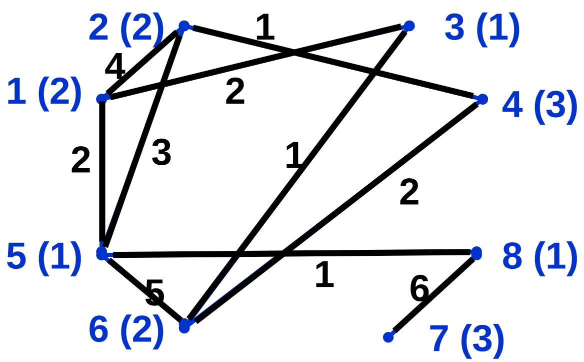
Lecture 9:
Graph Partitioning

Outline

- 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
- Graph algorithms in sparse direct methods

Definition of Graph Partitioning

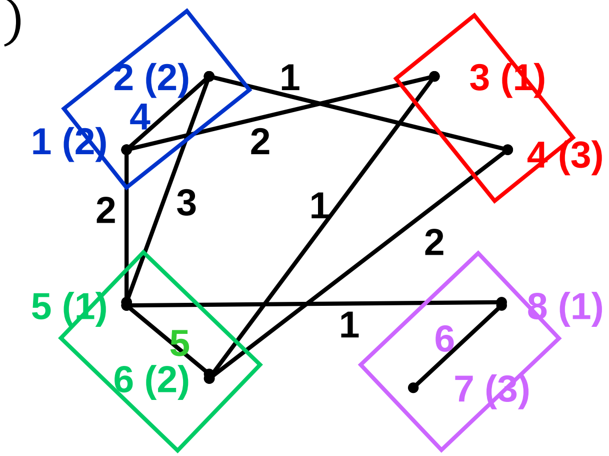
- Given a graph $G = (N, E, W_N, W_E)$
  - N = nodes (or vertices),
  - W_N = node weights
  - E = edges
  - W_E = edge weights



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

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



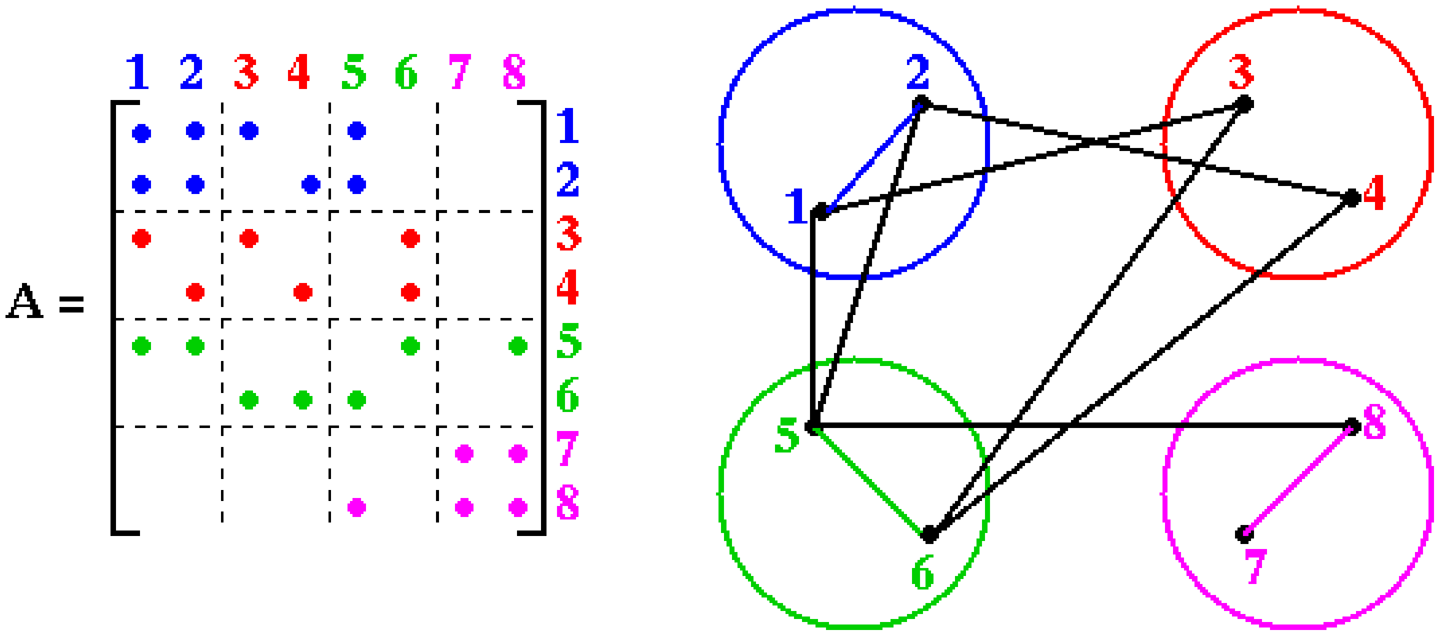
- Ex: N = {tasks}, W_N = {task costs}, edge (j,k) in E means task j sends $W_E(j,k)$ words to task k
- Choose a partition $N = N_1 \cup N_2 \cup \cdots \cup N_P$ such that
  - The sum of the node weights in each N_j is "about the same"
  - The sum of all edge weights of edges connecting all different pairs N_j and N_k is minimized (shown in black)
- Ex: balance the work load, while minimizing communication
- Special case of $N = N_1 \cup N_2$: 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
  - $N = \{1, \dots, n\}, \quad (j,k) \in E$ if $A(j,k)$ nonzero,
