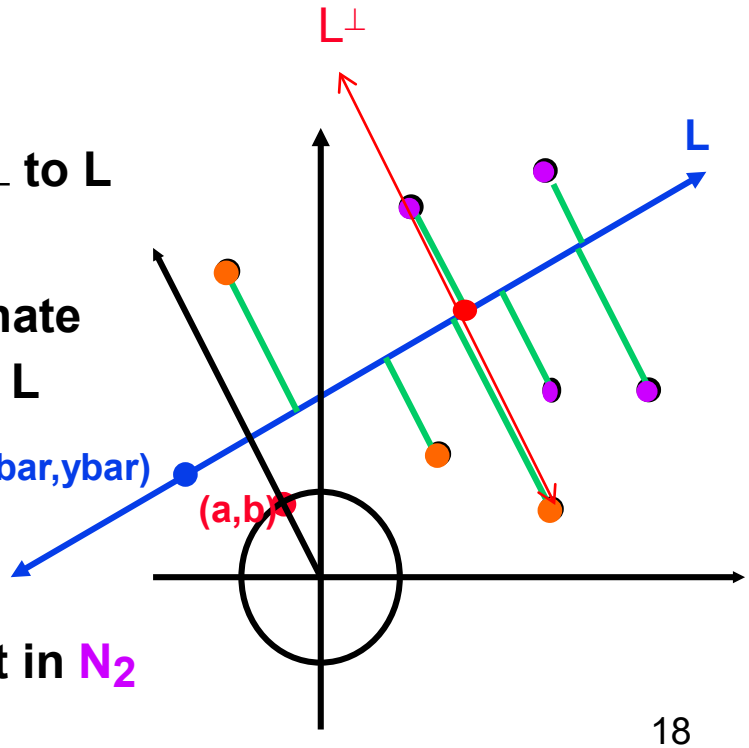
   $S_j = -b*(x_j-xbar) + a*(y_j-ybar)$ along L

3. **Compute the median**

   Let Sbar = median(S_1,…,S_n)

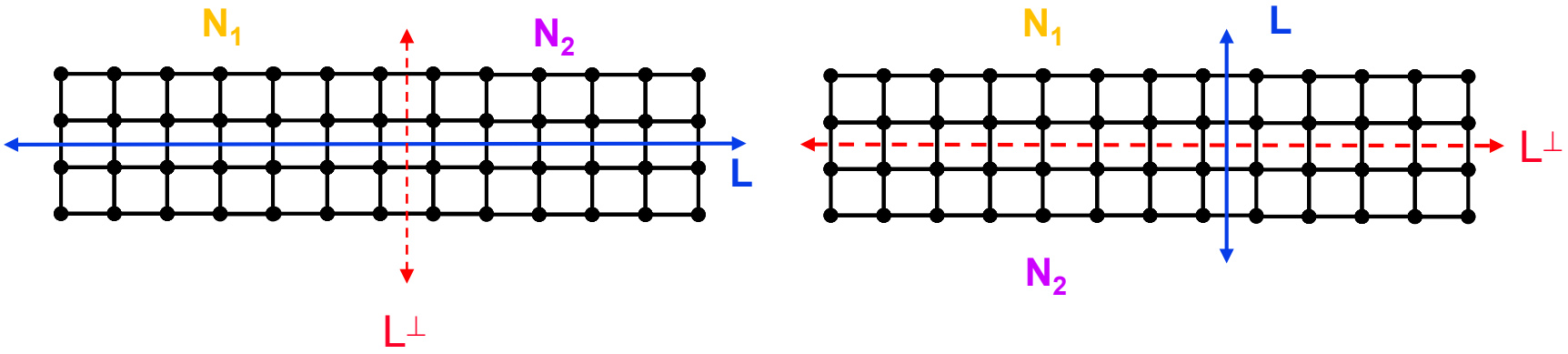
4. **Use median to partition the nodes**

   Let nodes with S_j < Sbar be in N_1, rest in N_2



Inertial Partitioning: Choosing L

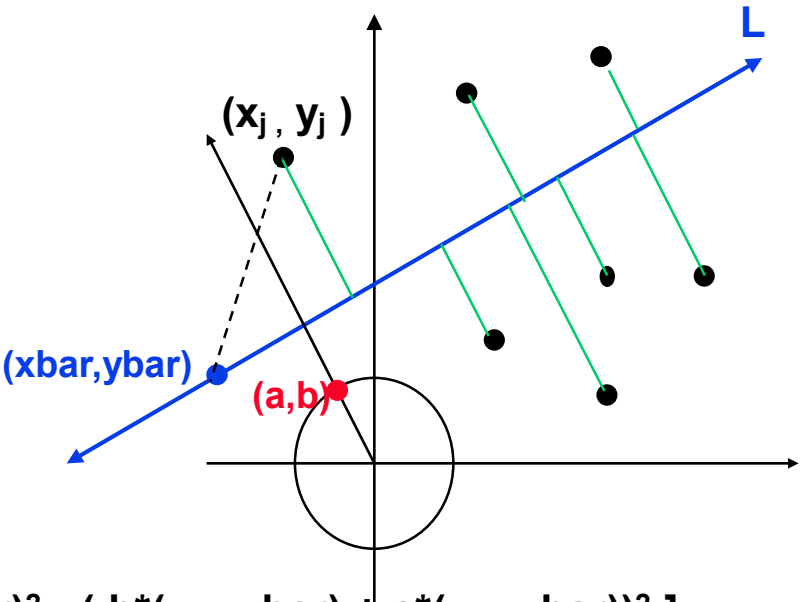
- Clearly prefer L, $L^{\perp}$ on left below



- Mathematically, L is a total least squares fit of the nodes
  - Minimize sum of squares of distances to L (green lines on last slide)
  - Equivalent to choosing L as axis of rotation that minimizes the moment of inertia of nodes (unit weights) - source of name

Inertial Partitioning: determine L (continued)

(a,b) is unit vector perpendicular to L



Σ_j (length of j-th green line)^2

$= \Sigma_j \; [(x_j - xbar)^2 + (y_j - ybar)^2 - (-b*(x_j - xbar) + a*(y_j - ybar))^2]$

… Pythagorean Theorem

$= a^2 * \Sigma_j (x_j - xbar)^2 \; + \; 2*a*b* \Sigma_j (x_j - xbar)*(x_j - ybar) \; + \; b^2 \Sigma_j (y_j - ybar)^2$

$= a^2 * X1 \; + \; 2*a*b* \; X2 \; + \; b^2 * X3$

$$= [a\ b] * \begin{bmatrix} X1 & X2 \\ X2 & X3 \end{bmatrix} * \begin{bmatrix} a \\ b \end{bmatrix}$$

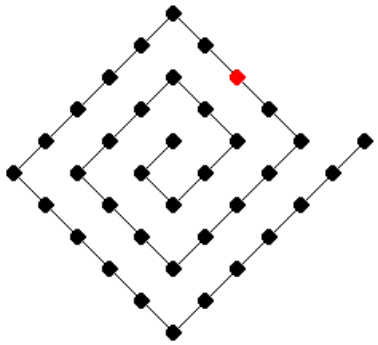
Minimized by choosing

(xbar , ybar) = $(\Sigma_j x_j , \Sigma_j y_j) / n$ = center of mass

(a,b) = eigenvector of smallest eigenvalue of $\begin{bmatrix} X1 & X2 \\ X2 & X3 \end{bmatrix}$

Nodal Coordinates: Summary

- Other variations on these algorithms
- Algorithms are efficient (i.e., fast)
- Rely on graphs having nodes connected (mostly) to "nearest neighbors" in space
  - algorithm does not depend on where actual edges are!
- Common when graph arises from physical model
- Ignores edges, but can be used as good starting guess for subsequent partitioning that does examine edges
- Can do poorly if graph connection is not spatial:

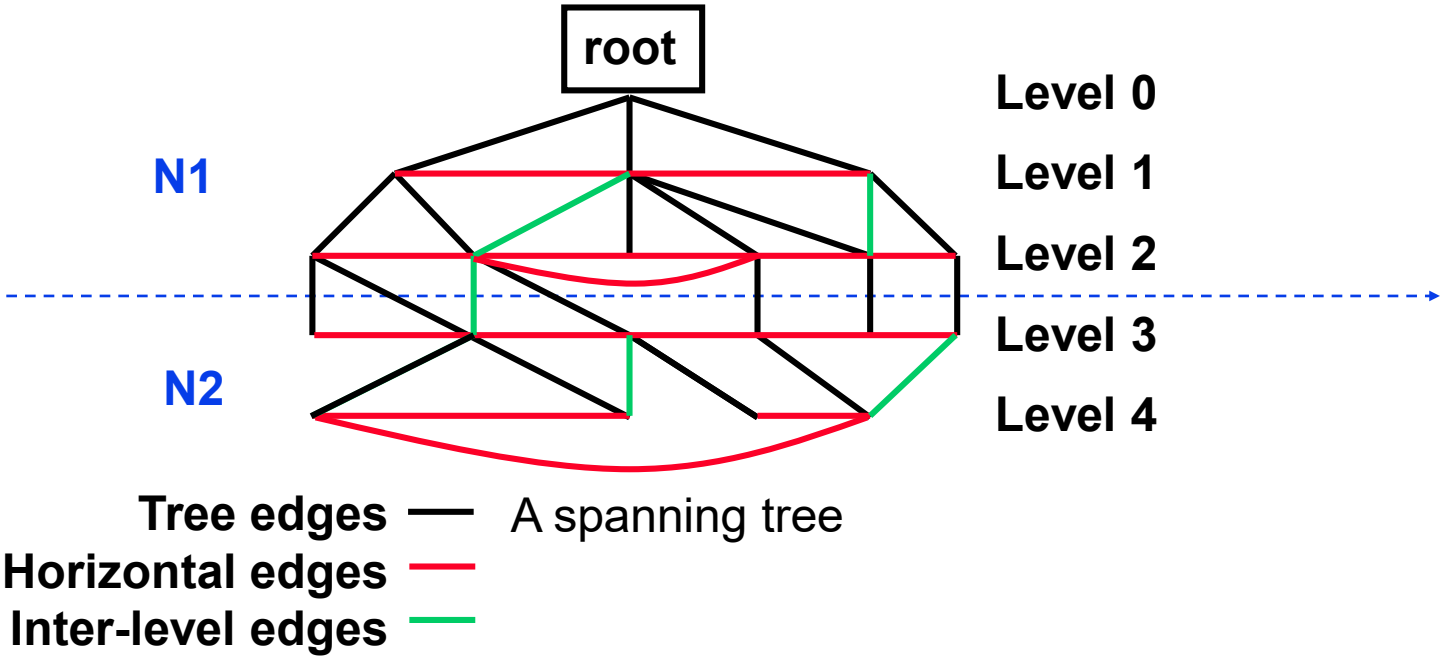


Outline of Graph Partitioning Lectures

- Review definition of Graph Partitioning problem
- Overview of heuristics
- **Partitioning with Nodal Coordinates**
  - **Ex: In finite element models, node at point in (x,y) or (x,y,z) space**
- **Partitioning without Nodal Coordinates**
  - **Ex: In model of WWW, nodes are web pages**
- **Multilevel Acceleration**
  - **BIG IDEA, appears often in scientific computing**
- **Comparison of Methods and Applications**
- **Beyond Graph Partitioning: Hypergraphs**

