# CS 267
# Lecture 18: Graph Partitioning

**Aydin Buluc**

[slides from James Demmel and John Gilbert]

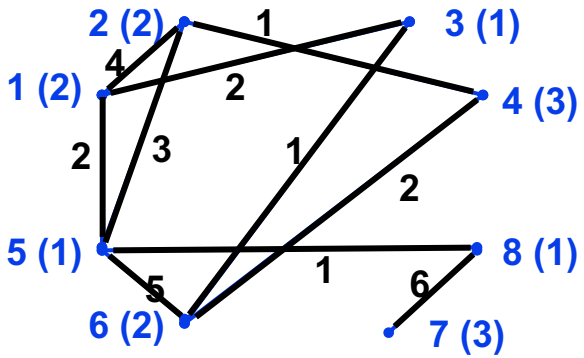
https://sites.google.com/lbl.gov/cs267-spr2019/

# 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
- Available Implementations
- Beyond Graph Partitioning: Hypergraphs

# Definition of Graph Partitioning

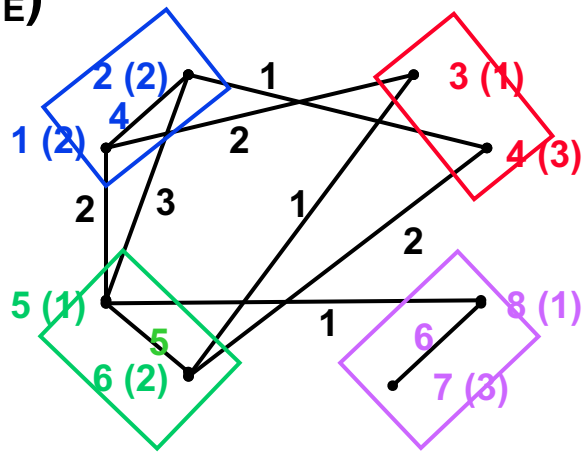
- Given a graph G = (N, E, $W_N$, $W_E$)
  - N = nodes (or vertices),
  - $W_N$ = node weights
  - E = edges
  - $W_E$ = edge weights




- 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



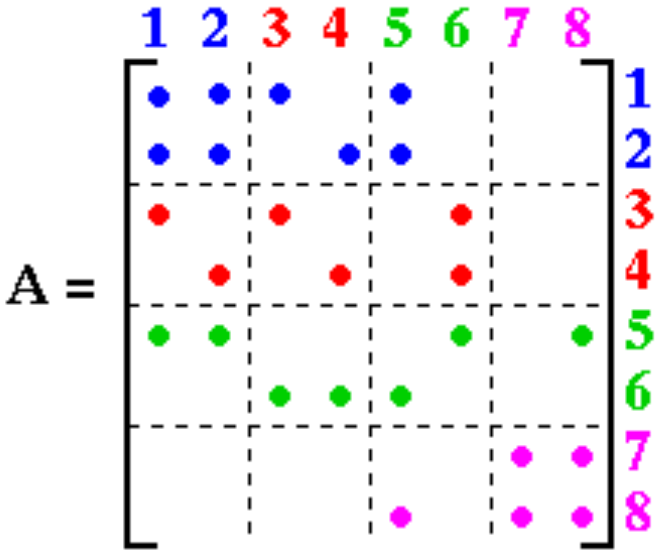

- Ex: N = {tasks}, $W_N$ = {task costs}, edge (j,k) in E means task j sends $W_E$(j,k) words to task k
- Choose a partition N = $N_1$ U $N_2$ U … U $N_P$ such that
  - The sum of the node weights in each $N_j$ is "about the same"
  - The sum of all edge weights of edges connecting all different pairs $N_j$ 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