  - $W_N(j) = \#$nonzeros in row j, $\quad W_E(j,k) = 1$
- 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 $y = y + Ax$

Partitioning a Sparse Symmetric Matrix



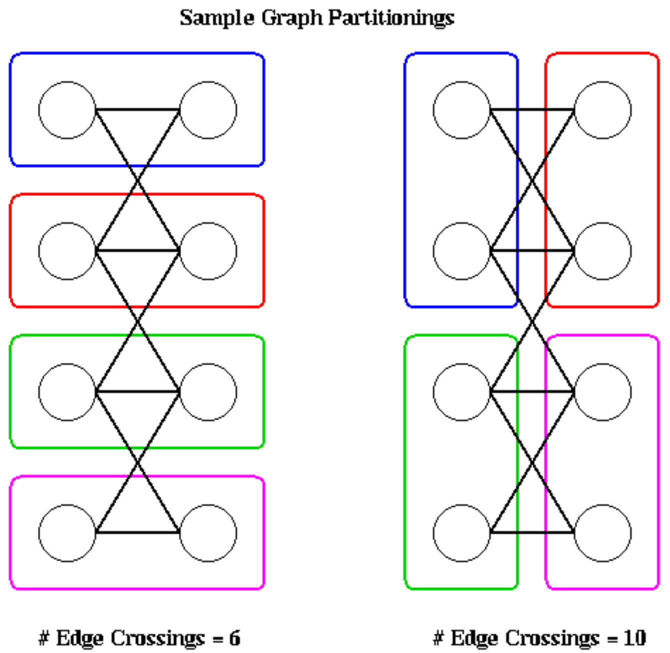

```
... declare A_local, A_remote(1:num_procs), x_local, x_remote, y_local
y_local = y_local + A_local * x_local
for all procs P that need part of x_local
          send(needed part of x_local, P)
for all procs P owning needed part of x_remote
          receive(x_remote, P)
          y_local = y_local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

Cost of Graph Partitioning

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

$$\binom{n}{n/2} = \frac{n!}{\left(\left(\frac{n}{2}\right)!\right)^2} \approx 2^n \left(\frac{2}{n\pi}\right)^{1/2}$$

possibilities



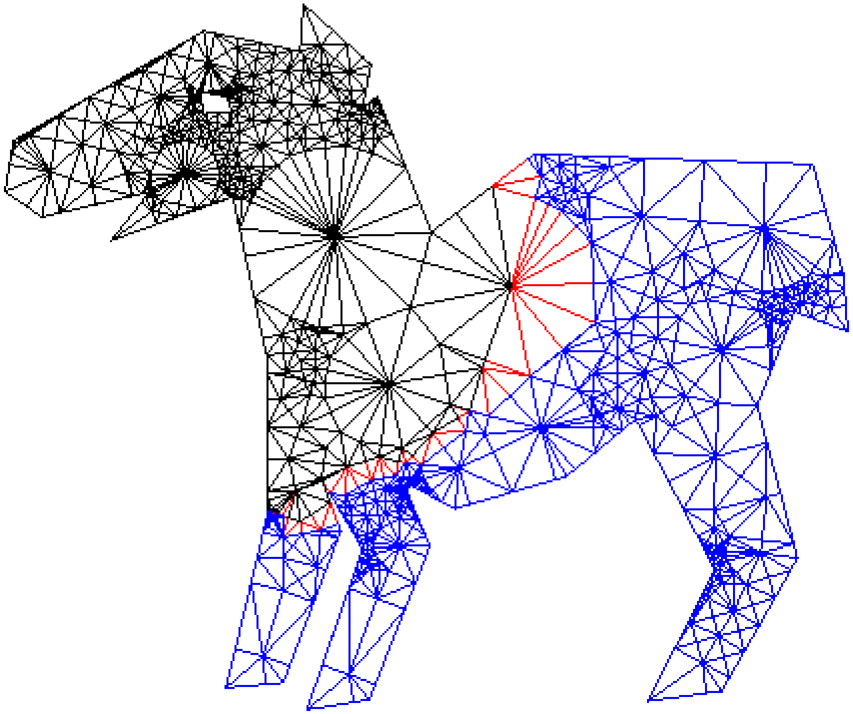
- Choosing optimal partitioning is NP-complete
  - (NP-complete = we can prove it is a hard as other well-known hard problems in a class Nondeterministic Polynomial time)
  - Only known exact algorithms have cost = exponential(n)
- We need good heuristics!