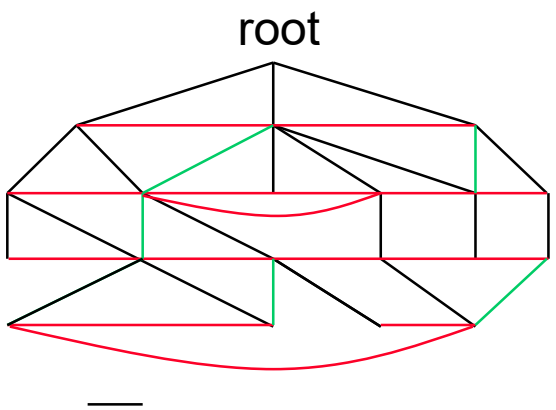
Coordinate-Free: Breadth First Search (BFS)

- Given G(N,E) and a root node r in N, BFS produces
  - **A subgraph T of G (same nodes, subset of edges)**
  - **T is a tree rooted at r**
  - **Each node assigned a level = distance from r**



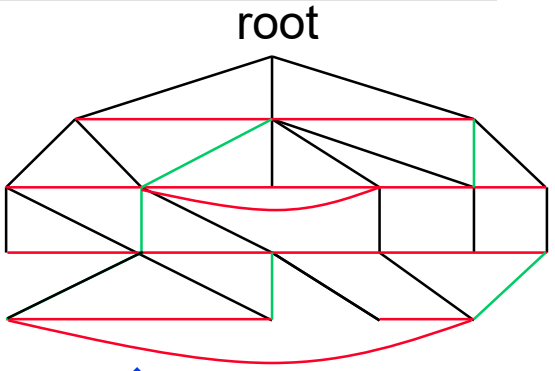
Breadth First Search (details)

- Queue (First In First Out, or FIFO)
  - Enqueue(x,Q) adds x to back of Q
  - x = Dequeue(Q) removes x from front of Q
- Compute Tree $T(N_T,E_T)$

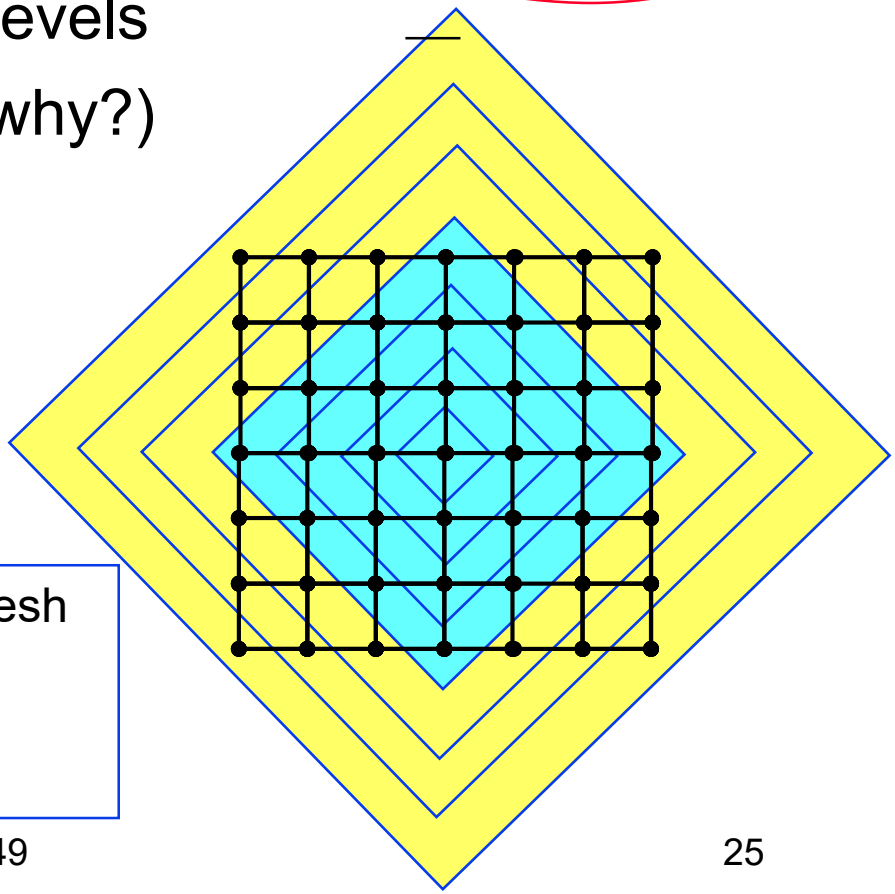



```
N_T = {(r,0)}, E_T = empty set      … Initially T = root r, which is at level 0
Enqueue((r,0),Q)                    … Put root on initially empty Queue Q
Mark r                              … Mark root as having been processed
While Q not empty                   … While nodes remain to be processed
    (n,level) = Dequeue(Q)          … Get a node to process
    For all unmarked children c of n
        N_T = N_T U (c,level+1)     … Add child c to N_T
        E_T = E_T U (n,c)           … Add edge (n,c) to E_T
        Enqueue((c,level+1),Q))     … Add child c to Q for processing
        Mark c                      … Mark c as processed
    Endfor
Endwhile
```

Partitioning via Breadth First Search



- BFS identifies 3 kinds of edges
  - Tree Edges - part of T
  - Horizontal Edges - connect nodes at same level
  - Interlevel Edges - connect nodes at adjacent levels
- No edges connect nodes in levels differing by more than 1 (why?)
- BFS partioning heuristic
  - N = N_1 U N_2, where
    - N_1 = {nodes at level <= L},
    - N_2 = {nodes at level > L}
  - Choose L so $|N_1|$ close to $|N_2|$



BFS partition of a 2D Mesh using center as root:
N1 = levels 0, 1, 2, 3
N2 = levels 4, 5, 6

Coordinate-Free: Kernighan/Lin

- Take an 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
- Given $G = (N,E,W_E)$ and a partitioning $N = A \cup B$, where $|A| = |B|$
  - $T = cost(A,B) = \Sigma \{W(e)$ where e connects nodes in A and B$\}$
  - Find subsets X of A and Y of B with $|X| = |Y|$
  - Consider swapping X and Y if it decreases cost:
    - $newA = (A - X) \cup Y$ and $newB = (B - Y) \cup X$
    - $newT = cost(newA, newB) < T = cost(A,B)$
- Need to compute newT efficiently for many possible X and Y, choose smallest (best)

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 = Σ {W(a,b) for b in B}
  - I(a) = internal cost of a in A = Σ {W(a,a') for other a' in A}
  - D(a) = cost of a in A = E(a) - I(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 (cost changes only when (a',a) or (a',b) exist)
  - 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 shown in red, $O(N^3)$
- 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 $2^{-|N|/30}$

A summary of improvements over Kernighan/Lin can be found in this recent survey:

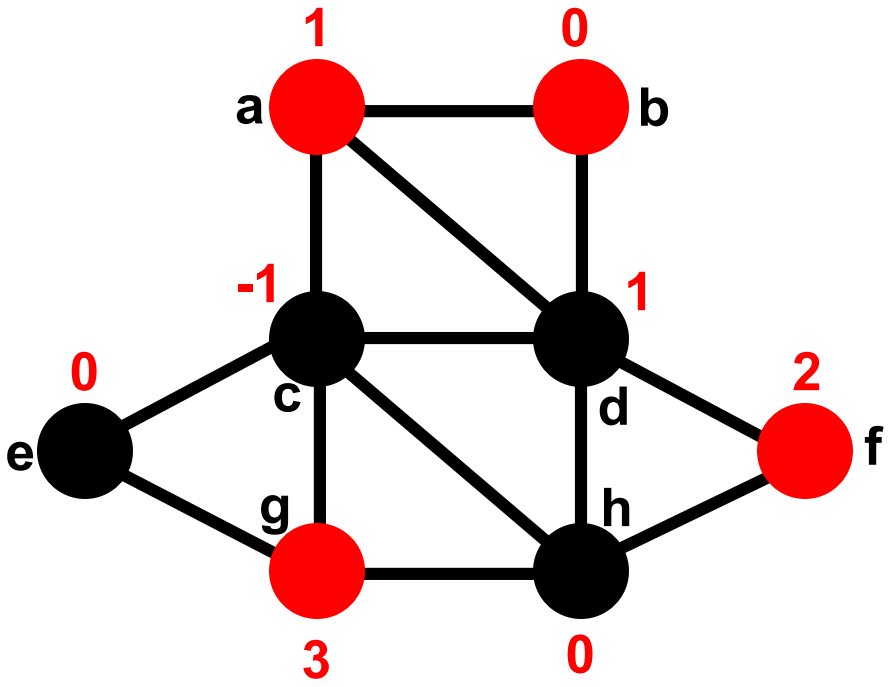
Buluç, A., Meyerhenke, H., Safro, I., Sanders, P., & Schulz, C. (2016). Recent advances in graph partitioning. In Algorithm Engineering.

Simplified Fiduccia-Mattheyses: Example (1)

Red nodes are in Part1; black nodes are in Part2.

The initial partition into two parts is arbitrary. In this case it cuts 8 edges.

The initial node gains by changing membership Part are shown in red.



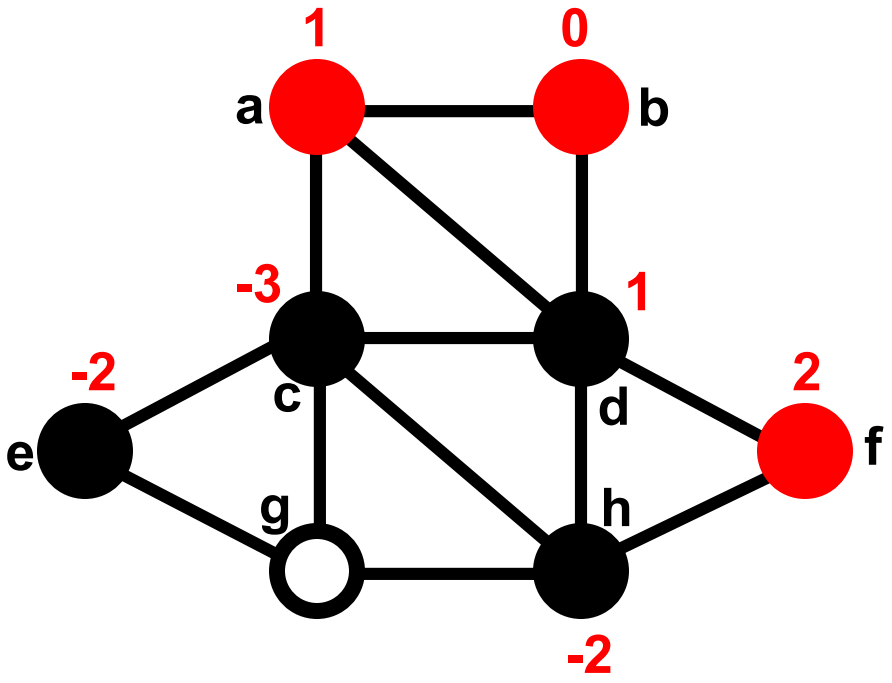
Nodes tentatively moved (and cut size after each pair):

none (8);

Simplified Fiduccia-Mattheyses: Example (2)

The node in Part1 with largest gain is g. We tentatively move it to Part2 and recompute the gains of its neighbors.

Tentatively moved nodes are hollow circles. After a node is tentatively moved its gain doesn't matter any more.