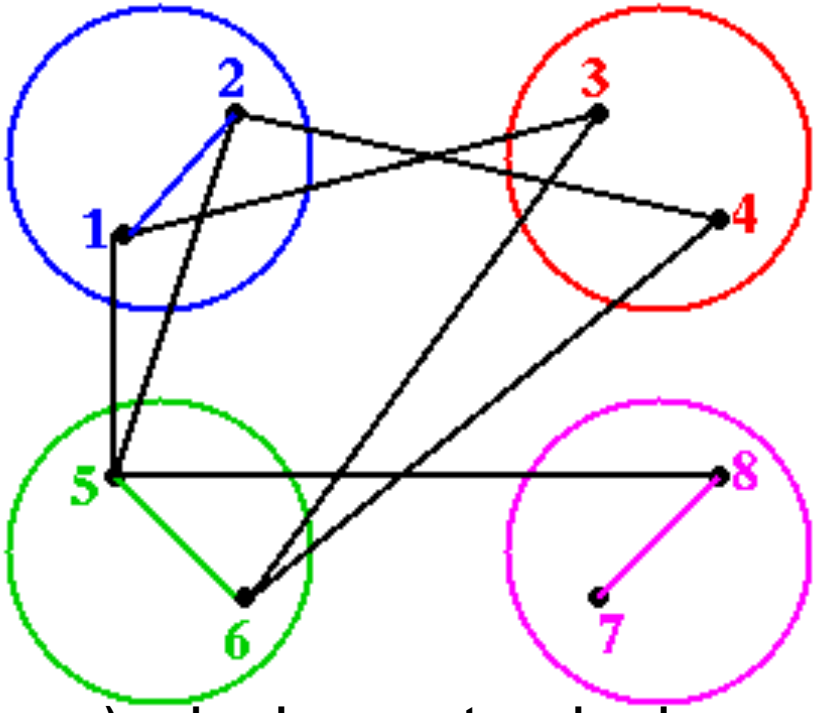
# Some Applications

- 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



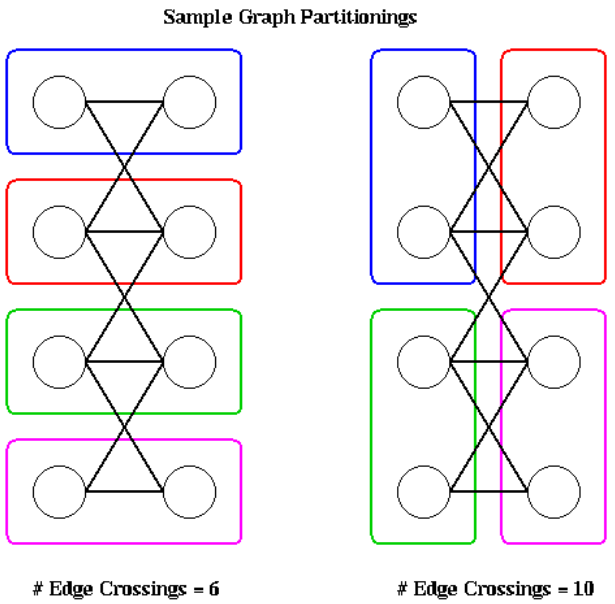

```
… declare A_local, A_remote(1:num_procs), x_local, x_remote, y_local
y_local = y_local + A_local * x_local
for all procs P that need part of x_local
        send(needed part of x_local, P)
for all procs P owning needed part of x_remote
        receive(x_remote, P)
        y_local = y_local + A_remote(P)*x_remote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

# 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



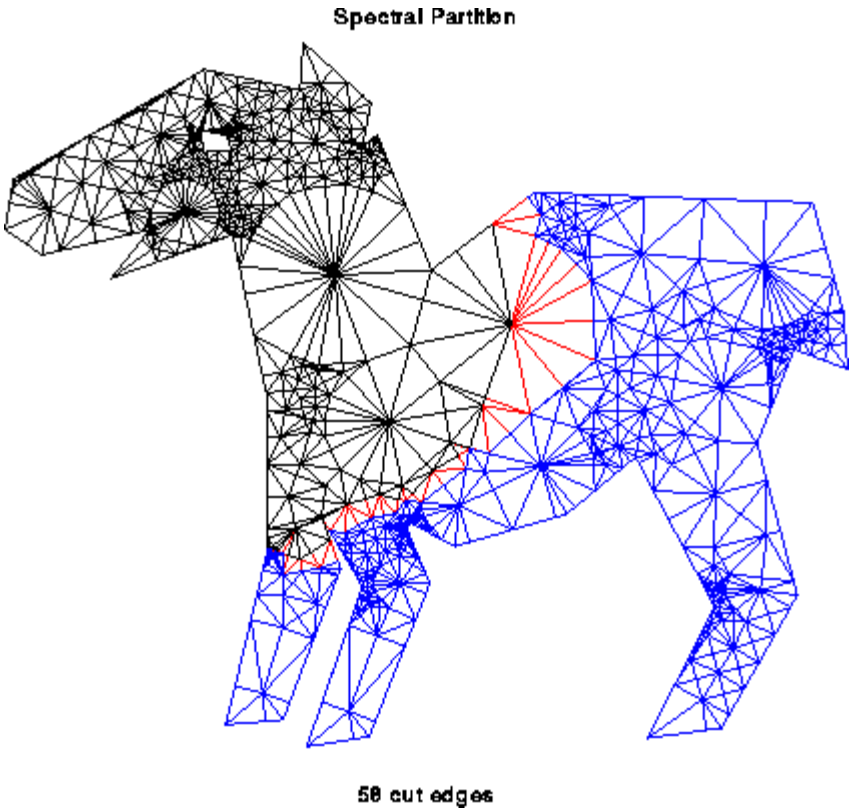

- Choosing optimal partitioning is NP-complete
  - (NP-complete = we can prove it is a hard as other well-known hard problems in a class Nondeterministic Polynomial time)
  - Only known exact algorithms have cost = exponential(n)
- We need good heuristics

# 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**
- **Available Implementations**
- **Beyond Graph Partitioning: Hypergraphs**

# First Heuristic: Repeated Graph Bisection

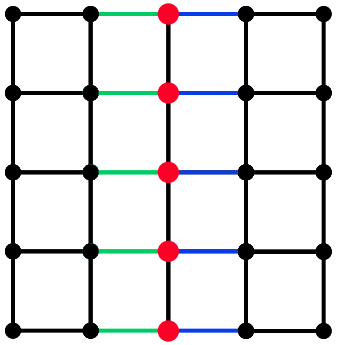
- To partition N into $2^k$ parts
  - bisect graph recursively k times
- Henceforth discuss mostly graph bisection



# 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 $N_2$

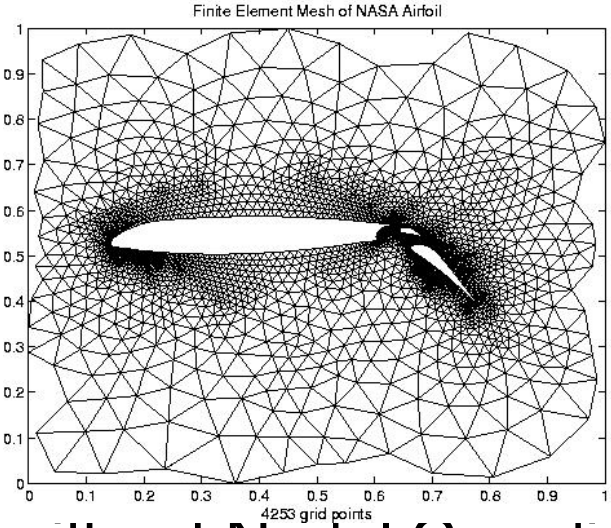
**G = (N, E), Nodes N and Edges E**
**$E_s$ = green edges or blue edges**
**$N_s$ = red vertices**



- 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

# 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**
- **Available Implementations**
- **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: ∃ 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, ∃ $N_s$ 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
- Choose a line L, and then choose a line $L^{\perp}$ perpendicular to it, with half the nodes on either side

1. **Choose 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 nj = (xj,yj), compute coordinate**

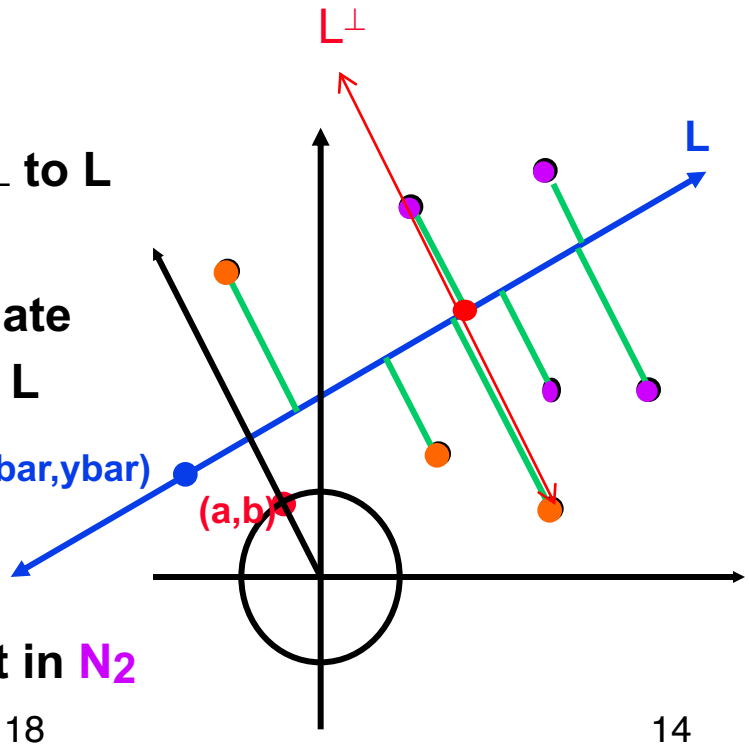
   **$S_j$ = -b*($x_j$-xbar) + a*($y_j$-ybar) along L**

3. **Compute the median**

   **Let Sbar = median($S_1$,…,$S_n$)**

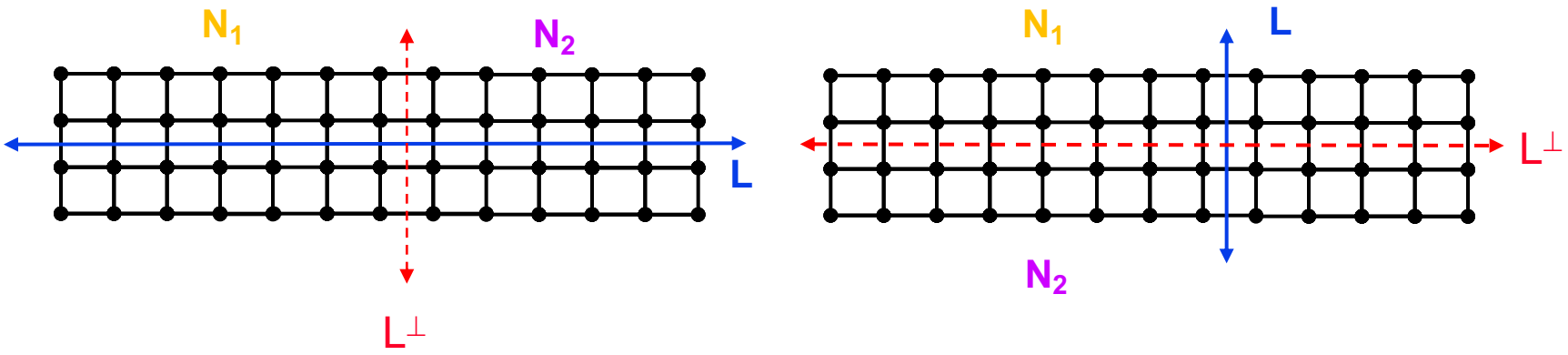
4. **Use median to partition the nodes**

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



# Inertial Partitioning: Choosing L

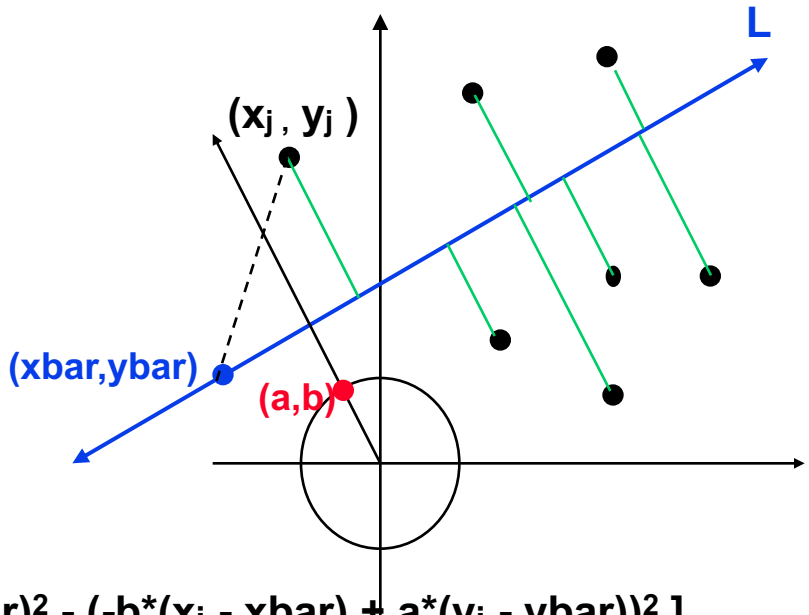
- Clearly prefer L, $L^{\perp}$ on left below




- Mathematically, choose L to be 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: choosing L (continued)

(a,b) is unit vector perpendicular to L




$\Sigma_j$ (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

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

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

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

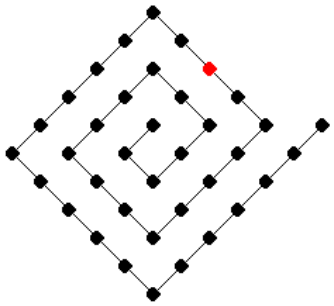
Minimized by choosing