Outline

- 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
- Graph algorithms in sparse direct methods

First Heuristic: Repeated Graph Bisection

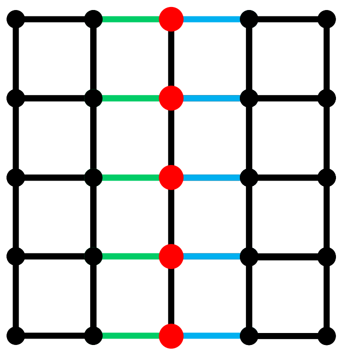
- To partition N into 2^k parts
  - bisect graph recursively k times
- Henceforth discuss mostly graph bisection



Edge Separators vs. Vertex Separators

- Edge Separator: E_s (subset of E) separates G if removing E_s from E leaves two approx. 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 approx. equal-sized, disconnected components of N: N_1 and N_2

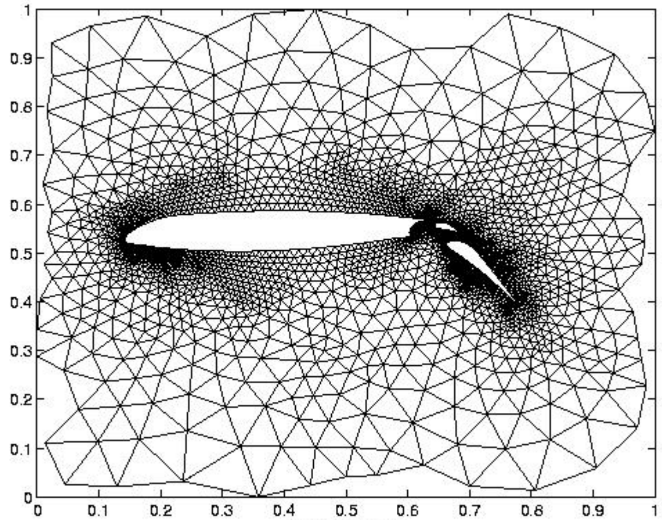
$G = (N, E)$, Nodes N and Edges E
$E_s =$ green edges or blue edges
$N_s =$ red vertices



- Making a 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 a 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

- 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
- Graph algorithms in sparse direct methods

Nodal Coordinates: How Well Can We Do?