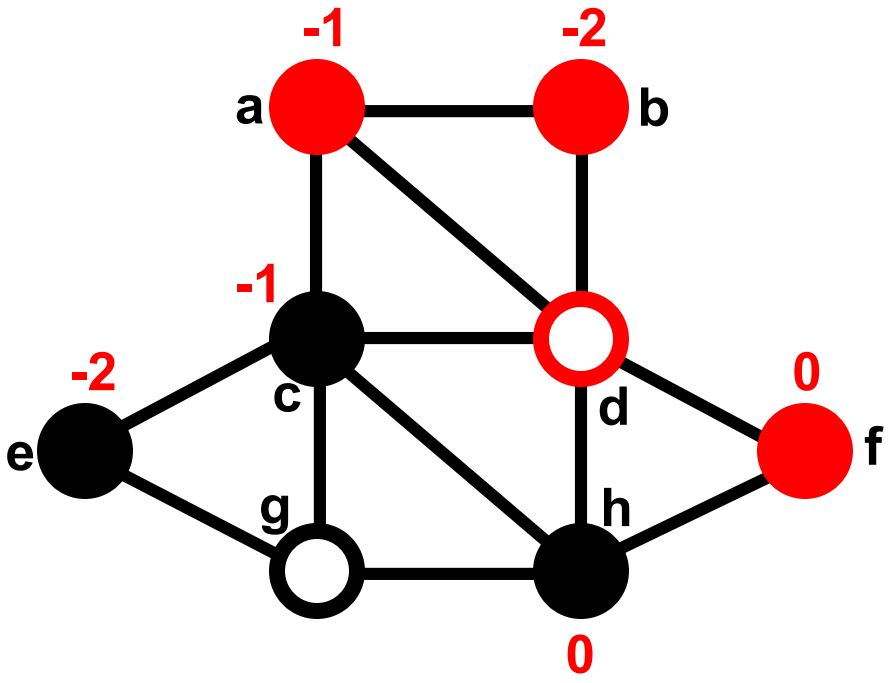
Nodes tentatively moved (and cut size after each pair):

none (8); g,

Simplified Fiduccia-Mattheyses: Example (3)

The node in Part2 with largest gain is d. We tentatively move it to Part1 and recompute the gains of its neighbors.

After this first tentative swap, the cut size is 4.

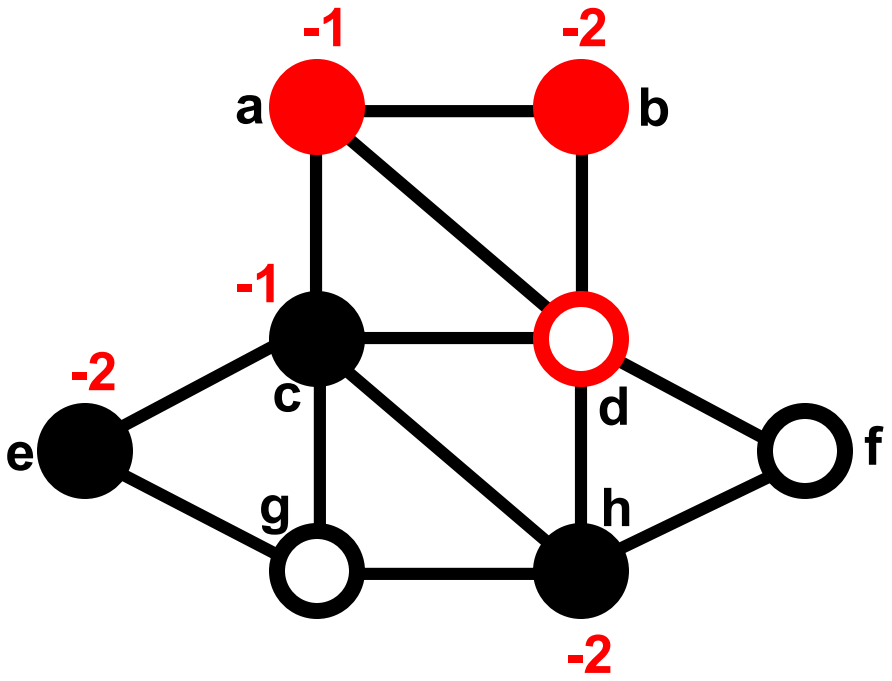


Nodes tentatively moved (and cut size after each pair):

none (8); g, d (4);

Simplified Fiduccia-Mattheyses: Example (4)

The unmoved node in Part1 with largest gain is f. We tentatively move it to Part2 and recompute the gains of its neighbors.



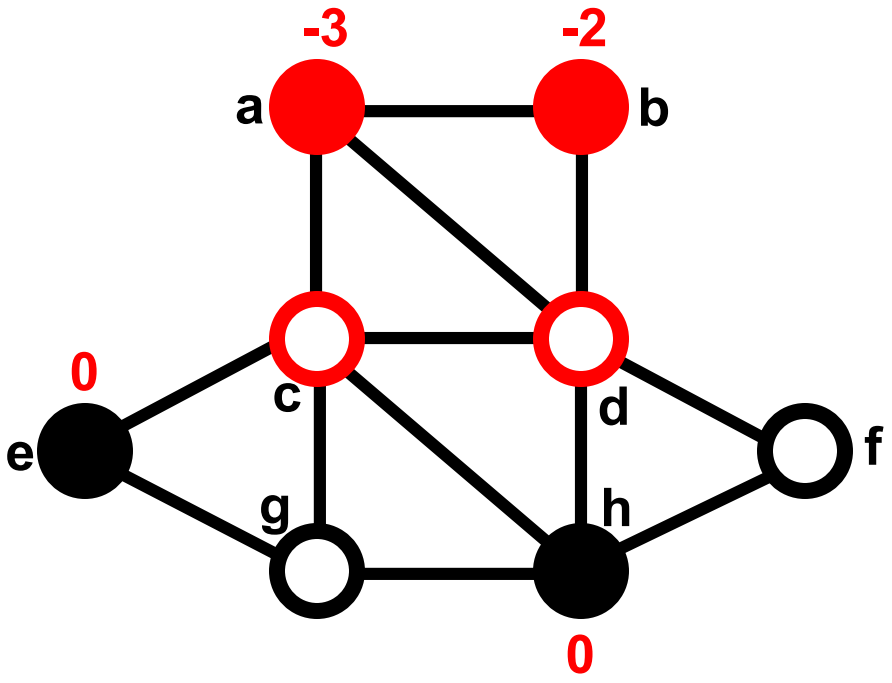
Nodes tentatively moved (and cut size after each pair):

none (8); g, d (4); f

Simplified Fiduccia-Mattheyses: Example (5)

The unmoved node in Part2 with largest gain is c. We tentatively move it to Part1 and recompute the gains of its neighbors.

After this tentative swap, the cut size is 5.

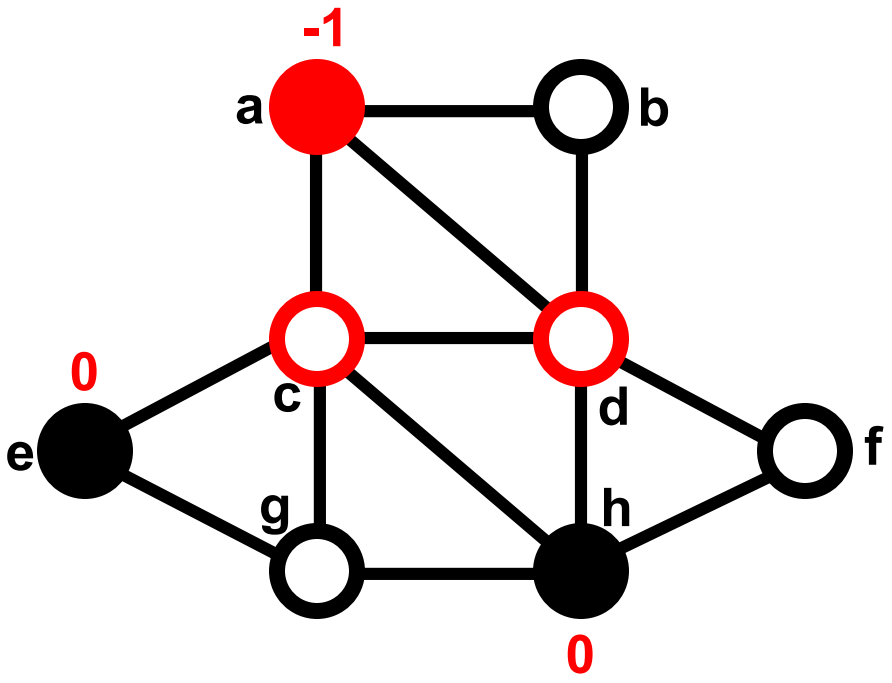


Nodes tentatively moved (and cut size after each pair):

none (8); g, d (4); f, c (5);

Simplified Fiduccia-Mattheyses: Example (6)

The unmoved node in Part1 with largest gain is b. We tentatively move it to Part2 and recompute the gains of its neighbors.



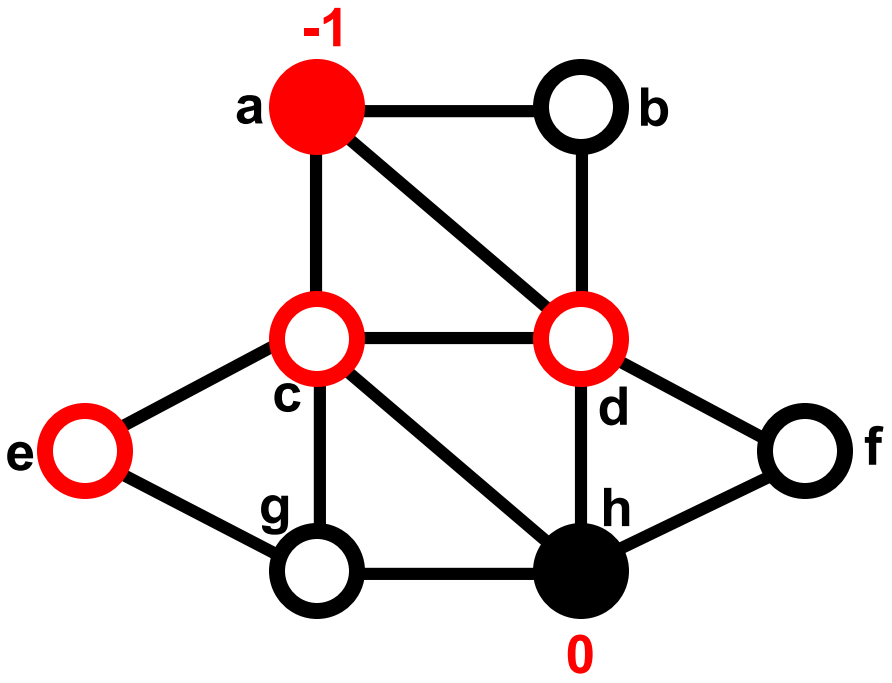
Nodes tentatively moved (and cut size after each pair):

none (8); g, d (4); f, c (5); b

Simplified Fiduccia-Mattheyses: Example (7)

There is a tie for largest gain between the two unmoved nodes in Part2. We choose one (say e) and tentatively move it to Part1. It has no unmoved neighbors so no gains are recomputed.

After this tentative swap the cut size is 7.