(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
- 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 partitioners that do examine edges
- Can do poorly if graph connectivity is not spatial:



- Details at
  - www.cs.berkeley.edu/~demmel/cs267/lecture18/lec
  - www.cs.ucsb.edu/~gilbert
  - www-bcf.usc.edu/~shanghua/

# 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**

# Coordinate-Free: 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
- Given G = (N,E,$W_E$) and a partitioning N = A U 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) U Y  and  newB = (B – Y) U 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 = $\Sigma$ {W(a,b) for b in B}
  - I(a) = internal cost of a in A = $\Sigma$ {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
  - 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

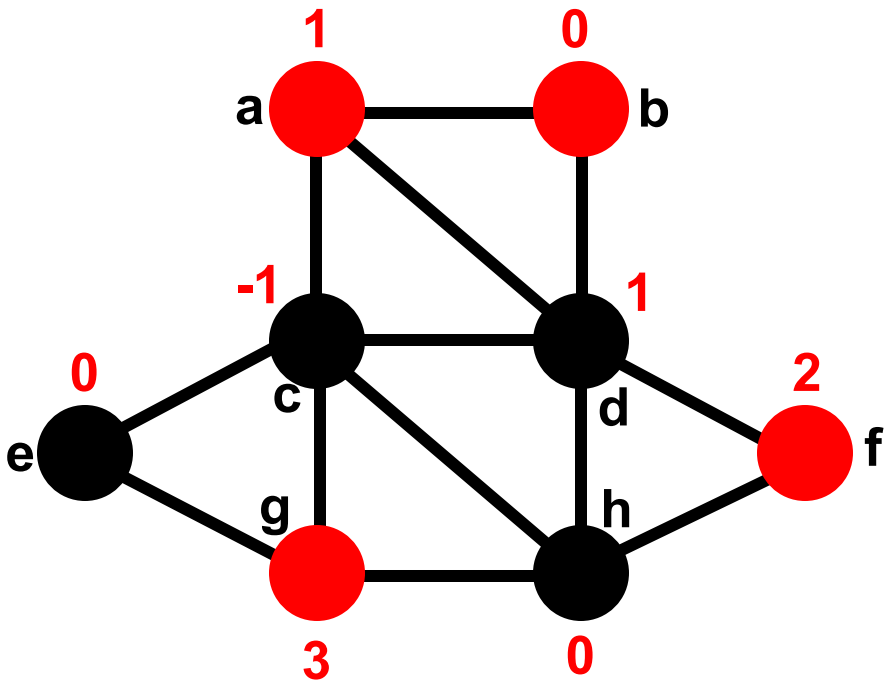
```
Compute  T = cost(A,B) for initial A, B                    … cost = O(|N|^2)
Repeat
    … One pass greedily computes |N|/2 possible X,Y to swap, picks best
    Compute costs D(n) for all n in N                      … cost = O(|N|^2)
    Unmark all nodes in N                                  … cost = 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 = Σk=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
```

# 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 are shown in red.



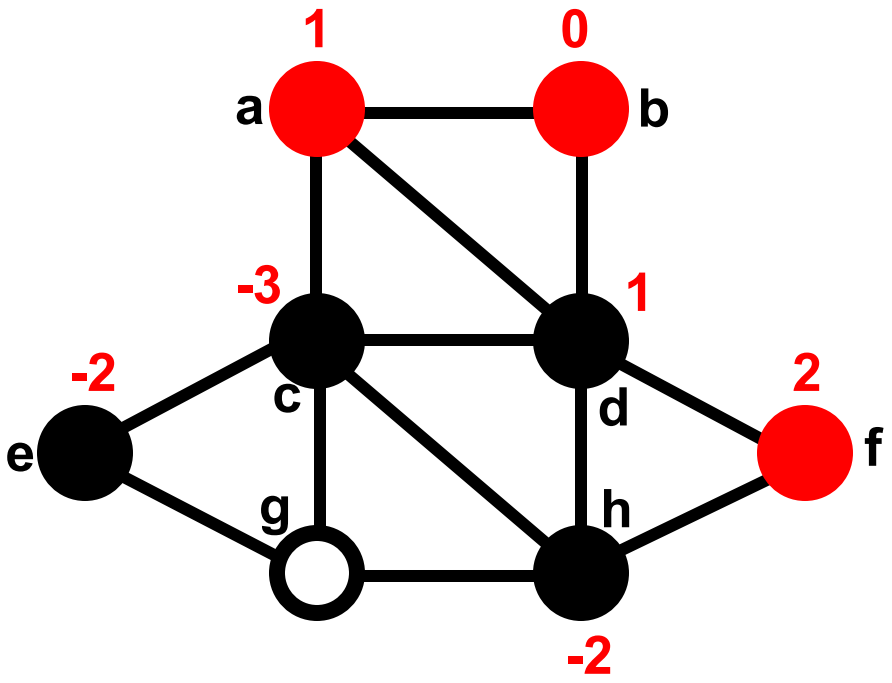

Nodes tentatively moved (and cut size after each pair):

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.



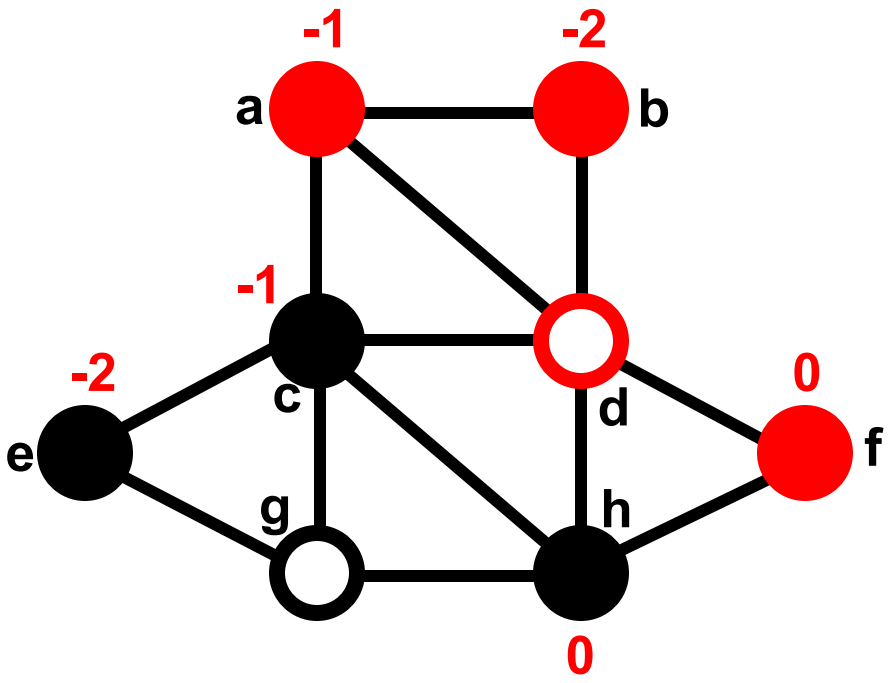

Nodes tentatively moved (and cut size after each pair):

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.

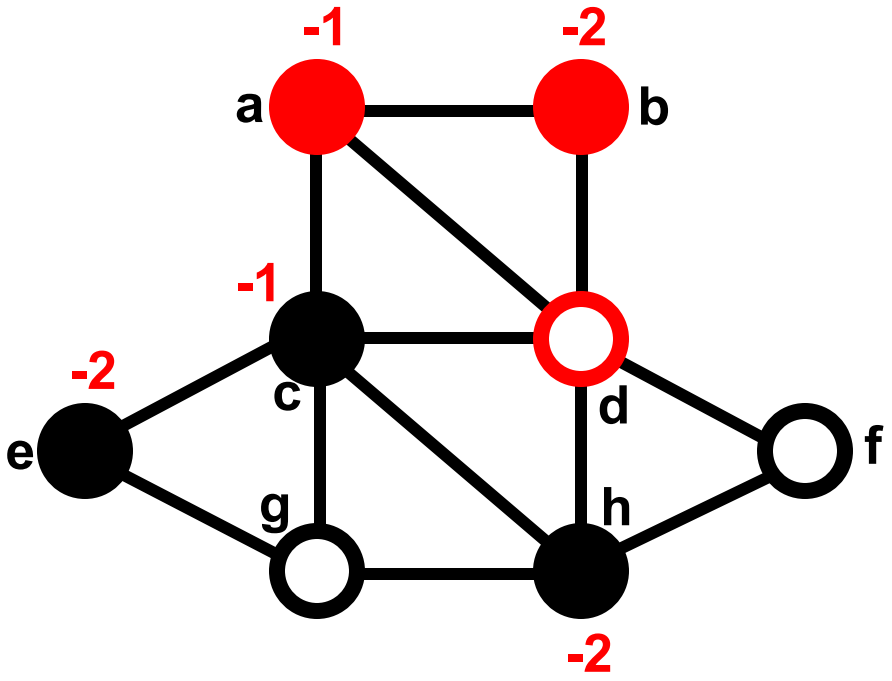



Nodes tentatively moved (and cut size after each pair):

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.



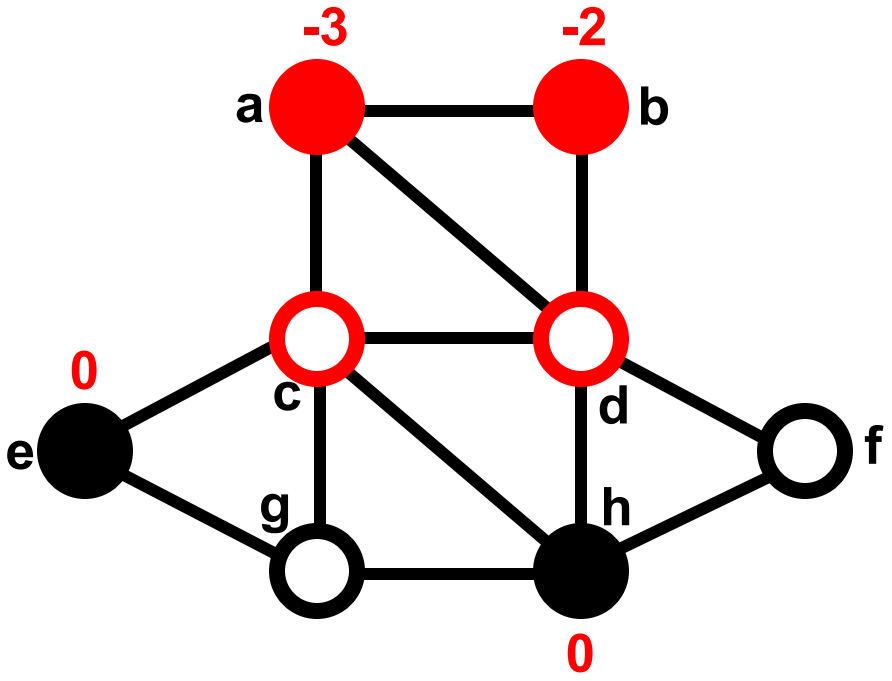

Nodes tentatively moved (and cut size after each pair):

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.

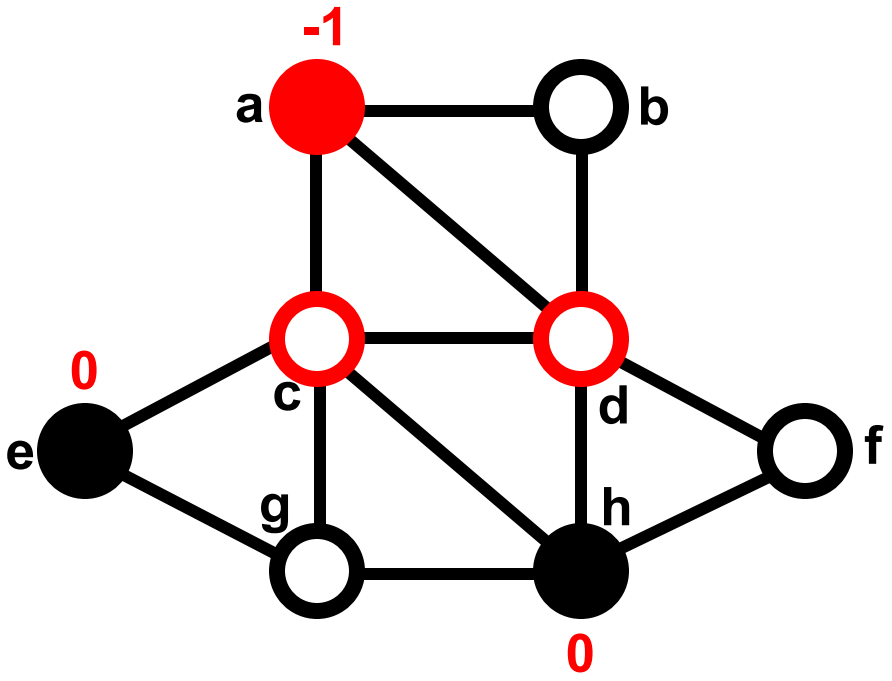



Nodes tentatively moved (and cut size after each pair):

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.



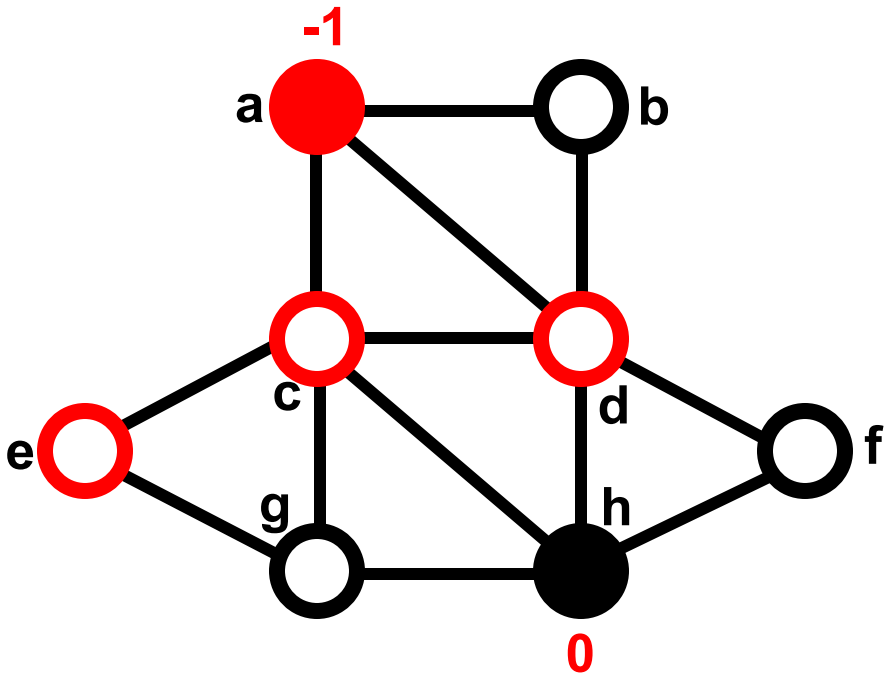

Nodes tentatively moved (and cut size after each pair):

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.

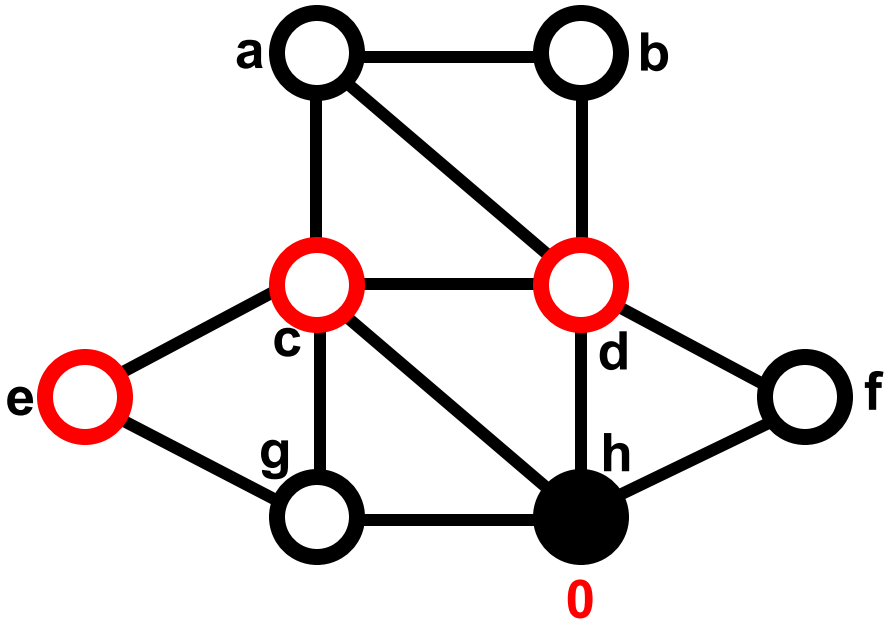



Nodes tentatively moved (and cut size after each pair):

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.



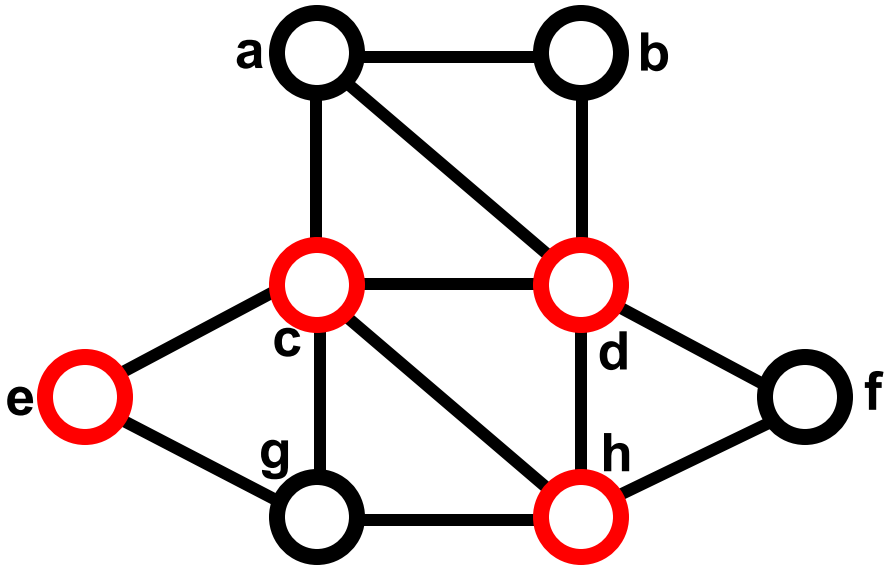

Nodes tentatively moved (and cut size after each pair):

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.



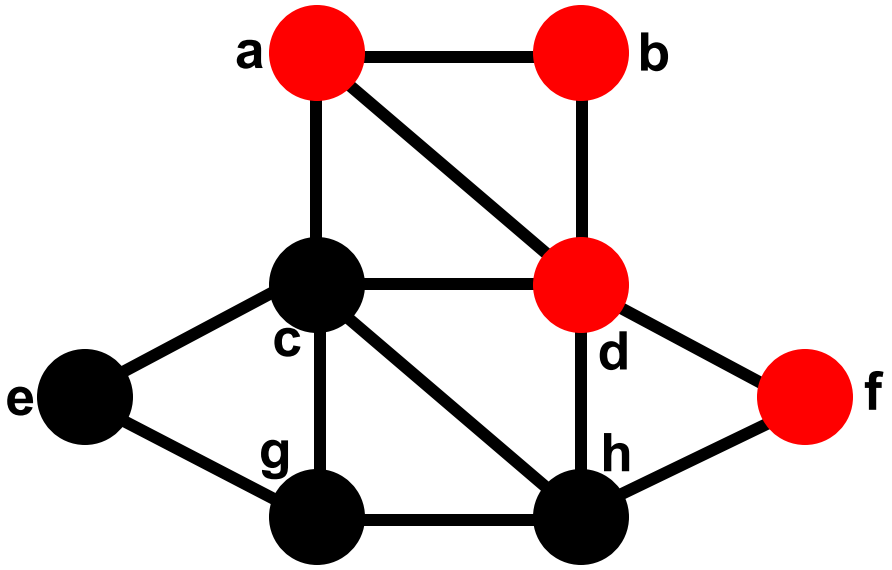

Nodes tentatively moved (and cut size after each pair):

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):

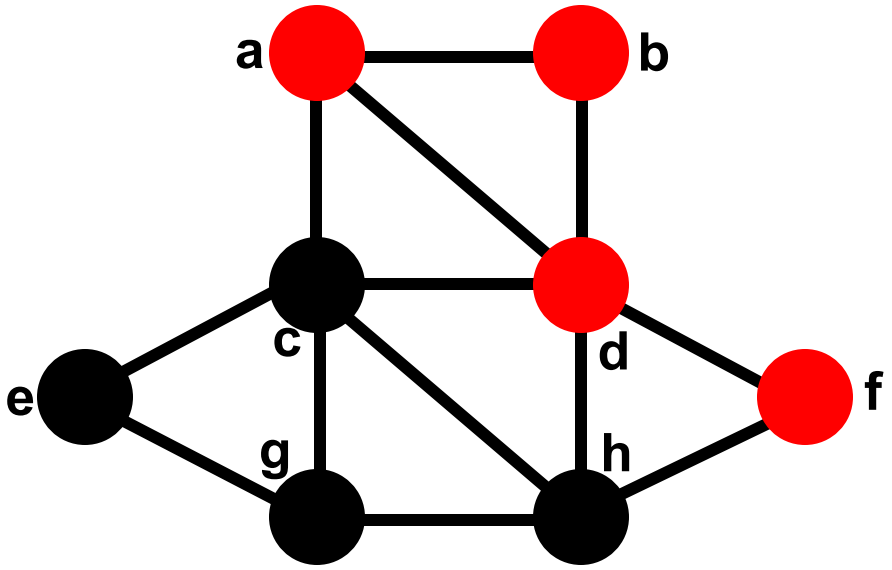
none (8); **g, d (4);** f, c (5); b, e (7); a, h (8)