- A planar graph can be drawn in plane without edge crossings
- Ex: $m \times m$ 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, $\exists$ $\mathrm{N_s}$ such that
  - $N = N_1 \cup N_s \cup N_2$ is a partition,
  - $|N_1| \leq 2/3\,|N|$ and $|N_2| \leq 2/3\,|N|$
  - $|N_s| \leq (8\,|N|)^{1/2}$

- 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 $a(x-\bar{x})+b(y-\bar{y})=0$,

   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

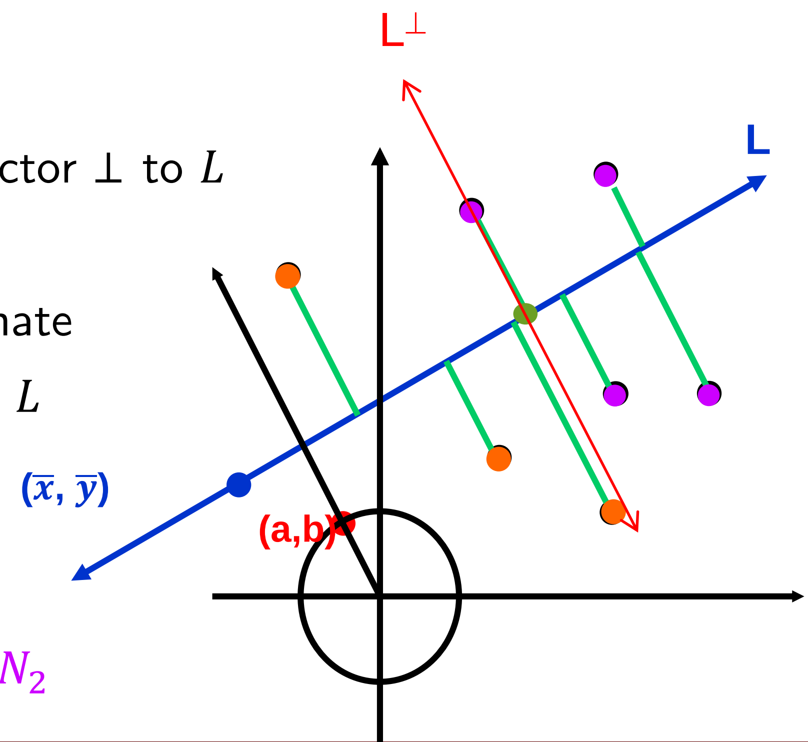
   $S_j=-b(x_j-\bar{x})+a(y_j-\bar{y})$ along L

3. Compute the median

   Let $\bar{S}=\text{median}(S_1,\ldots,S_n)$

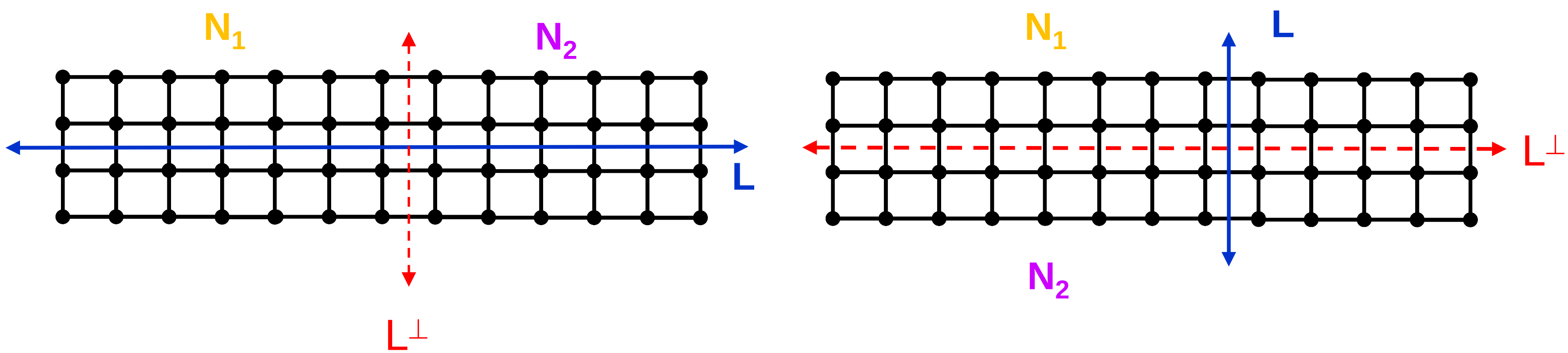
4. Use median to partition the nodes

   Let nodes with $S_j<\bar{S}$ be in N_1, rest in N_2



Inertial Partitioning: Choosing L

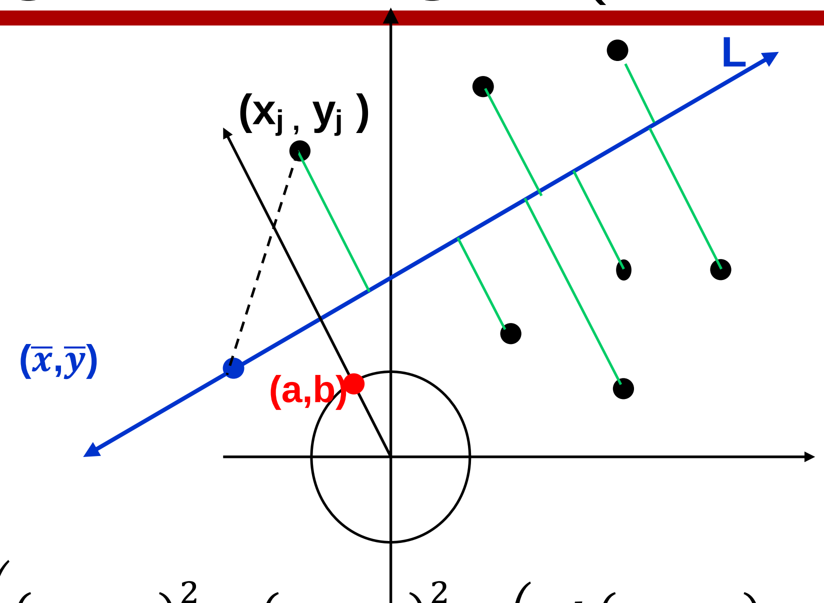
- Clearly prefer L, $L^{\perp}$ on left below



- 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



$\sum_j$ (length of j–th green line)2 $= \sum_j \left(\left(x_j - \bar{x}\right)^2 + \left(y_j - \bar{y}\right)^2 - \left(-b\left(x_j - \bar{x}\right) + a\left(y_j - \bar{y}\right)\right)^2 \right)$

… Pythagorean Theorem