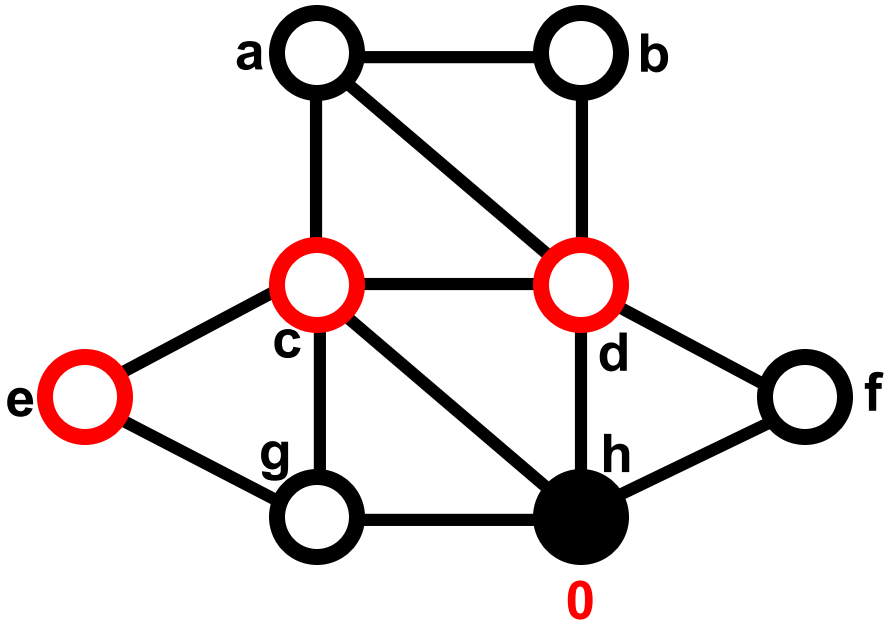


Nodes tentatively moved (and cut size after each pair):

none (8); g, d (4); f, c (5); b, e (7);

Simplified Fiduccia-Mattheyses: Example (8)

The unmoved node in Part1 with the largest gain (the only one) is a. We tentatively move it to Part2. It has no unmoved neighbors so no gains are recomputed.



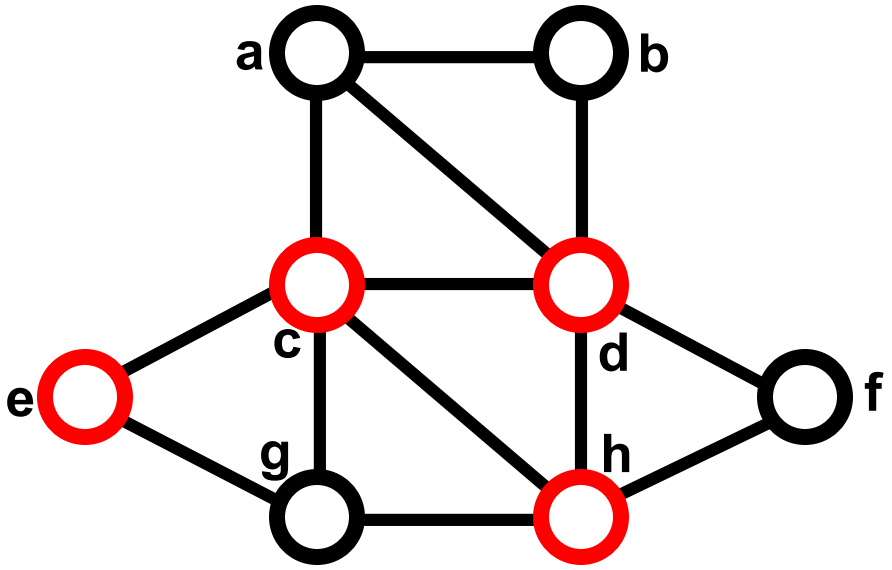
Nodes tentatively moved (and cut size after each pair):

none (8); g, d (4); f, c (5); b, e (7); a

Simplified Fiduccia-Mattheyses: Example (9)

The unmoved node in Part2 with the largest gain (the only one) is h. We tentatively move it to Part1.

The cut size after the final tentative swap is 8, the same as it was before any tentative moves.



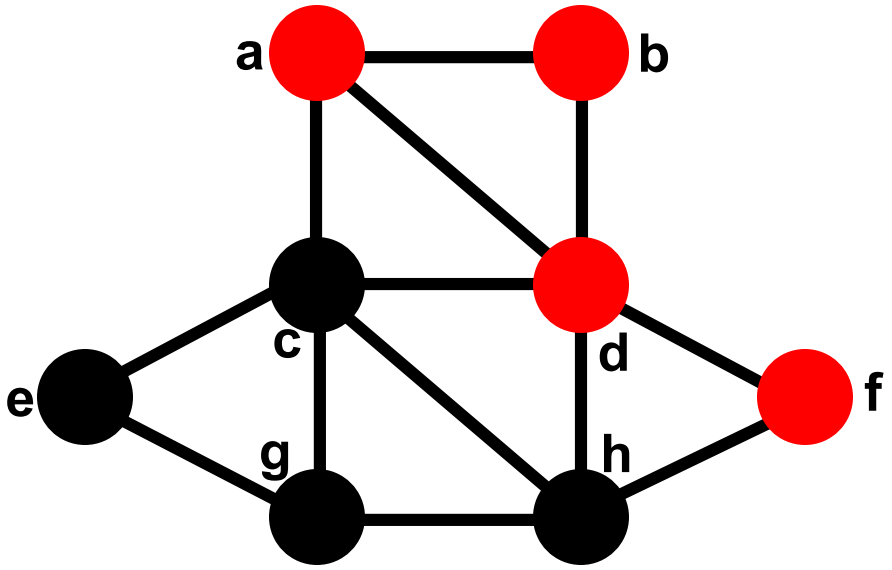
Nodes tentatively moved (and cut size after each pair):

none (8); g, d (4); f, c (5); b, e (7); a, h (8)

Simplified Fiduccia-Mattheyses: Example (10)

After every node has been tentatively moved, we look back at the sequence and see that the smallest cut was 4, after swapping g and d. We make that swap permanent and undo all the later tentative swaps.

This is the end of the first improvement step.



Nodes tentatively moved (and cut size after each pair):

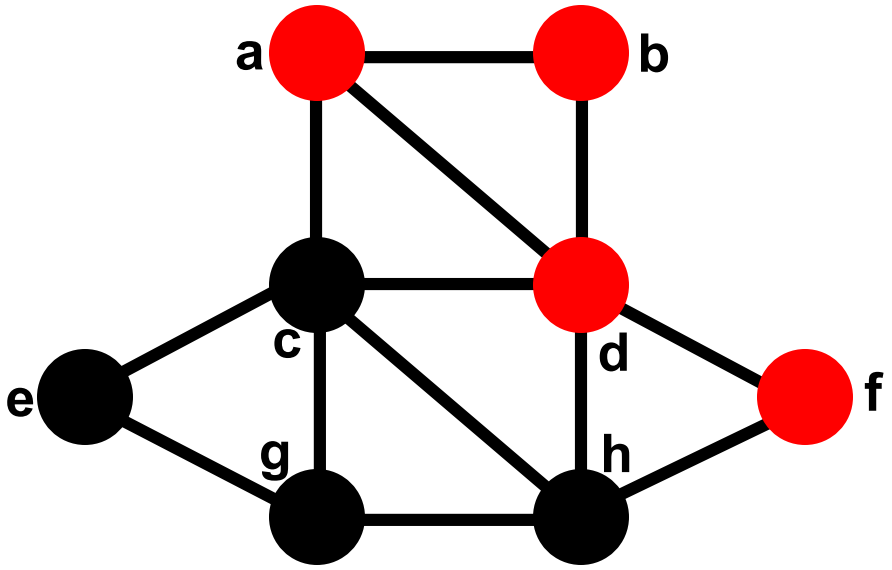
none (8); **g, d (4);** f, c (5); b, e (7); a, h (8)

Simplified Fiduccia-Mattheyses: Example (11)

Now we recompute the gains and do another improvement step starting from the new size-4 cut. The details are not shown.

The second improvement step doesn't change the cut size, so the algorithm ends with a cut of size 4.

In general, we keep doing improvement steps as long as the cut size keeps getting smaller.

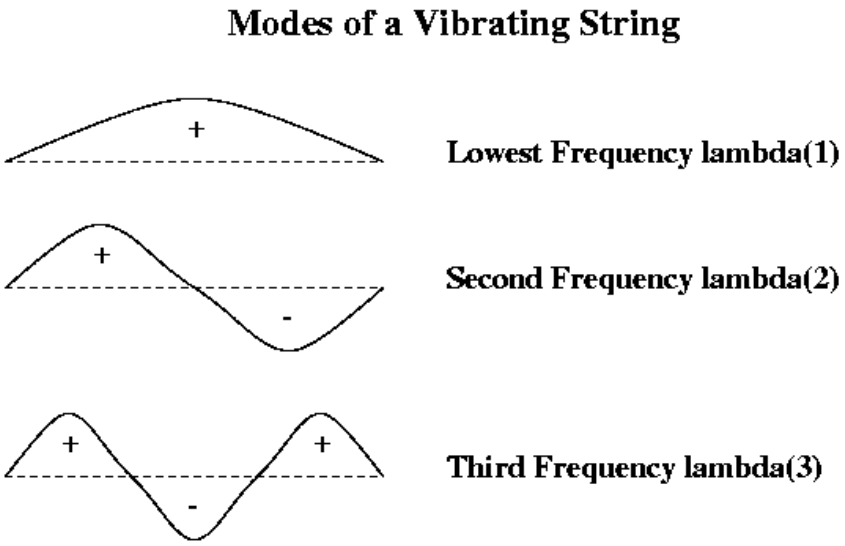


Coordinate-Free (2nd ex. method): Spectral Bisection

- Based on theory of Fiedler (1970s), popularized by Pothen, Simon, Liou (1990)
- Motivation, by analogy to a vibrating string
- Basic definitions
- Vibrating string, revisited
- 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)



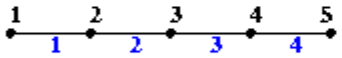
Basic Definitions

- *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. For an edge e=(i,j), 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 Simple Meshes

Incidence and Laplacian Matrices

Graph G

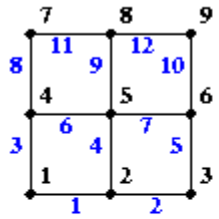


Incidence Matrix In(G)

	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 Laplacian Matrix

- *Theorem 1:* Given G, L(G) has the following properties
  (proof on 1996 CS267 web page)
  - L(G) is symmetric.
    - This means the eigenvalues of L(G) are real and its eigenvectors are real and orthogonal.
  - $In(G) * (In(G))^T = L(G)$
  - The eigenvalues of L(G) are nonnegative:
    - $0 = \lambda_1 \leq \lambda_2 \leq \ldots \leq \lambda_n$
  - The number of connected components of G is equal to the number of λ_i equal to 0.
  - *Definition*: $\lambda_2(L(G))$ is the algebraic connectivity of G
    - The magnitude of λ_2 measures connectivity
    - In particular, $\lambda_2 \neq 0$ if and only if G is connected.

Spectral Bisection Algorithm

- Spectral Bisection Algorithm:
  - Compute eigenvector v_2 corresponding to $\lambda_2(L(G))$
  - 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.
  - Recall $\lambda_2(L(G))$ 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 $\lambda_2(L(G_1)) \leq \lambda_2(L(G))$, i.e. the algebraic connectivity of G_1 is less than or equal to the algebraic connectivity of G.