# 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.



# 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 (INI<=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 $2^{-INI/30}$

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

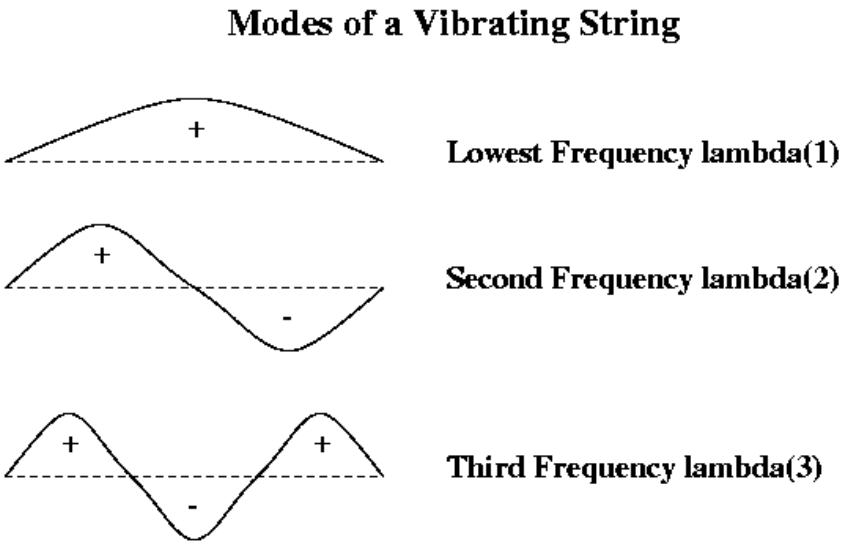
Buluç, A., Meyerhenke, H., Safro, I., Sanders, P., & Schulz, C. (2016). Recent advances in graph partitioning. In Algorithm Engineering.

# Coordinate-Free: 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)



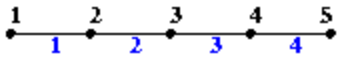
# Basic Definitions

- *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. 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 Simple Meshes

Incidence and Laplacian Matrices

Graph G

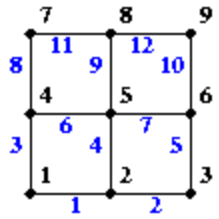



Incidence Matrix In(G)

| | 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

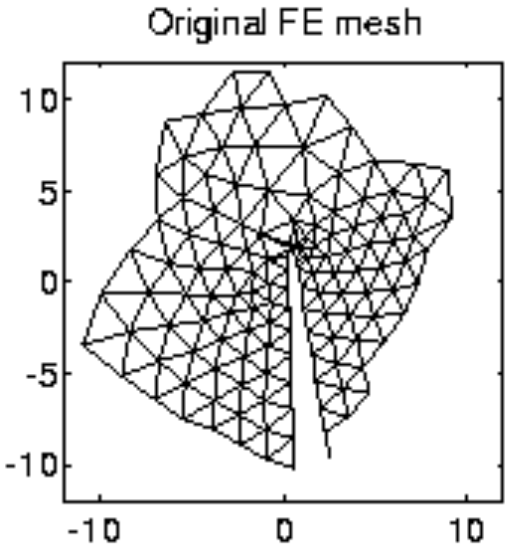
# Properties of Laplacian Matrix

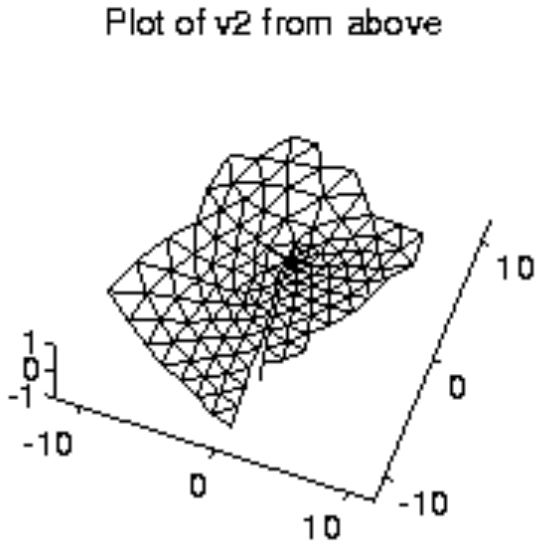
- *Theorem 1:* Given G, L(G) has the following properties
  (proof on 1996 CS267 web page)
  - L(G) is symmetric.
    - This means the eigenvalues of L(G) are real and its eigenvectors are real and orthogonal.
  - $In(G) * (In(G))^T = L(G)$
  - The eigenvalues of L(G) are nonnegative:
    - $0 = \lambda_1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

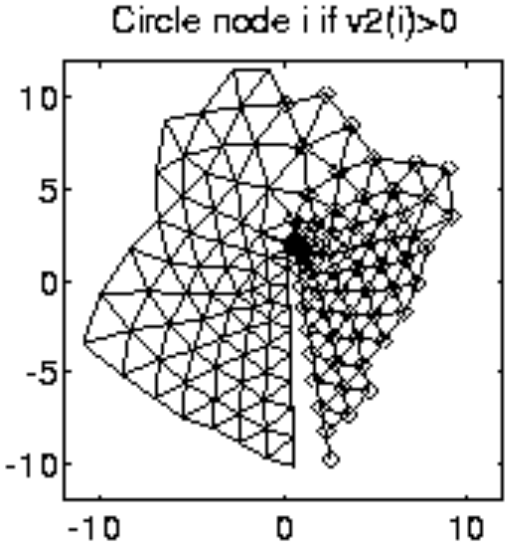
# Spectral Bisection Algorithm

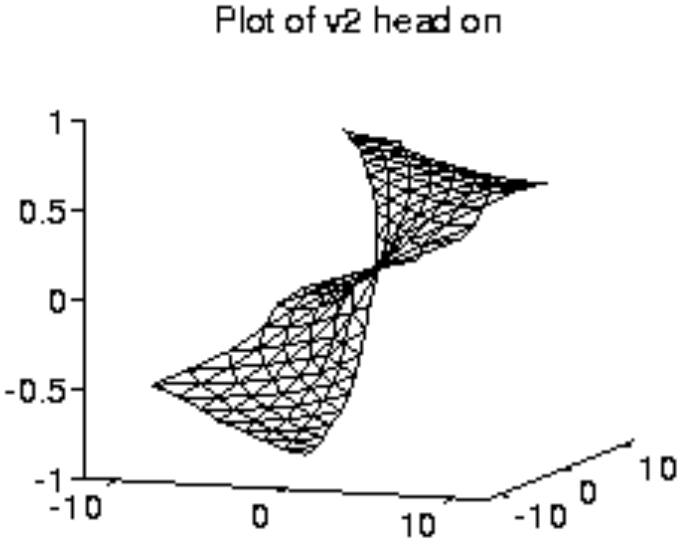
- Spectral Bisection Algorithm:
  - Compute eigenvector $v_2$ corresponding to $\lambda_2(L(G))$
  - For each node n of G
    - if $v_2(n) < 0$ put node n in partition N-
    - else put node n in partition N+
- Why does this make sense? Some reasons...
  - *Theorem 2 (Fiedler, 1975):* Let G be connected, and N- and N+ defined as above. Then N- is connected. If no $v_2(n) = 0$, then N+ is also connected. (proof on 1996 CS267 web page)
  - 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. (proof on 1996 CS267 web page)

# 2nd eigenvector of L(planar mesh)



# Computing $v_2$ and $\lambda_2$ 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

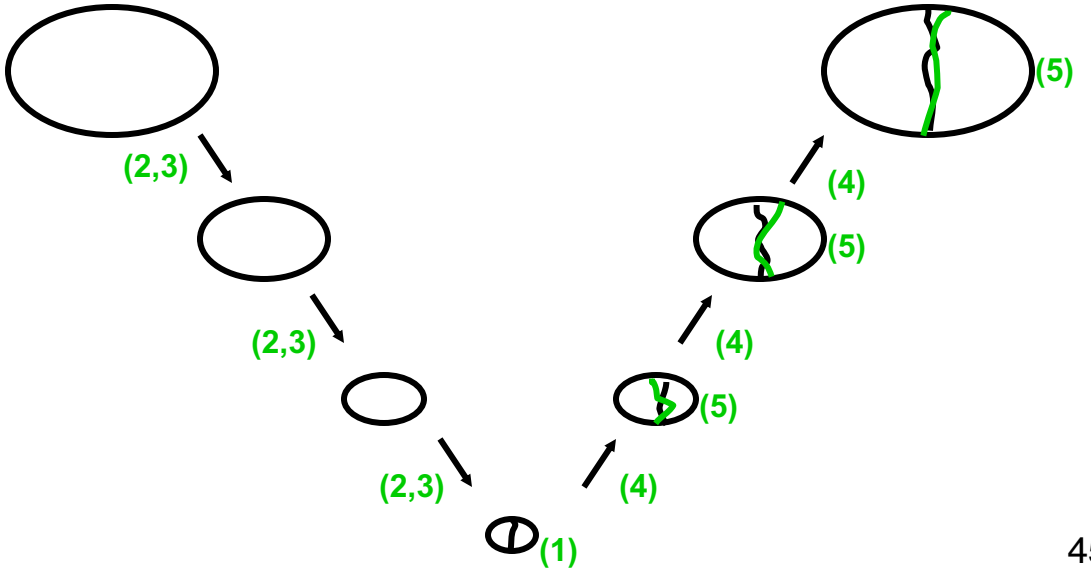
# Introduction to Multilevel Partitioning

- 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 Gc = (Nc, Ec)
(3)         (Nc+ , Nc- ) = Multilevel_Partition( Nc, Ec )
(4)         Expand (Nc+ , Nc- ) 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

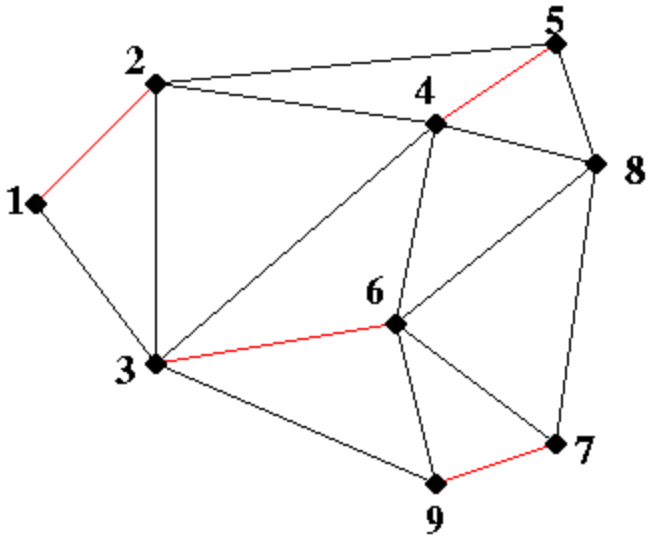
- Coarsen graph and expand partition using maximal matchings
- Improve partition using Kernighan-Lin

# Maximal Matching

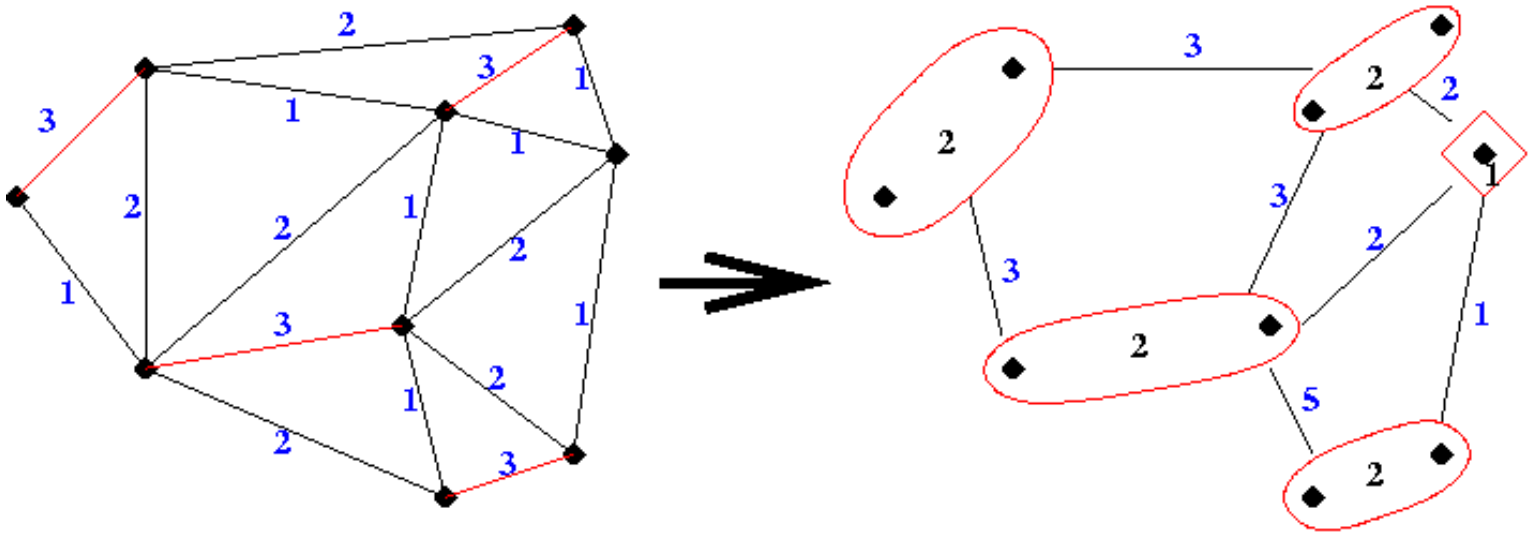
- *Definition*: A matching of a graph G(N,E) is a subset $E_m$ of E such that no two edges in $E_m$ share an endpoint
- *Definition:* A maximal matching of a graph G(N,E) is a matching $E_m$ to which no more edges can be added and remain a matching
- A simple greedy algorithm computes a maximal matching:

```
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**

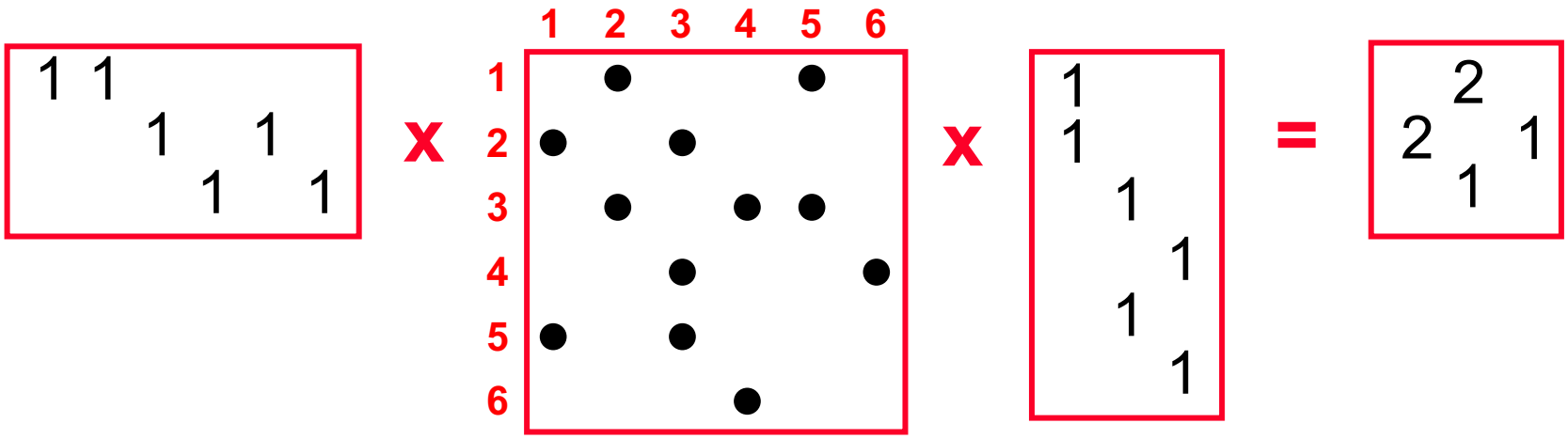
    **Put edge ee = (n(e),n(r)) in $E_c$**

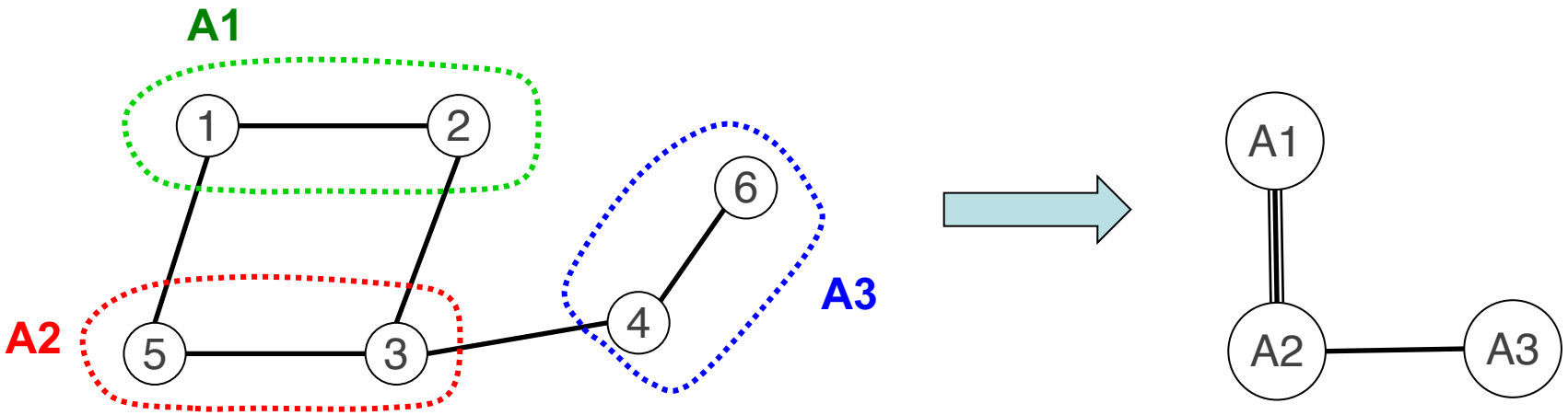
    W(ee) = W(e' )

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

adding edge weights

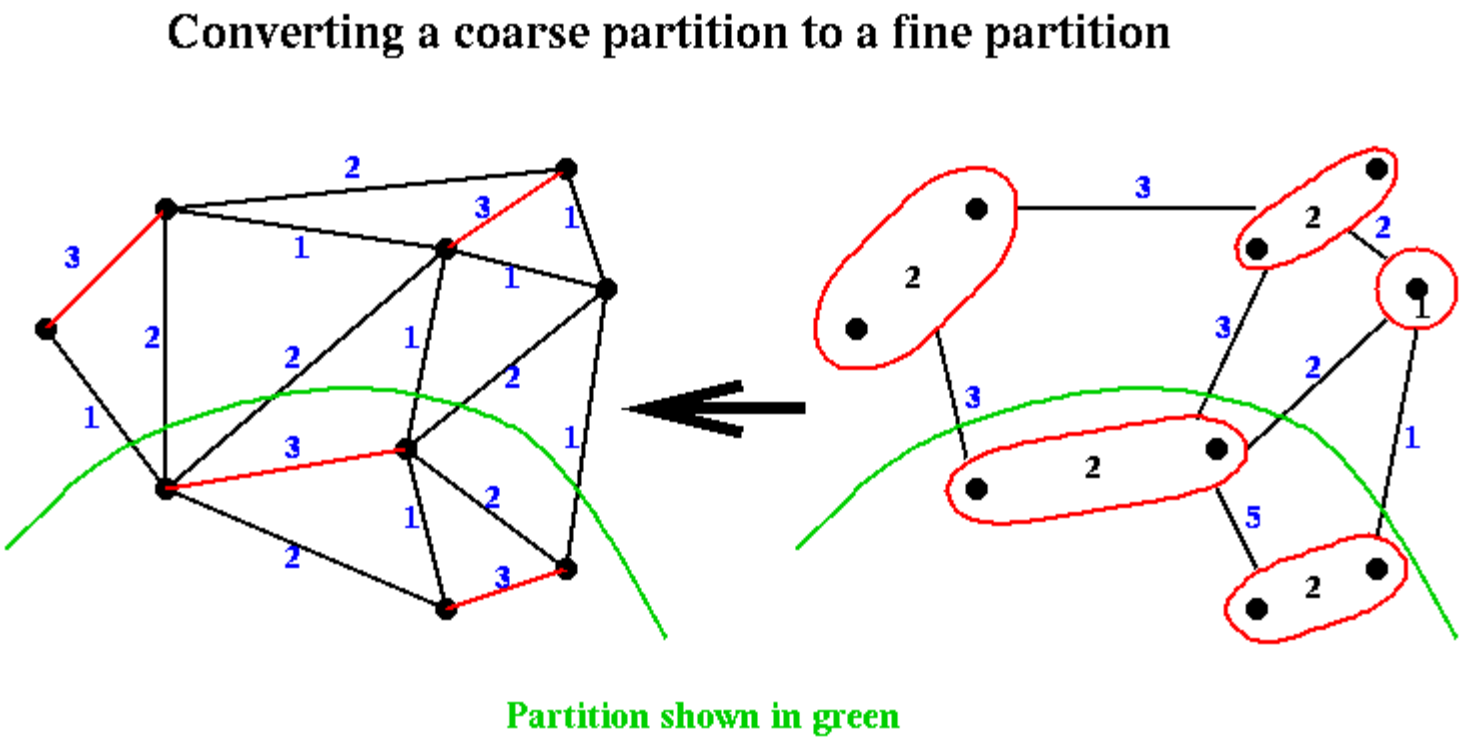
# Coarsening via sparse matrix-matrix products