$$= a^2 \sum_j \left(x_j - \bar{x}\right)^2 + 2ab \sum_j (x_j - \bar{x})(y_j - \bar{y}) + b^2 \sum_j \left(y_j - \bar{y}\right)^2$$
$$= a^2 X1 \quad + \quad 2abX2 \quad + \quad b^2 X3$$
$$= [a\ b] \begin{bmatrix} X1 & X2 \\ X2 & X3 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix}$$

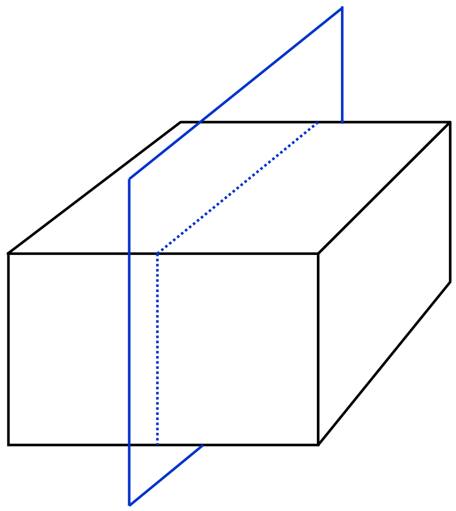
Minimized by choosing

$(\bar{x}, \bar{y}) = \left(\sum_j x_j, \sum_j y_j\right)/n$ = center of mass

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

Nodal Coordinates: Random Spheres

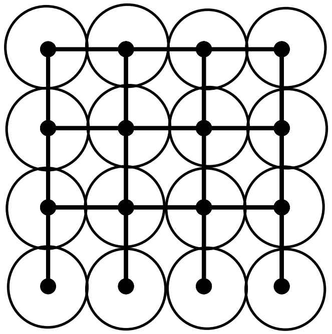
- Generalize nearest neighbor idea of a planar graph to higher dimensions
  - Any graph can fit in 3D without edge crossings
  - Capture intuition of planar graphs of being connected to "nearest neighbors" but in higher than 2 dimensions
- For intuition, consider graph defined by a regular 3D mesh
- An $n \times n \times n$ mesh of $|N| = n^3$ nodes
  - Edges to 6 nearest neighbors
  - Partition by taking plane parallel to 2 axes
  - Cuts $n^2 = |N|^{2/3} = O\left(|E|^{2/3}\right)$ edges
- For the general graphs
  - Need a notion of "well-shaped" like mesh



Random Spheres: Well-Shaped Graphs

- Approach due to Miller, Teng, Thurston, Vavasis