Spectral Bisection Algorithm

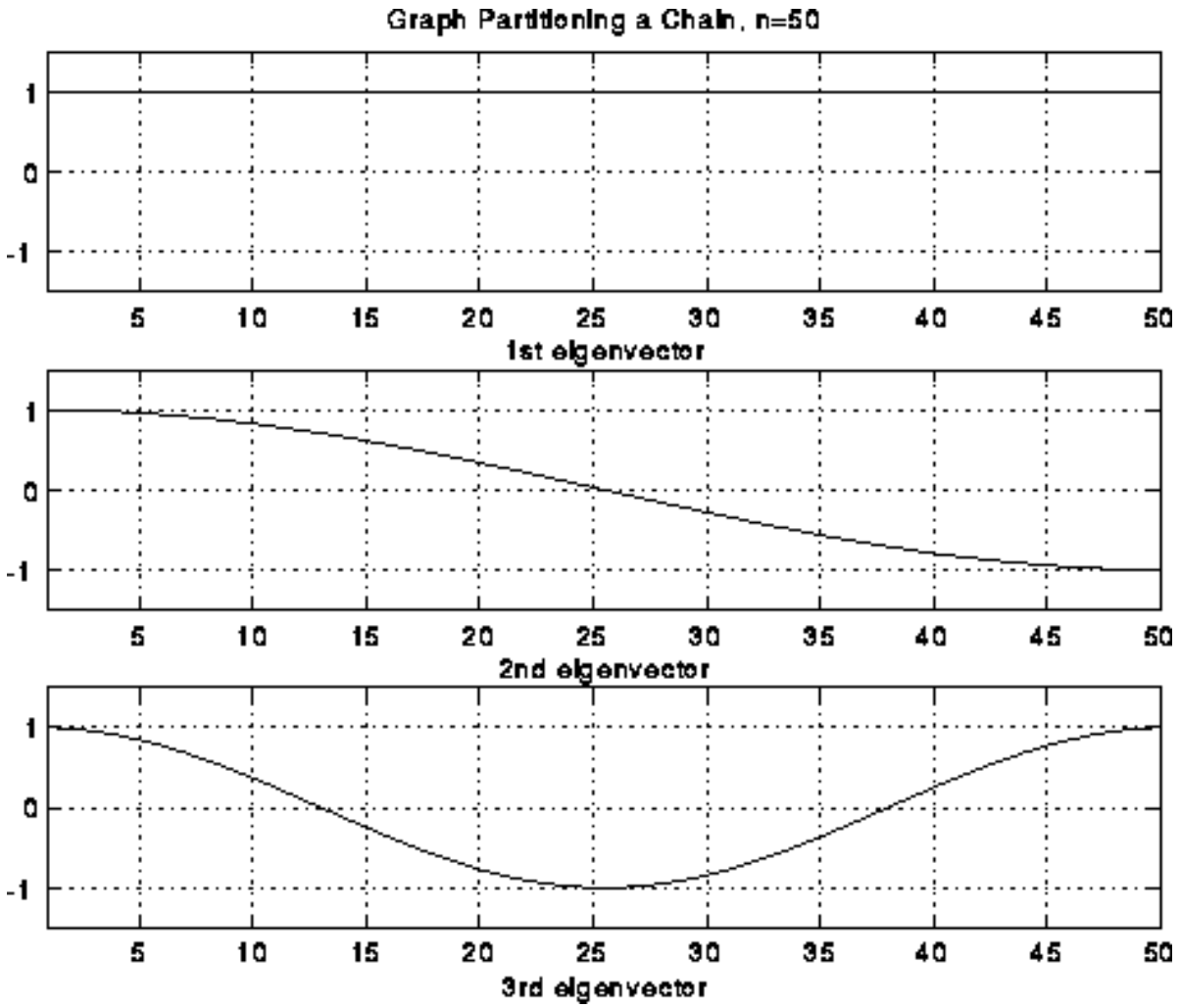
- Spectral Bisection Algorithm:
  - Compute eigenvector v_2 corresponding to $\lambda_2(L(G))$
  - For each node n of G
    - if $v_2(n) < 0$ put node n in partition N-
    - else put node n in partition N+
- Why does this make sense? More reasons...
  - *Theorem 4 (Fiedler, 1975):* Let G be connected, and N1 and N2 be any partition into part of equal size |N|/2. Then the number of edges connecting N1 and N2 is at least .25 * |N| * $\lambda_2(L(G))$.

Eigenvectors of L(1D mesh)

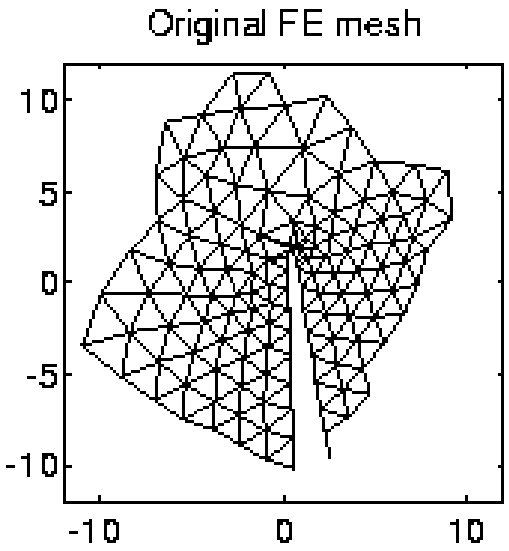
Eigenvector 1 (all ones)

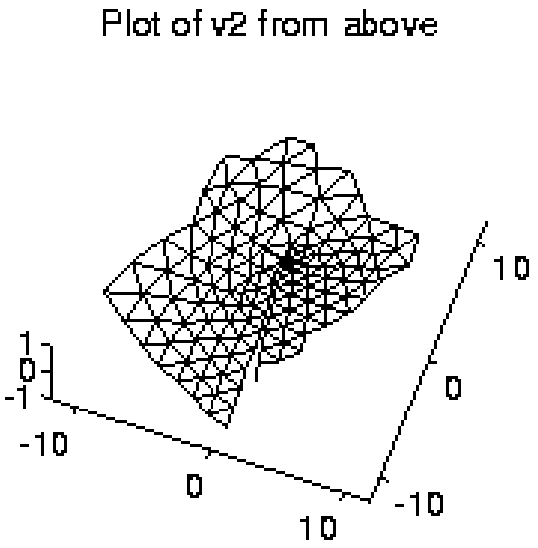
Eigenvector 2

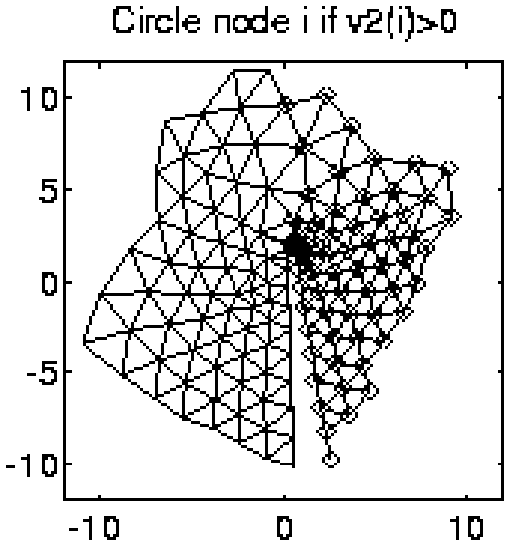
Eigenvector 3

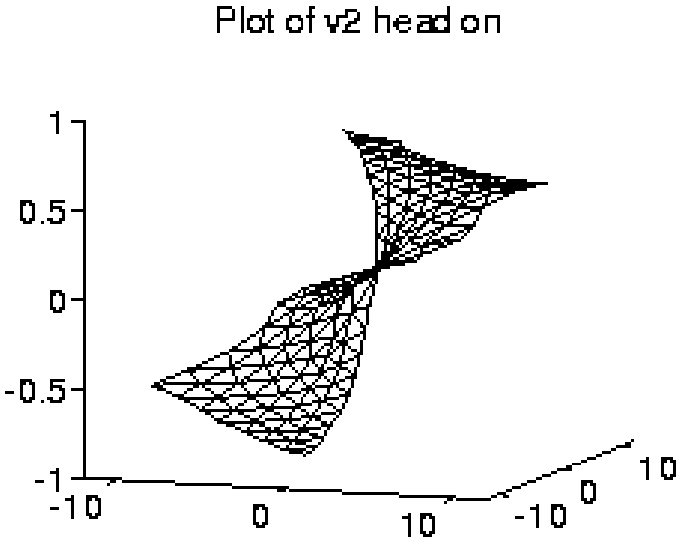


2nd eigenvector of L(planar mesh)

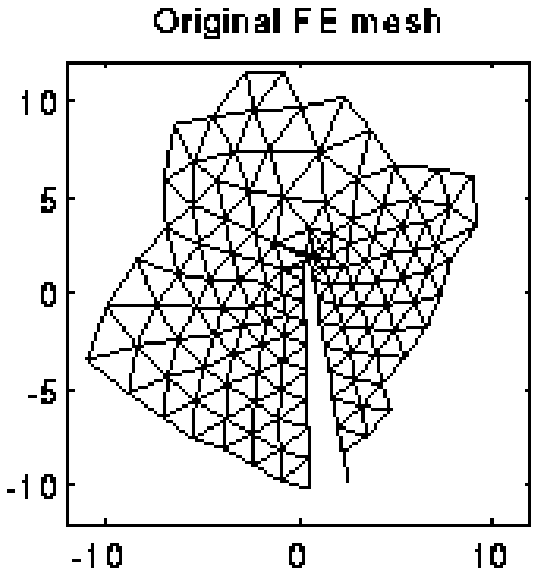


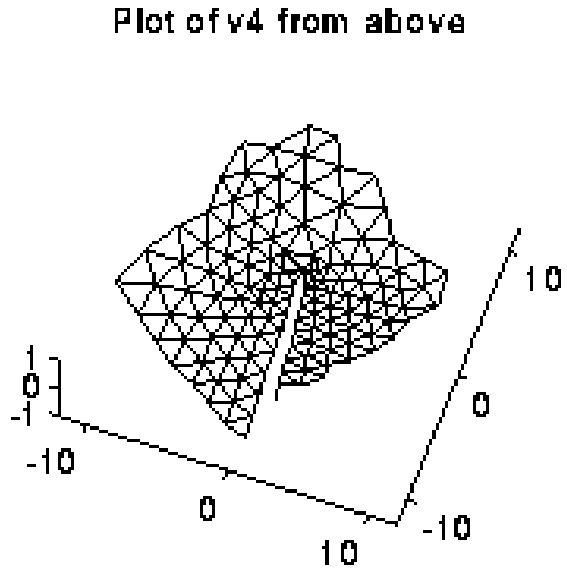


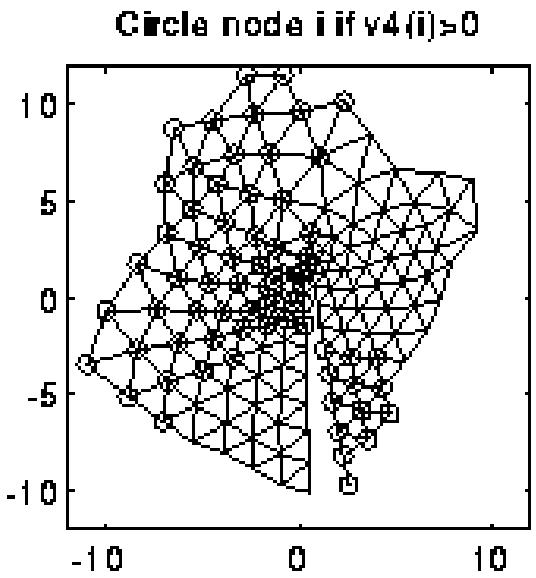


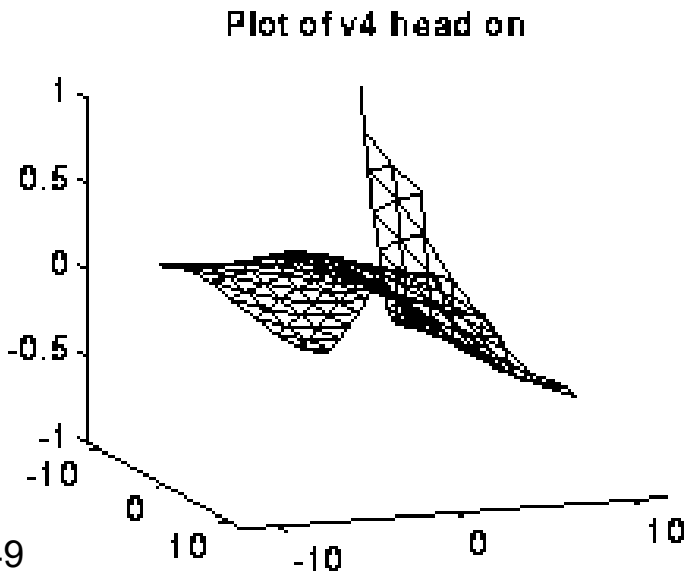


4th eigenvector of L(planar mesh)