Aydin Buluç and John R. Gilbert. Parallel sparse matrix-matrix multiplication and indexing: Implementation and experiments. *SIAM Journal of Scientific Computing (SISC), 2012.*

# Expanding a partition of $G_c$ to a partition of G



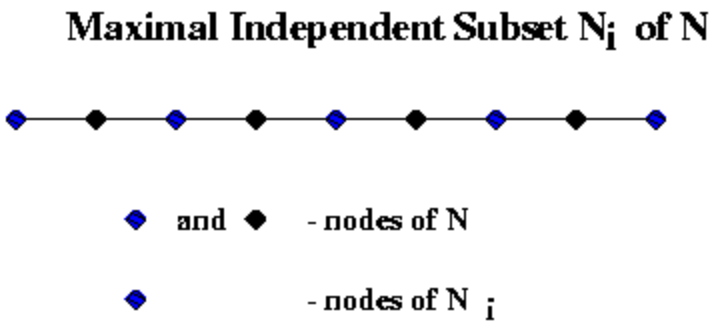
# Multilevel Spectral Bisection

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

# Maximal Independent Sets

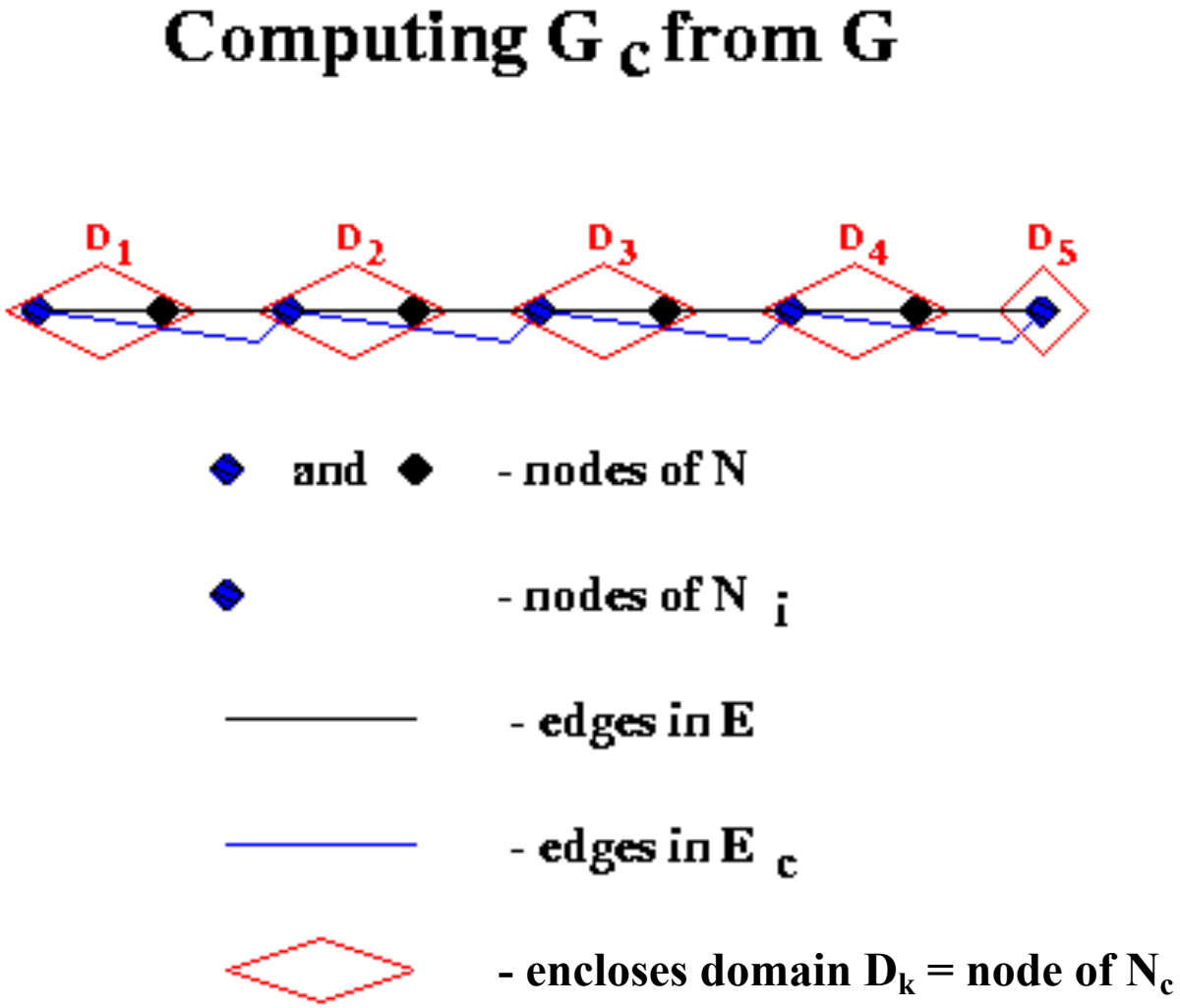
- *Definition*: An independent set of a graph G(N,E) is a subset $N_i$ of N such that no two nodes in $N_i$ are connected by an edge
- *Definition:* A maximal independent set (MIS) of a graph G(N,E) is an independent set $N_i$ to which no more nodes can be added and remain an independent set
- A simple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

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




- We will see parallel MIS in the graph algorithms class next week

# Example of Coarsening



# 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

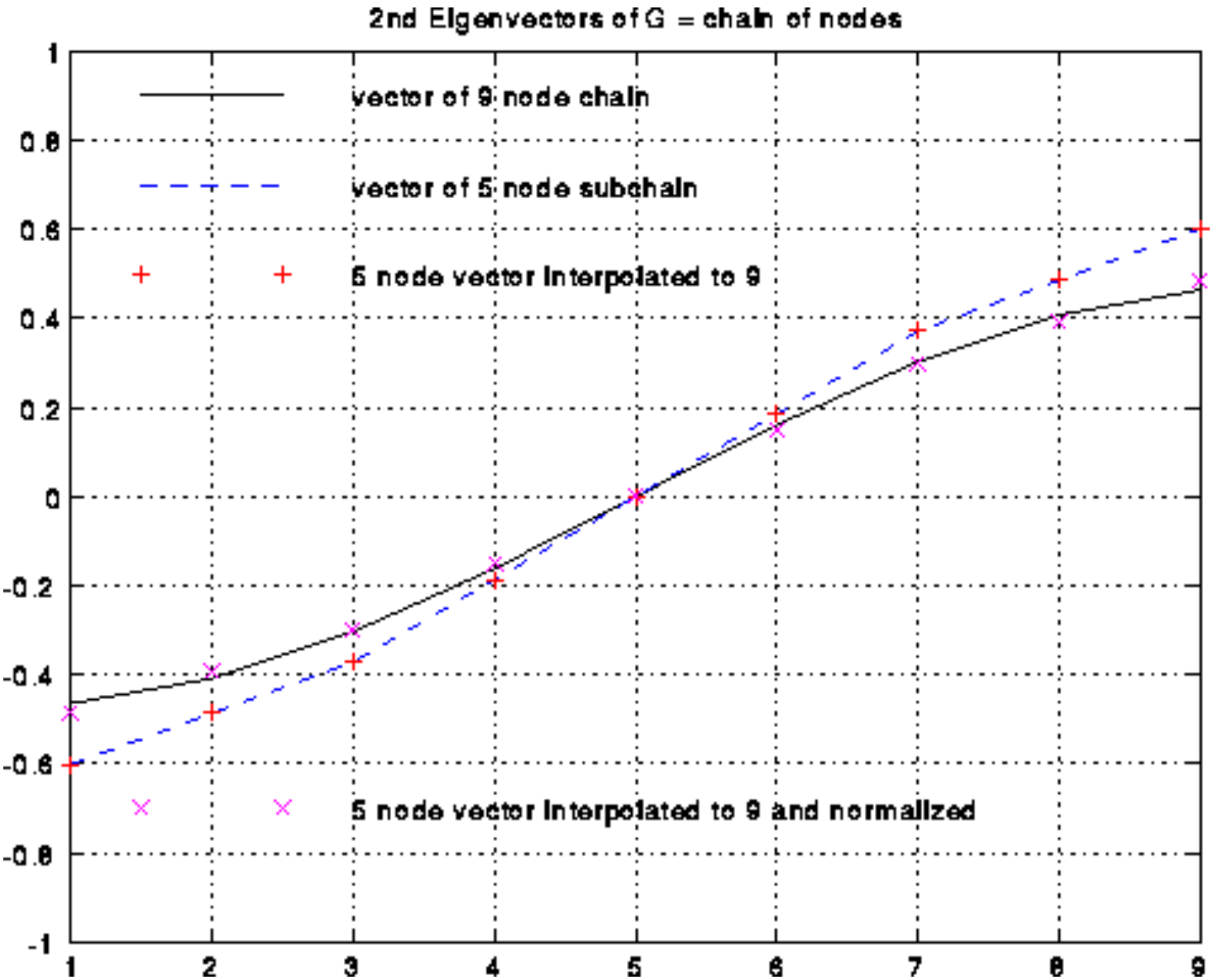
until no unmarked edges

# 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**

# Example: 1D mesh of 9 nodes



# 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,

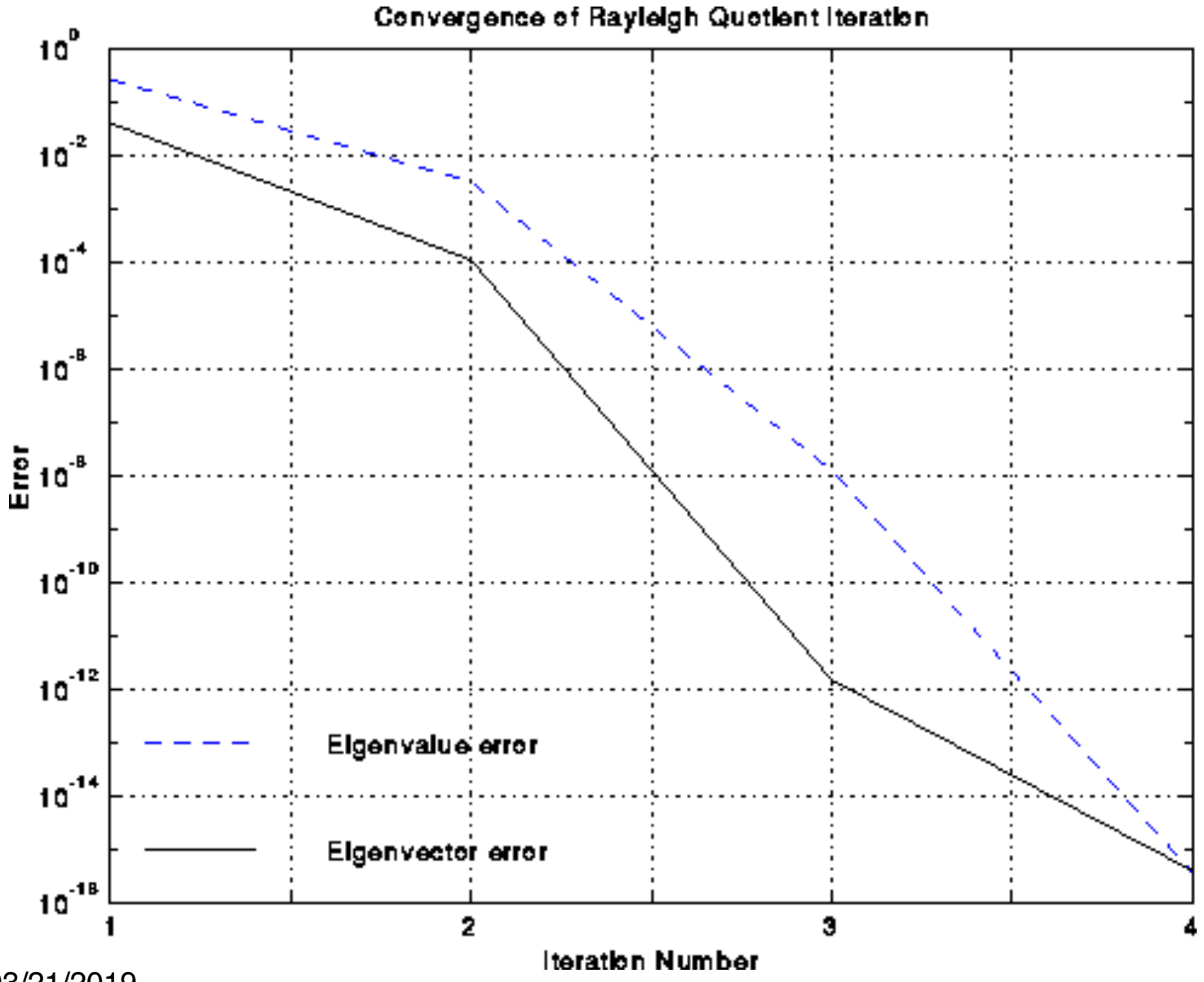
… which uses matrix-vector multiply (no surprise)