- Def: A k-ply neighborhood system in d dimensions is a set $\{D_1, \ldots, D_n\}$ of closed disks in $\mathbb{R}^d$ such that no point in $\mathbb{R}^d$ is strictly interior to more than k disks
- Def: An (α, k) overlap graph is a graph defined in terms of $\alpha \geq 1$ and a k-ply neighborhood system $\{D_1, \ldots, D_n\}$: There is a node for each D_j, and an edge from j to i if expanding the radius of the smaller of D_j and D_i by $> \alpha$ causes the two disks to overlap

Ex: $n \times n$ mesh is a (1,1) overlap graph
Ex: Any planar graph is (α, k) overlap for some α, k



2D Mesh is (1,1) overlap graph

Generalizing Lipton/Tarjan to Higher Dimensions

- *Theorem* (Miller, Teng, Thurston, Vavasis, 1993):

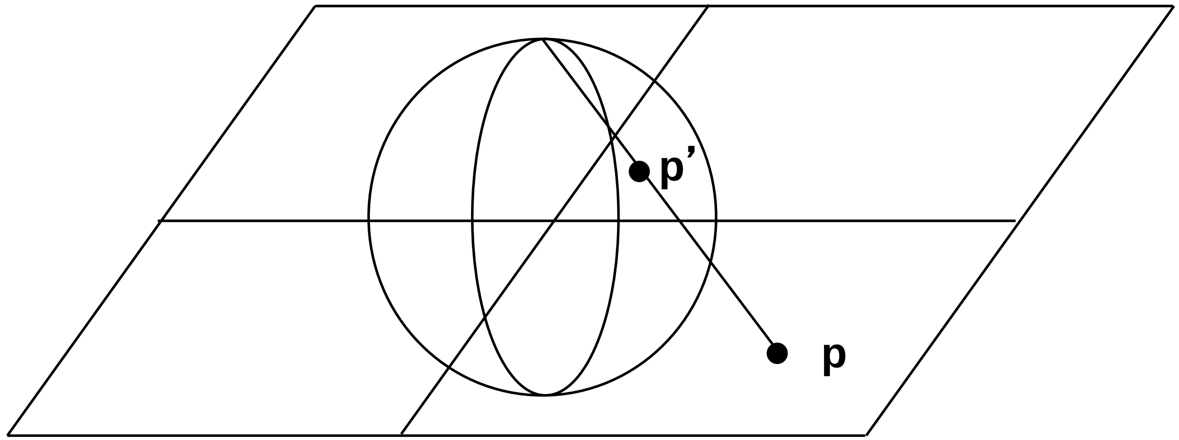
Let $G = (N, E)$ be an (α, k) overlap graph in d dimensions with $n = |N|$. Then there is a vertex separator N_s such that

  - $N \ = \ N_1 \cup \ N_s \cup \ N_2$ and
  - N_1 and N_2 each has at most $n(d+1)/(d+2)$ nodes
  - N_s has at most $O\left(\alpha k^{1/d} n^{(d-1)/d}\right)$ nodes

- When $d = 2$, similar to Lipton/Tarjan
- Algorithm:
  - Choose a sphere S in $\mathbb{R}^d$
  - Edges that S "cuts" form edge separator E_s
  - Build N_s from E_s
  - Choose S "randomly", so that it satisfies Theorem with high probability