Computing v_2 and λ_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 (BLAS1)
    r = A*q(j)               … matrix vector multiplication, the most expensive step
    r = r - b(j-1)*v(j-1)    … "axpy", or scalar*vector + vector (BLAS1)
    a(j) = v(j)^T * r        … dot product (BLAS1)
    r = r - a(j)*v(j)        … "axpy" (BLAS1)
    b(j) = ||r||             … compute vector norm (BLAS1)
until convergence            … details omitted
```

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

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

Spectral Bisection: Summary

- Laplacian matrix represents graph connectivity
- Second eigenvector gives a graph bisection
  - Roughly equal "weights" in two parts
  - Weak connection in the graph will be separator
- 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
  - Have we made progress?
    - The first matrix-vector multiplies are slow, but use them to learn how to make the rest faster

Outline of Graph Partitioning Lectures

- Review definition of Graph Partitioning problem
- Overview of heuristics
- Partitioning with Nodal Coordinates
  - Ex: In finite element models, node at point in (x,y) or (x,y,z) space
- Partitioning without Nodal Coordinates
  - Ex: In model of WWW, nodes are web pages
- **Multilevel Acceleration**
  - **BIG IDEA, appears often in scientific computing**
- Available Implementations
- Beyond Graph Partitioning: Hypergraphs

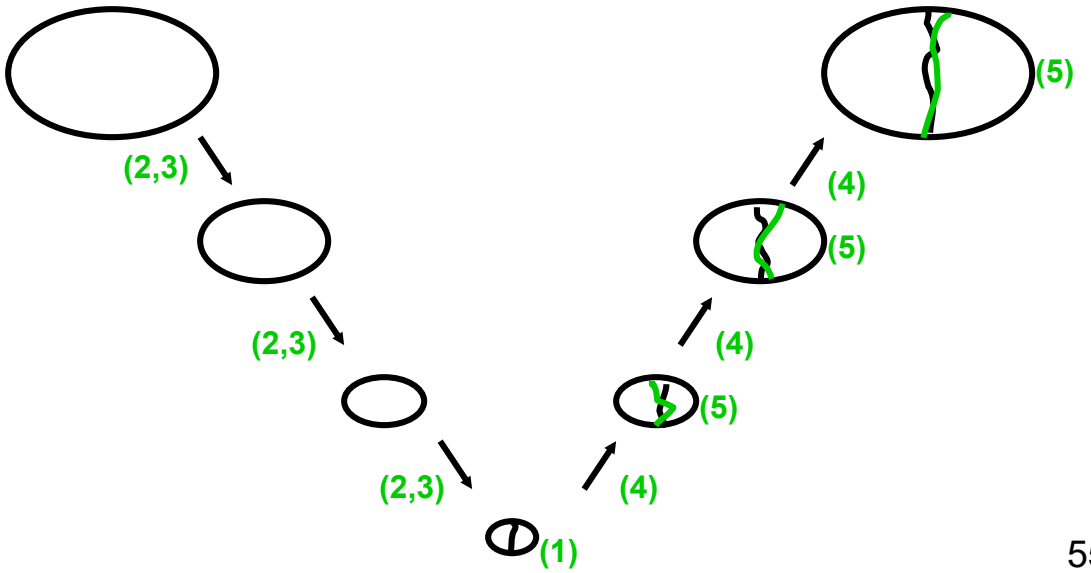
Introduction to Multilevel Partitioning

- If we want to partition G(N,E), but it is too big to do efficiently, what can we do?
  - 1) Replace G(N,E) by a coarse approximation $G_C(N_C,E_C)$, and partition G_C instead
  - 2) Use partition of G_C to get a rough partitioning of G, and then iteratively improve it
- What if G_C still too big?
  - Apply same idea recursively

Multilevel Partitioning - High Level Algorithm

```
(N+,N- ) = Multilevel_Partition( N, E )
        … recursive partitioning routine returns N+ and N- where N = N+ U N-
        if |N| is small
(1)         Partition G = (N,E)  directly to get N = N+ U N-
            Return (N+, N- )
        else
(2)         Coarsen G to get an approximation G_C = (N_C, E_C)
(3)         (N_C+ , N_C- ) = Multilevel_Partition( N_C, E_C )
(4)         Expand (N_C+ , N_C- ) to a partition  (N+ , N- ) of N
(5)         Improve the partition ( N+ , N- )
            Return ( N+ , N- )
        endif
```

"V - cycle:"



How do we

- Coarsen?
- Expand?
- Improve?

Multilevel Kernighan-Lin

- Coarsen graph and expand partition using maximal matchings
- Improve partition using Kernighan-Lin

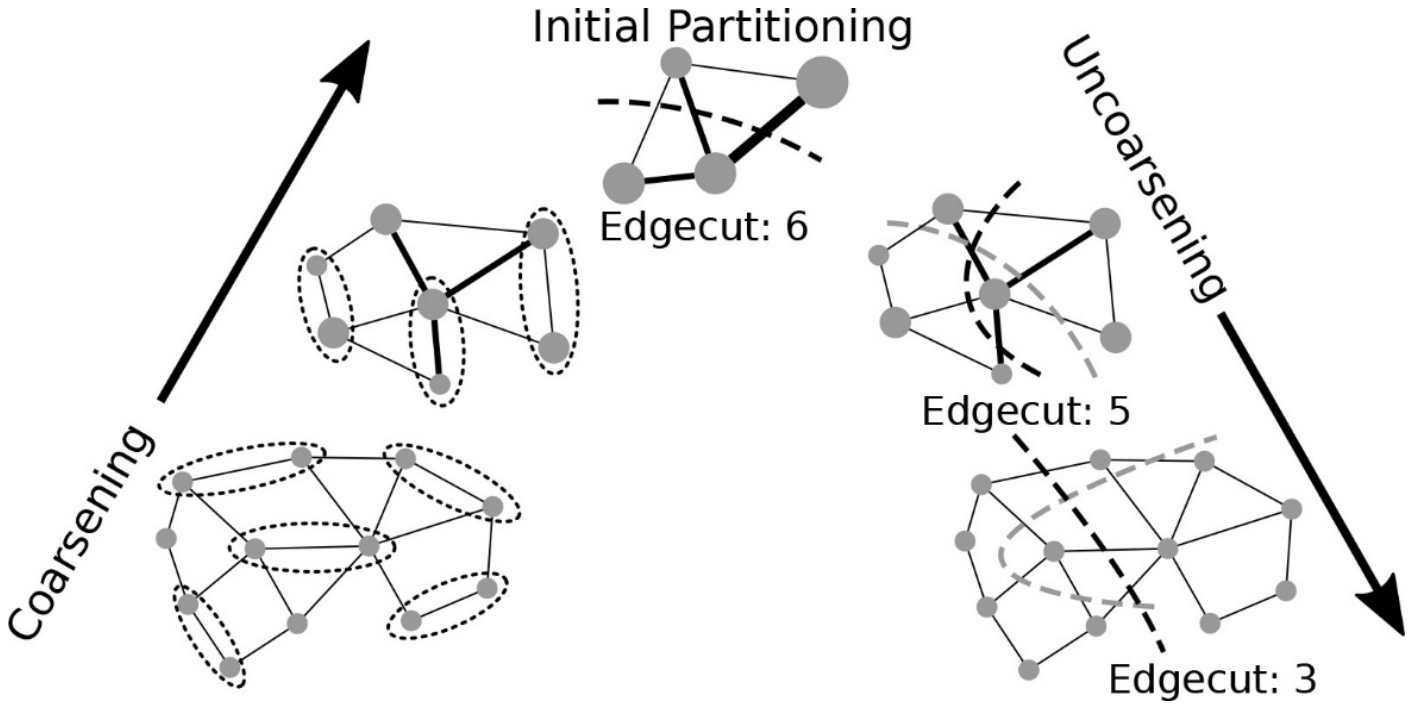


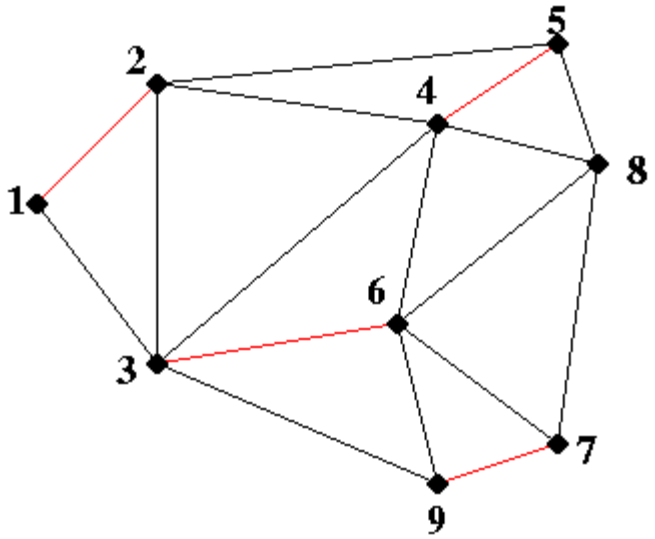
Figure from Dominique LaSalle et al., A Parallel Hill-Climbing Refinement Algorithm for Graph Partitioning, Proc. ICPP 2016.

Maximal Matching

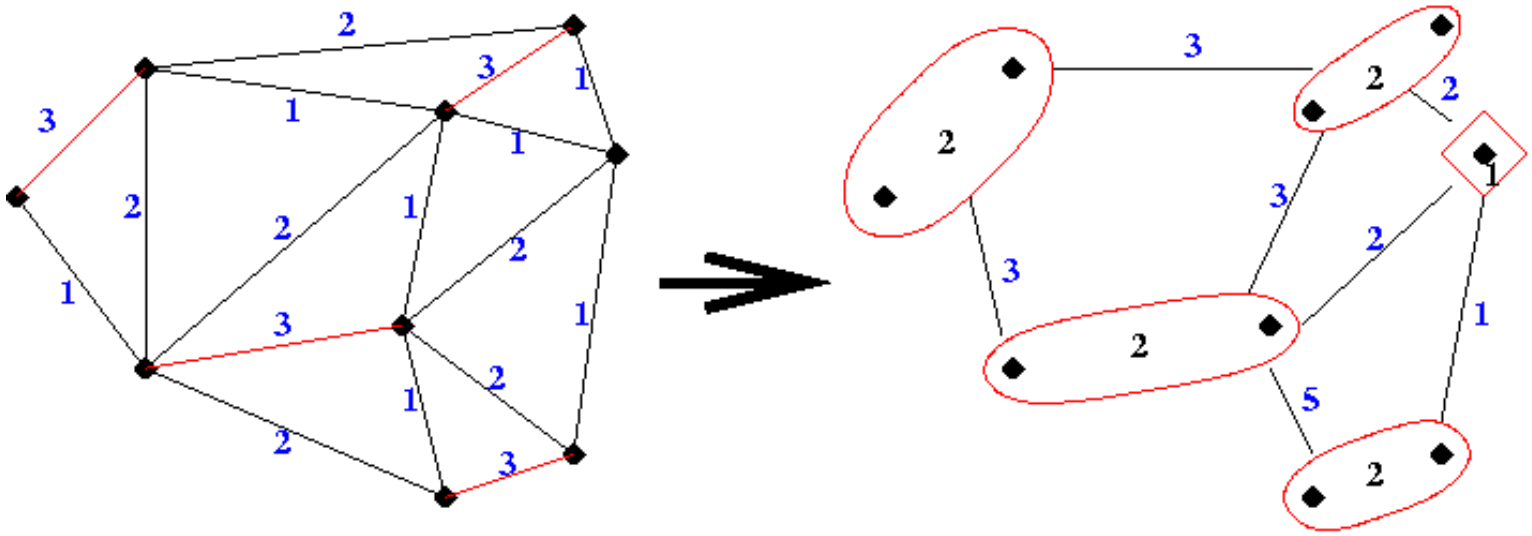
- *Definition*: A matching of a graph G(N,E) is a subset E_m of E such that no two edges in E_m share an endpoint
- *Definition:* A maximal matching of a graph G(N,E) is a matching E_m to which no more edges can be added and remain a matching
- A simple greedy algorithm computes a maximal matching:

```
let Em be empty
mark all nodes in N as unmatched
for i = 1 to |N|      … visit the nodes in any order
    if i has not been matched
        mark i as matched
        if there is an edge e=(i,j)  where j is also unmatched,
            add e to Em
            mark j as matched
        endif
    endif
endfor
```

Maximal Matching: Example



Example of Coarsening



$G = (N, E)$

E_m is shown in red

Edge weights shown in blue

Node weights are all one

$G_c = (N_c, E_c)$

N_c is shown in red

Edge weights shown in blue

Node weights shown in black

Coarsening using a maximal matching (details)

1) Construct a maximal matching E_m of G(N,E)

for all edges e=(j,k) in E_m 2) collapse matched nodes into a single one

  Put node n(e) in N_c

  W(n(e)) = W(j) + W(k) … gray statements update node/edge weights

for all nodes n in N not incident on an edge in E_m 3) add unmatched nodes

  Put n in N_c … do not change W(n)

… Now each node r in N is "inside" a unique node n(r) in N_c

… 4) Connect two nodes in Nc if nodes inside them are connected in E

for all edges e=(j,k) in E_m

  for each other edge e' =(j,r) or (k,r) in E

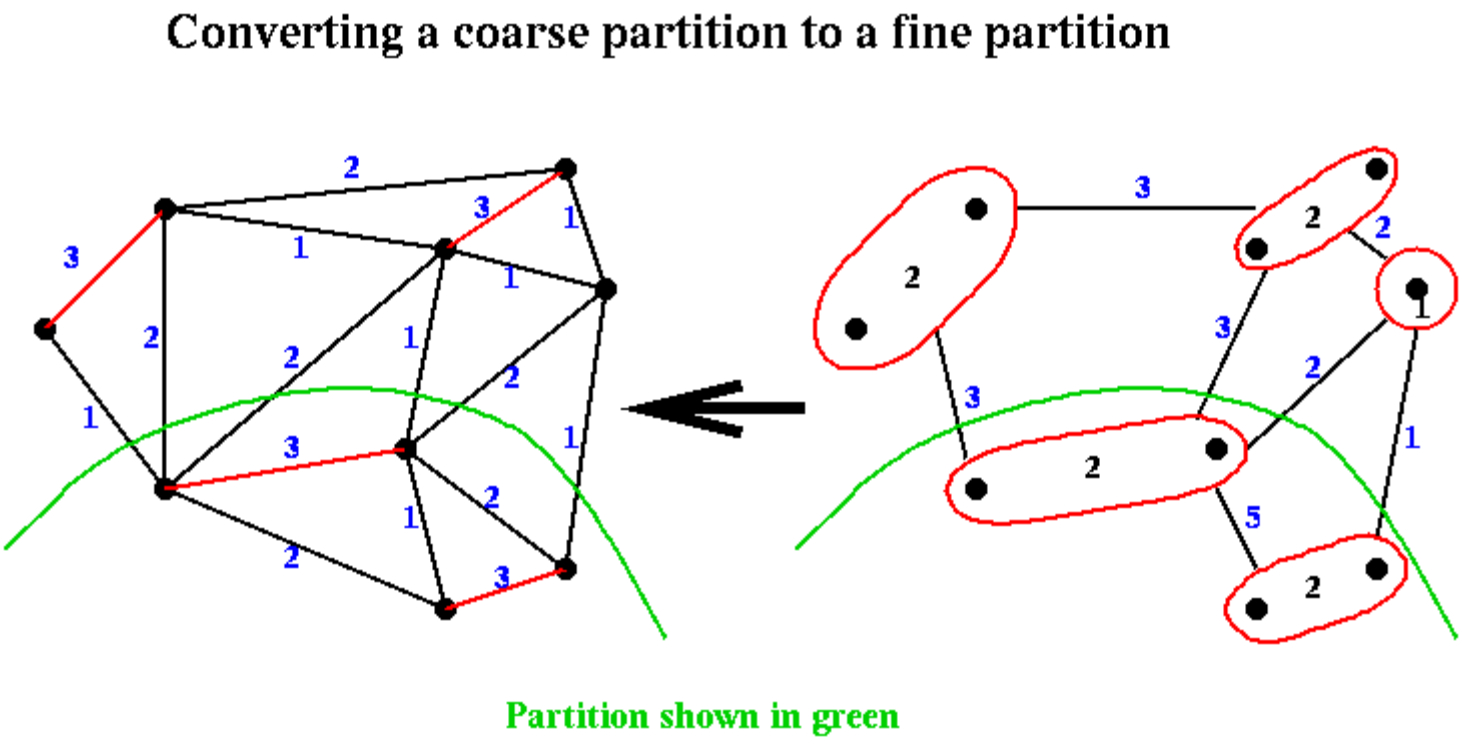
    Put edge ee = (n(e),n(r)) in E_c

    W(ee) = W(e')

If there are multiple edges connecting two nodes in N_c, collapse them,

  adding edge weights

Expanding a partition of G_c to a partition of G



Multilevel Spectral Bisection

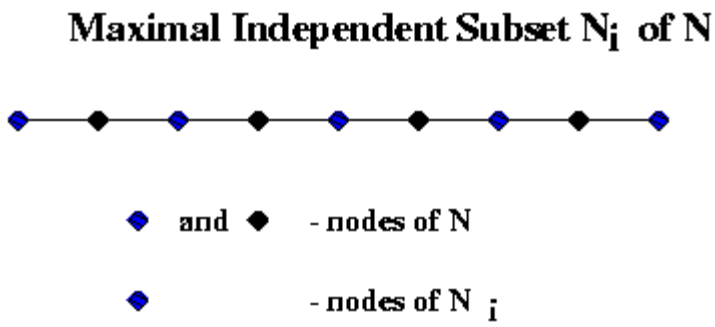
- Coarsen graph and expand partition using maximal independent sets
- Improve partition using Rayleigh Quotient Iteration

(We skip Multilevel Spectral Bisection in this course)

Maximal Independent Sets (another way to define coarsening)

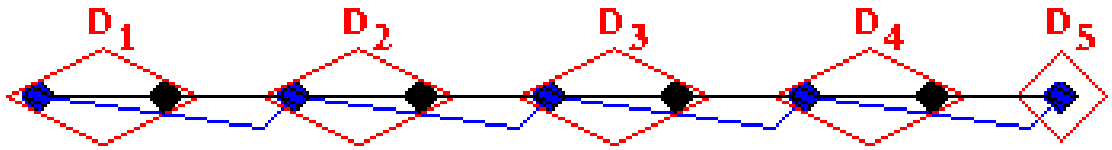
- *Definition*: An independent set of a graph G(N,E) is a subset N_i of N such that no two nodes in N_i are connected by an edge
- *Definition:* A maximal independent set of a graph G(N,E) is an independent set N_i to which no more nodes can be added and remain an independent set
- A simple greedy algorithm computes a maximal independent set:

```
let N_i be empty
for k = 1 to |N|      … visit the nodes in any order
    if  node k is not adjacent to any node already in N_i
        add k to N_i
    endif
endfor
```



Example of Coarsening

Computing G_c from G



- and - nodes of N
- nodes of N_i
- edges in E
- edges in E_c
- encloses domain D_k = node of N_c

Coarsening using Maximal Independent Sets (details)

… Build "domains" D(k) around each node k in N_i to get nodes in N_C
… Add an edge to E_C whenever it would connect two such domains
E_C = empty set
for all nodes k in N_i
    D(k) = ({k}, empty set)
    … first set contains nodes in D(k), second set contains edges in D(k)
unmark all edges in E
repeat
    choose an unmarked edge e = (k,j) from E
    if exactly one of k and j (say k) is in some D(m)
        mark e
        add j and e to D(m)
    else if k and j are in two different D(m)'s (say D(mk) and D(mj))
        mark e
        add edge (mk, mj) to E_C
    else if both k and j are in the same D(m)
        mark e
        add e to D(m)
    else
        leave e unmarked
    endif
until no unmarked edges

Expanding a partition of G_c to a partition of G

- Need to convert an eigenvector v_c of $L(G_c)$ to an approximate eigenvector v of L(G)
- Use interpolation:

For each node j in N
  if j is also a node in N_c, then
    $v(j) = v_c(j)$ … use same eigenvector component
  else
    v(j) = average of $v_c(k)$ for all neighbors k of j in N_c
  end if
endif

Improve eigenvector: Rayleigh Quotient Iteration

j = 0

pick starting vector v(0) … from expanding v_c

repeat

j=j+1

$r(j) = v^T(j-1) * L(G) * v(j-1)$

… r(j) = *Rayleigh Quotient* of v(j-1)

… = good approximate eigenvalue

$v(j) = (L(G) - r(j)*I)^{-1} * v(j-1)$

… expensive to do exactly, so solve approximately

… using an iteration called SYMMLQ,

… which uses matrix-vector multiply (no surprise)

v(j) = v(j) / || v(j) || … normalize v(j)

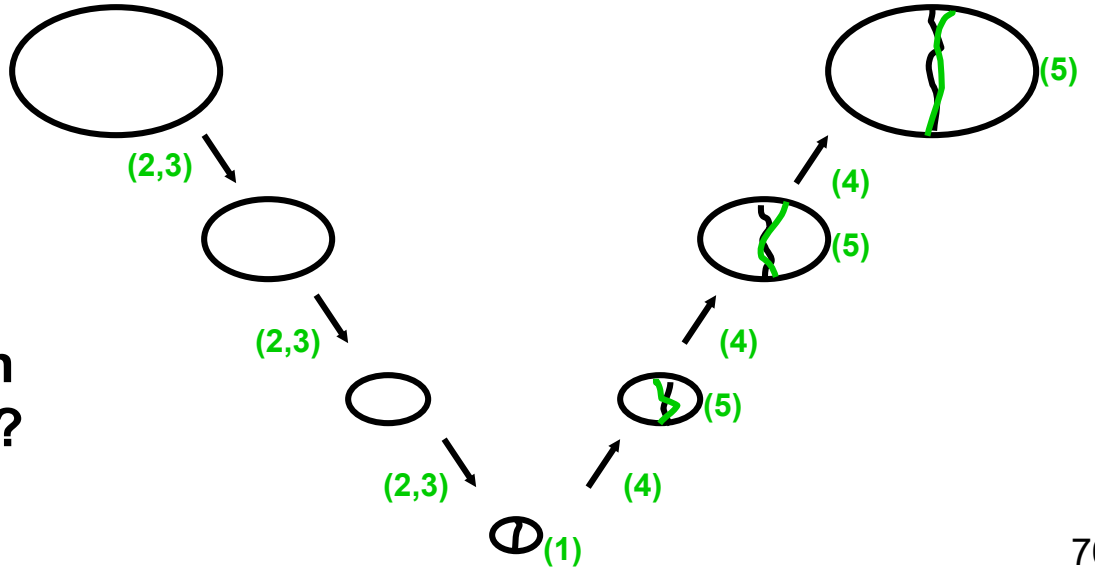
until v(j) converges

… Convergence is very fast: cubic

Multilevel Partitioning - High Level Algorithm