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

**until v(j) converges**

… Convergence is very fast: cubic

# Example of cubic convergence for 1D mesh



# 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

# Available Implementations

- Multilevel Kernighan/Lin
  - METIS and ParMETIS (glaros.dtc.umn.edu/gkhome/views/metis)
  - SCOTCH and PT-SCOTCH (www.labri.fr/perso/pelegrin/scotch/)
- Matlab toolbox for geometric and spectral partitioning by Gilbert, Tang, and Li: https://github.com/YingzhouLi/meshpart
- Multilevel Spectral Bisection
  - S. Barnard and H. Simon, "A fast multilevel implementation of recursive spectral bisection …", 1993
  - Chaco (SC'14 Test of Time Award)
- Hybrids possible
  - Ex: Use Kernighan/Lin to improve a partition from spectral bisection
- Recent packages with collection of techniques
  - Zoltan (www.cs.sandia.gov/Zoltan)
  - KaHIP (http://algo2.iti.kit.edu/kahip/)

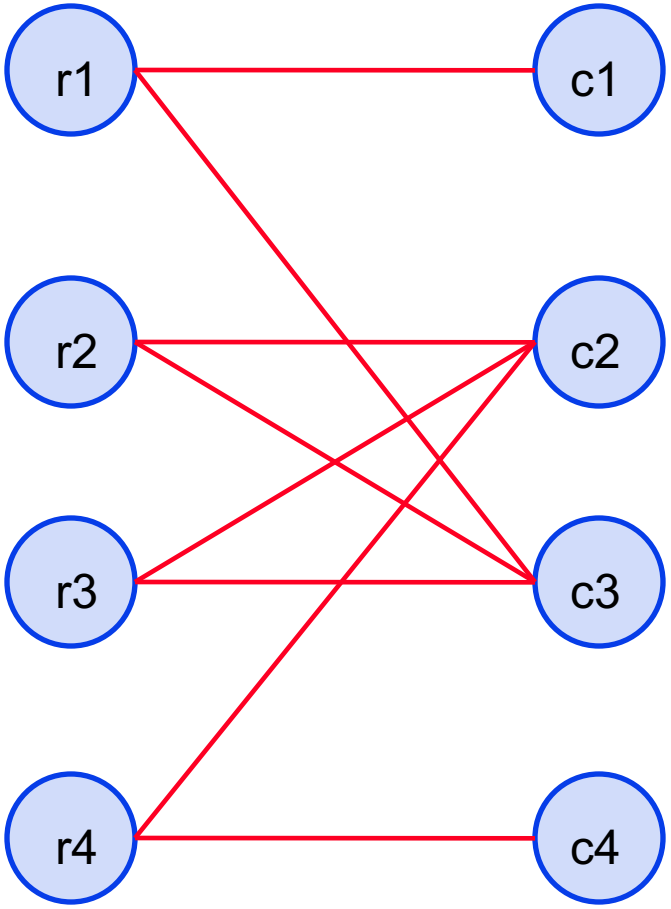
# Outline of Graph Partitioning Lectures

- 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**

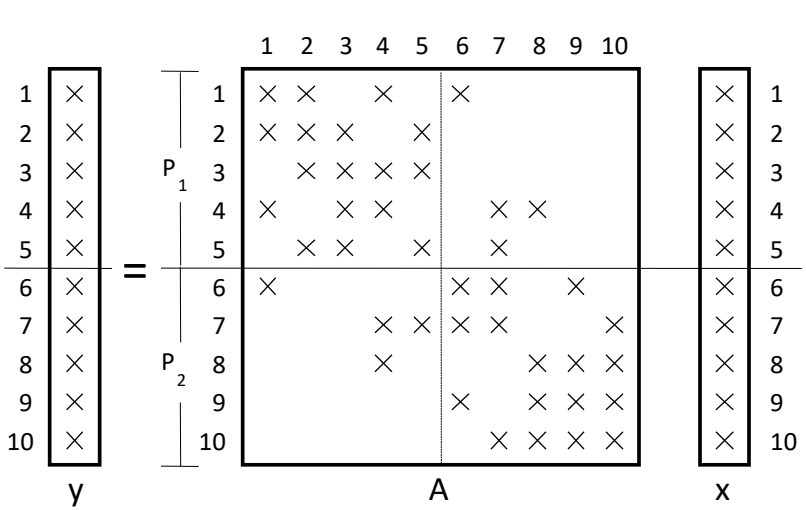
# Beyond simple graph partitioning:

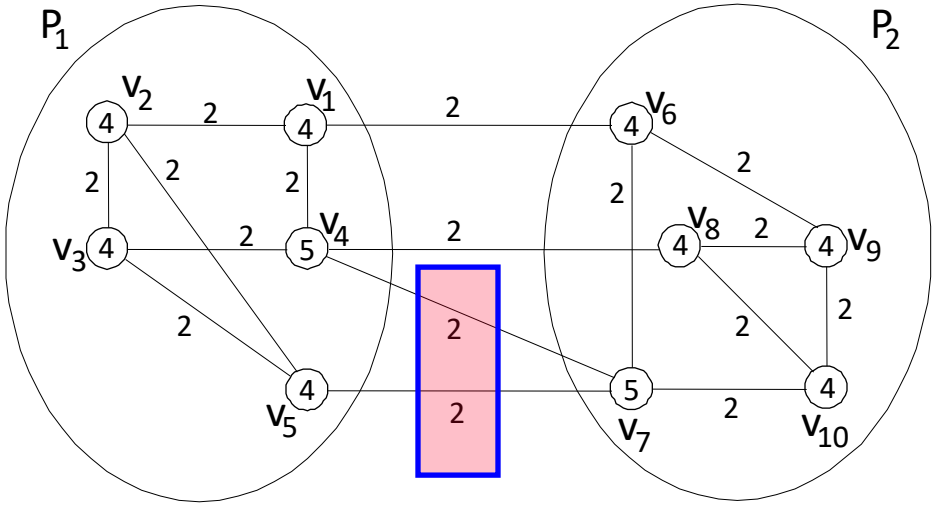
## Representing a sparse matrix as a hypergraph

$$\begin{bmatrix} \times & 0 & \times & 0 \\ 0 & \times & \times & 0 \\ 0 & \times & \times & 0 \\ 0 & \times & 0 & \times \end{bmatrix}$$



# A sparse matrix in the graph model




edge $(v_i, v_j) \in E \Rightarrow$

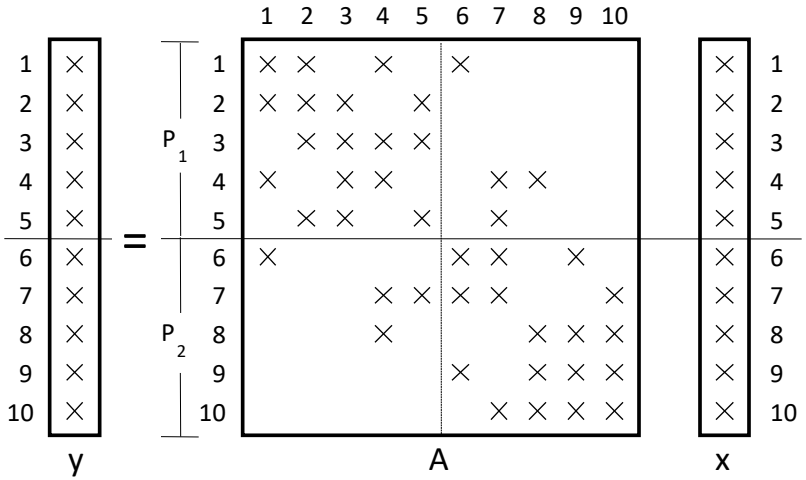
$y(i) \leftarrow y(i) + A(i,j)\, x(j)$ and $y(j) \leftarrow y(j) + A(j,i)\, x(i)$

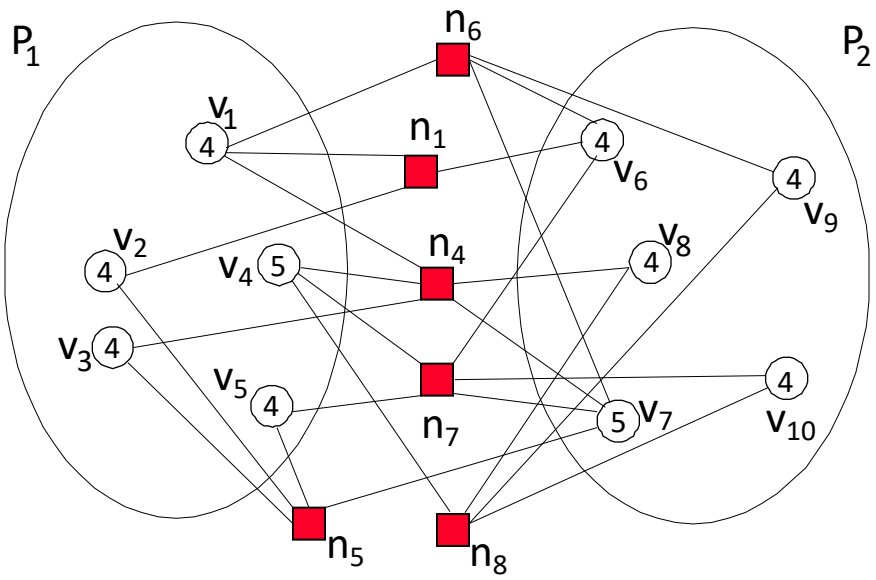
$P_1$ performs: $y(4) \leftarrow y(4) + A(4,7)\, x(7)$ and

$y(5) \leftarrow y(5) + A(5,7)\, x(7)$

x(7) only needs to be communicated **once !**

# A sparse matrix in the hypergraph model




- Column-net model for block-row distributions
- Rows are vertices, columns are nets (hyperedges)

Each {vertex, net} pair represents unique nonzero net-cut metric:

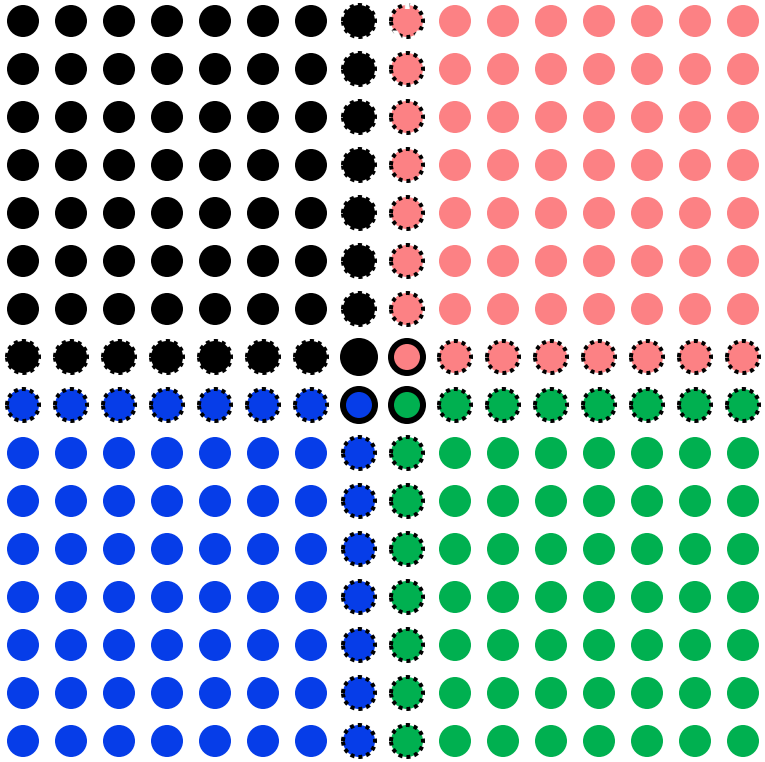
$$\text{cutsize}(\Pi) = \Sigma_{n \in NE} w(n_i)$$

connectivity-1 metric:

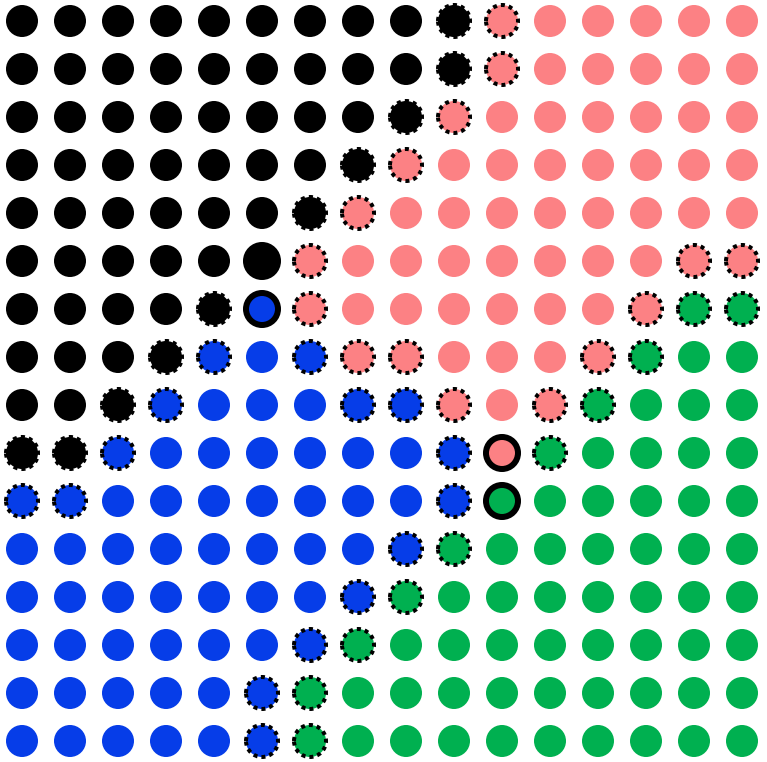
$$\text{cutsize}(\Pi) = \Sigma_{n \in NE} w(n_i)\,(c(n_j) - 1)$$

# Two Different 2D Mesh Partitioning Strategies

Graph:
Cartesian Partitioning



Hypergraph:
MeshPart Algorithm [Ucar, Catalyurek, 2010]

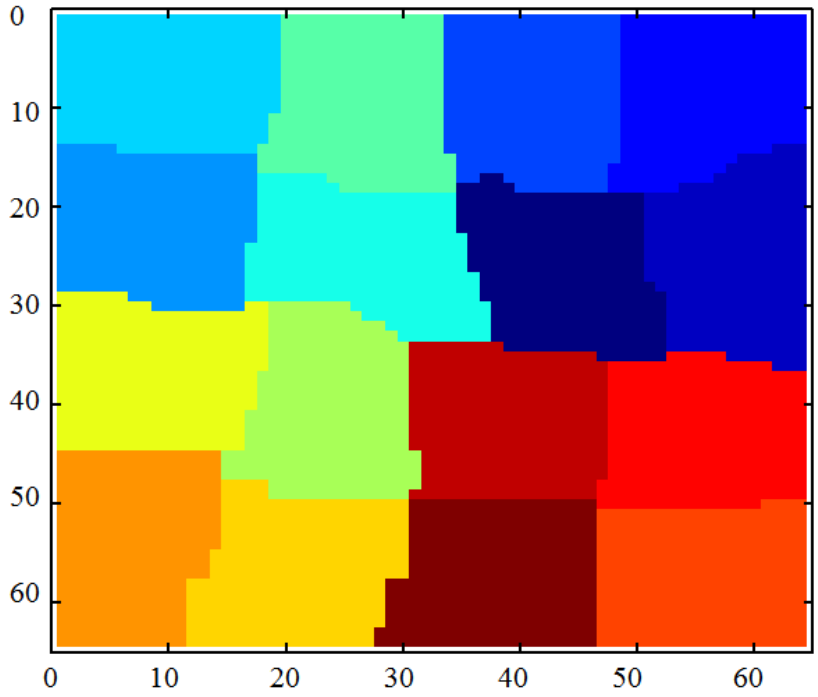


Total SpMV communication volume = 64

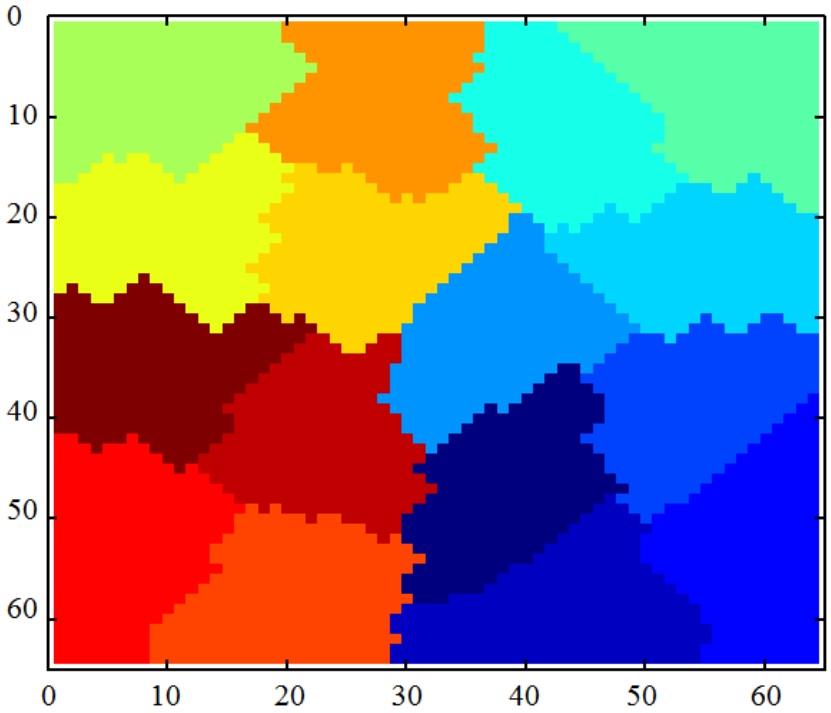
Total SpMV communication volume = 58

# Experimental Results: Hypergraph vs. Graph Partitioning

**64x64 Mesh (5-pt stencil), 16 processors**




Graph Partitioning (Metis)
Total Comm. Vol = 777
Max Vol per Proc = 69

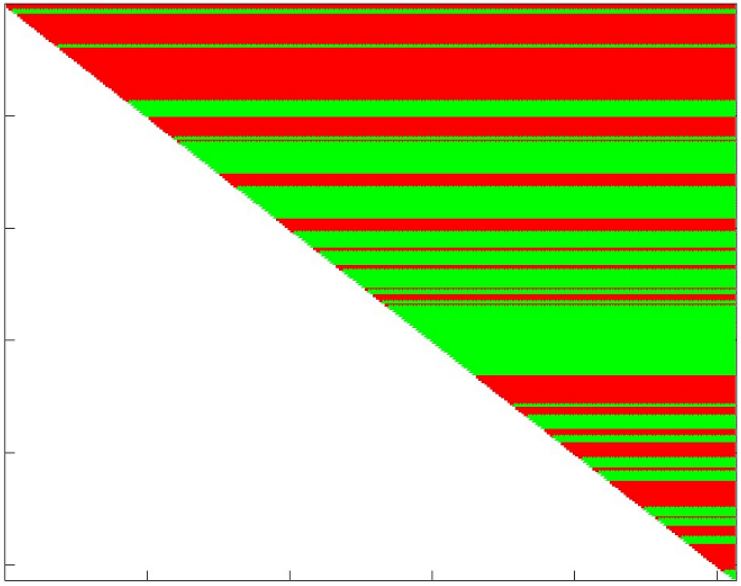



Hypergraph Partitioning (PaToH)
Total Comm. Vol = 719
Max Vol per Proc = 59

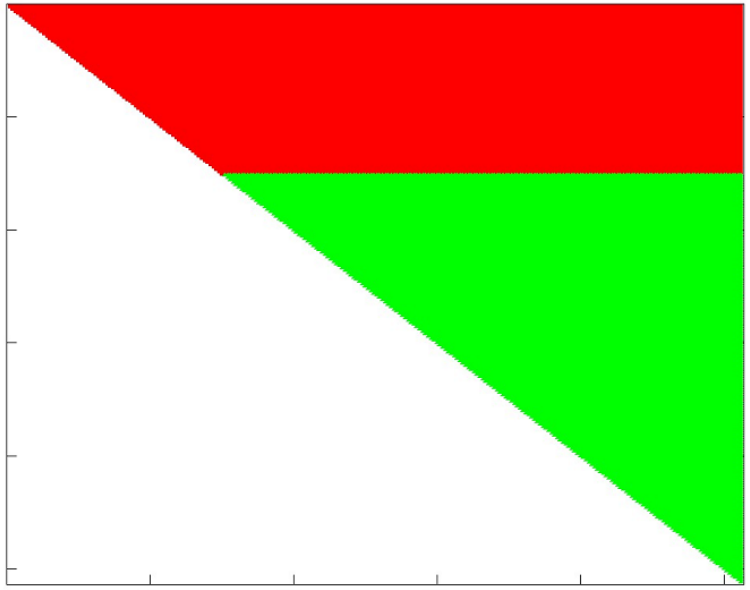
**~8% reduction in total communication volume using hypergraph partitioning (PaToH) versus graph partitioning (METIS)**

# Further Benefits of Hypergraph Model: Nonsymmetric Matrices

- Graph model of matrix has edge (i,j) if either A(i,j) or A(j,i) nonzero
- Same graph for A as $|A| + |A^T|$
- Ok for symmetric matrices, what about nonsymmetric?
  - Try A upper triangular



**Graph Partitioning (Metis)**
Total Communication Volume= 254
Load imbalance ratio = 6%



**Hypergraph Partitioning (PaToH)**
Total Communication Volume= 181
Load imbalance ratio = 0.1%

# Summary: Graphs versus Hypergraphs

- Pros and cons
  - When matrix is non-symmetric, the graph partitioning model (using $A+A^T$ ) loses information, resulting in suboptimal partitioning in terms of communication and load balance.
  - Even when matrix is symmetric, graph cut size is not an accurate measurement of communication volume
  - Hypergraph partitioning model solves both these problems
  - However, hypergraph partitioning (PaToH) can be much more expensive than graph partitioning (METIS)
- Hypergraph partitioners: PaToH, HMETIS, ZOLTAN

# Is Graph Partitioning a Solved Problem?

- Myths of partitioning due to Bruce Hendrickson

1. Edge cut = communication cost
2. Simple graphs are sufficient
3. Edge cut is the right metric
4. Existing tools solve the problem
5. Key is finding the right partition
6. Graph partitioning is a solved problem

- Slides and myths based on Bruce Hendrickson's:

  "Load Balancing Myths, Fictions & Legends"

# Myth: Partition Quality is Paramount

- When structure are changing dynamically during a simulation, need to partition dynamically
  - Speed may be more important than quality
  - **Partitioner must run fast in parallel**
    - Another chicken and egg problem here
  - Partition should be incremental
    - Change minimally relative to prior one
  - Must not use too much memory

- Recent research on streaming partitioning:
  - Stanton, I. and Kliot, G., "Streaming graph partitioning for large distributed graphs". KDD, 2012.
  - The idea is used by many graph processing systems such as PowerGraph and GPS