```
(N+,N- ) = Multilevel_Partition( N, E )
    … recursive partitioning routine returns N+ and N- where N = N+ U N-
    if |N| is small
(1)     Partition G = (N,E)  directly to get N = N+ U N-
        Return (N+, N- )
    else
(2)     Coarsen G to get an approximation G_C = (N_C, E_C)
(3)     (N_C+ , N_C- ) = Multilevel_Partition( N_C, E_C )
(4)     Expand (N_C+ , N_C- ) to a partition  (N+ , N- ) of N
(5)     Improve the partition ( N+ , N- )
        Return ( N+ , N- )
    endif
```

"V - cycle:"



When to stop with coarsening?
How many nodes in the coarsest graph?

Outline of Graph Partitioning Lectures

- Review definition of Graph Partitioning problem
- Overview of heuristics
- Partitioning with Nodal Coordinates
  - Ex: In finite element models, node at point in (x,y) or (x,y,z) space
- Partitioning without Nodal Coordinates
  - Ex: In model of WWW, nodes are web pages
- Multilevel Acceleration
  - BIG IDEA, appears often in scientific computing
- **Comparison of Methods and Applications**
- Beyond Graph Partitioning: Hypergraphs

Available Implementations

- **Multilevel Kernighan/Lin**
  - **METIS and ParMETIS (www.cs.umn.edu/~metis)**
  - **SCOTCH (www.labri.fr/perso/pelegrin/scotch/)**
- **Multilevel Spectral Bisection**
  - **S. Barnard and H. Simon, "A fast multilevel implementation of recursive spectral bisection …", Proc. 6th SIAM Conf. On Parallel Processing, 1993**
  - **Chaco (www.cs.sandia.gov/~bahendr/chaco.html)**
- **Hybrids possible**
  - **Ex: Using Kernighan/Lin to improve a partition from spectral bisection**
- **Recent package, collection of techniques**
  - **Zoltan (www.cs.sandia.gov/Zoltan)**
- **See www.cs.sandia.gov/~bahendr/partitioning.html**

Comparison of methods

- Compare only methods that use edges, not nodal coordinates
  - CS267 webpage and KK95a (see below) have other comparisons
- Metrics
  - Speed of partitioning
  - Number of edge cuts
  - Other application dependent metrics
- Summary
  - No one method best
  - Multi-level Kernighan/Lin fastest by far, comparable to Spectral in the number of edge cuts
    - www-users.cs.umn.edu/~karypis/metis/publications/main.html
    - see publications KK95a and KK95b and D. LaSalle et al., A Parallel Hill-Climbing Refinement Algorithm for Graph Partitioning, Proc. ICPP 2016.
  - Spectral gives better cuts for some applications
    - Ex: image segmentation
    - See “Normalized Cuts and Image Segmentation” by J. Malik, J. Shi

Number of edges cut for a 64-way partition

For Multilevel Kernighan/Lin, as implemented in METIS (see KK95a)

Graph	# of Nodes	# of Edges	# Edges cut for 64-way partition	Expected # cuts for 2D mesh	Expected # cuts for 3D mesh	Description
144	144649	1074393	88806	6427	31805	3D FE Mesh
4ELT	15606	45878	2965	2111	7208	2D FE Mesh
ADD32	4960	9462	675	1190	3357	32 bit adder
AUTO	448695	3314611	194436	11320	67647	3D FE Mesh
BBMAT	38744	993481	55753	3326	13215	2D Stiffness M.
FINAN512	74752	261120	11388	4620	20481	Lin. Prog.
LHR10	10672	209093	58784	1746	5595	Chem. Eng.
MAP1	267241	334931	1388	8736	47887	Highway Net.
MEMPLUS	17758	54196	17894	2252	7856	Memory circuit
SHYY161	76480	152002	4365	4674	20796	Navier-Stokes
TORSO	201142	1479989	117997	7579	39623	3D FE Mesh

Expected # cuts for 64-way partition of 2D mesh of n nodes

$$n^{1/2} + 2*(n/2)^{1/2} + 4*(n/4)^{1/2} + \ldots + 32*(n/32)^{1/2} \sim 17 * n^{1/2}$$

Expected # cuts for 64-way partition of 3D mesh of n nodes =

$$n^{2/3} + 2*(n/2)^{2/3} + 4*(n/4)^{2/3} + \ldots + 32*(n/32)^{2/3} \sim 11.5 * n^{2/3}$$

Speed of 256-way partitioning (from KK95a)

Partitioning time in seconds

Graph	# of Nodes	# of Edges	Multilevel Spectral Bisection	Multilevel Kernighan/ Lin	Description
144	144649	1074393	607.3	48.1	3D FE Mesh
4ELT	15606	45878	25.0	3.1	2D FE Mesh
ADD32	4960	9462	18.7	1.6	32 bit adder
AUTO	448695	3314611	2214.2	179.2	3D FE Mesh
BBMAT	38744	993481	474.2	25.5	2D Stiffness M.
FINAN512	74752	261120	311.0	18.0	Lin. Prog.
LHR10	10672	209093	142.6	8.1	Chem. Eng.
MAP1	267241	334931	850.2	44.8	Highway Net.
MEMPLUS	17758	54196	117.9	4.3	Memory circuit
SHYY161	76480	152002	130.0	10.1	Navier-Stokes
TORSO	201142	1479989	1053.4	63.9	3D FE Mesh

Kernighan/Lin much faster than Spectral Bisection!

Edgecut and serial runtimes for 64-way partitioning

	Greedy		RB-FM		KPM		MTFM		HS	
Graph	Edgecut	Time (s)	Edgecut	Time (s)	Edgecut	Time (s)	Edgecut	Time (s)	Edgecut	Time (s)
t60k	2,565	0.085	2,433	0.202	**2,378**	0.481	2,445	0.108	2,401	0.109
wing	9,727	0.146	9,074	0.309	8,783	1.351	8,772	0.248	**8,592**	0.220
fe_pwt	9,451	0.091	9,124	0.204	8,776	0.485	8,929	0.132	**8,775**	0.126
fe_body	5,710	0.079	**5,236**	0.221	5,289	0.429	5,458	0.097	5,352	0.096
vibrobox	54,046	0.205	54,799	0.293	53,405	0.994	53,028	0.232	**52,835**	0.247
finan512	11,500	0.148	**10,710**	0.395	11,388	0.803	11,632	0.297	11,350	0.183
bcsstk33	116,821	0.237	117,623	0.280	114,427	0.725	**114,168**	0.257	114,322	0.273
bcsstk29	63,929	0.127	62,348	0.266	**61,149**	0.485	63,432	0.138	62,413	0.148
brack2	29,805	0.150	28,721	0.408	**28,104**	1.113	28,555	0.219	28,414	0.217
fe_ocean	27,312	0.198	23,011	0.586	**22,826**	2.032	23,385	0.364	22,896	0.349
fe_tooth	39,987	0.169	39,009	0.484	38,219	1.365	38,261	0.273	**38,032**	0.265
bcsstk31	67,391	0.168	65,103	0.386	**63,852**	0.953	65,470	0.215	64,568	0.225
fe_rotor	52,616	0.199	51,553	0.667	50,279	1.953	50,815	0.336	**50,251**	0.326
598a	63,413	0.225	63,346	0.745	**60,299**	2.303	60,756	0.401	60,440	0.370
bcsstk32	107,911	0.170	102,943	0.535	**102,382**	1.021	104,614	0.215	103,550	0.220
bcsstk30	190,499	0.193	187,906	0.546	**183,650**	0.968	186,719	0.237	185,576	0.257
wave	95,460	0.274	95,142	0.952	91,778	2.859	90,803	0.522	**90,515**	0.485
144	88,039	0.264	87,836	0.964	84,254	2.827	84,245	0.490	**83,643**	0.455
m14b	109,570	0.335	108,677	1.411	103,996	3.484	104,563	0.625	**103,878**	0.576
auto	191,400	0.681	191,933	2.841	183,624	6.652	181,215	1.355	**180,367**	1.203
AS365	54,767	3.154	52,993	15.519	51,943	15.984	50,740	3.822	**50,356**	3.669
NLR	60,287	3.538	58,430	17.249	57,437	17.553	55,775	4.319	**55,378**	4.134
adaptive	48,634	4.452	44,364	20.789	44,149	24.593	42,651	5.911	**42,344**	5.330
ldoor	439,153	1.624	420,011	9.834	**414,884**	5.846	421,377	1.771	415,463	1.816
Serena	1,852,915	3.140	1,869,282	15.720	1,813,903	31.495	1,762,200	5.079	**1,760,964**	4.828
audikw1	2,945,167	3.269	2,976,115	17.174	2,838,997	23.877	**2,830,537**	4.941	2,835,015	4.951
channel-500x.	1,356,670	6.633	1,274,048	31.187	1,305,570	64.801	1,274,548	12.781	**1,260,250**	11.258
dielFilterV3.	2,442,877	3.652	2,432,023	20.173	**2,321,037**	26.516	2,335,146	5.272	2,321,886	5.054
Flan_1565	2,463,890	4.376	2,392,417	25.932	**2,317,798**	19.220	2,344,077	5.412	2,335,178	5.846
nlpkkt240	10,303,386	65.096	10,326,787	297.996	10,171,102	156.130	**9,751,898**	96.392	9,763,379	108.195
Geo. Mean	105760.6	0.540	102440.1	1.795	100294.6	3.457	100542.3	0.791	**99634.8**	0.764

The five refinement algorithms: Greedy, Parallel Recursive Bisection FM (RB-FM), k-way Pairwise FM (KPM), Multi-Try FM (MTFM), and Hill-Scanning (HS), run on the graphs from the Graph Partitioning Archive [26] in the top section, and the University of Florida Sparse Matrix Collection [6] in the bottom section. The lowest mean edgecut achieved per graph is highlighted in bold.

Source: Dominique LaSalle et al., A Parallel Hill-Climbing Refinement Algorithm for Graph Partitioning, Proc. ICPP 2016.