Stereographic Projection

- Stereographic projection from plane to sphere
  - In $d = 2$, draw line from p to North Pole, projection p' of p is where the line and sphere intersect



p = (x,y) **p' = (2x,2y,x^2 + y^2 –1) / (x^2 + y^2 + 1)**

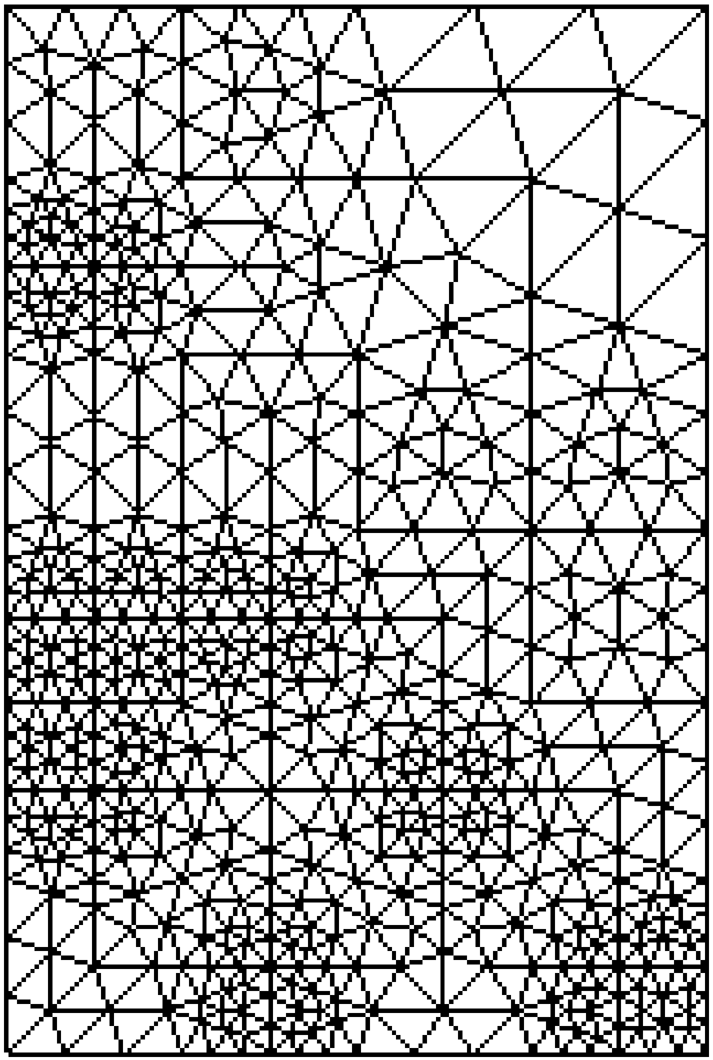
  - Similar in higher dimensions

Choosing a Random Sphere

- Do stereographic projection from $\mathbb{R}^d$ to sphere S in $\mathbb{R}^{d+1}$
- Find centerpoint of projected points
  - Any plane through centerpoint divides points approx. venly
  - There is a linear programming algorithm, cheaper heuristics
- *Conformally map* points on sphere
  - *Rotate* points around origin so centerpoint at $(0, \ldots 0, r)$ for some r
  - *Dilate* points (unproject, multiply by $\left(\frac{1-r}{1+r}\right)^{1/2}$, project)
    - this maps centerpoint to origin (0,...,0), spreads points around S
- Pick a random plane through origin
  - Intersection of plane and sphere S is "circle"
- Unproject circle
  - yields desired circle C in $\mathbb{R}^d$
- Create N_s: j belongs to N_s if αD_j intersects C

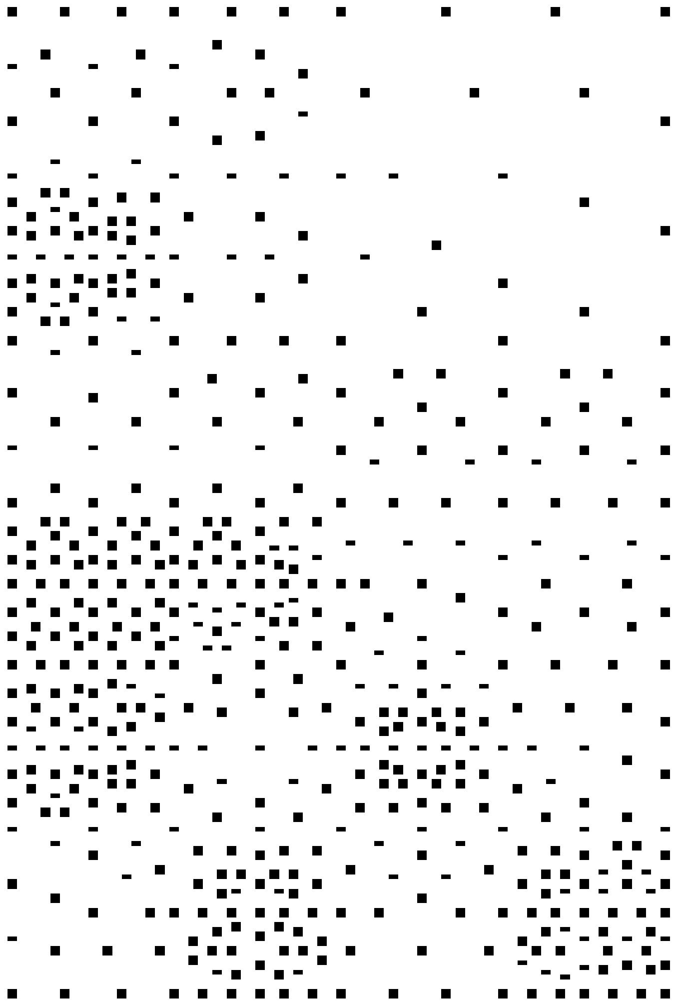
Random Sphere Algorithm (Gilbert)

Finite Element Mesh



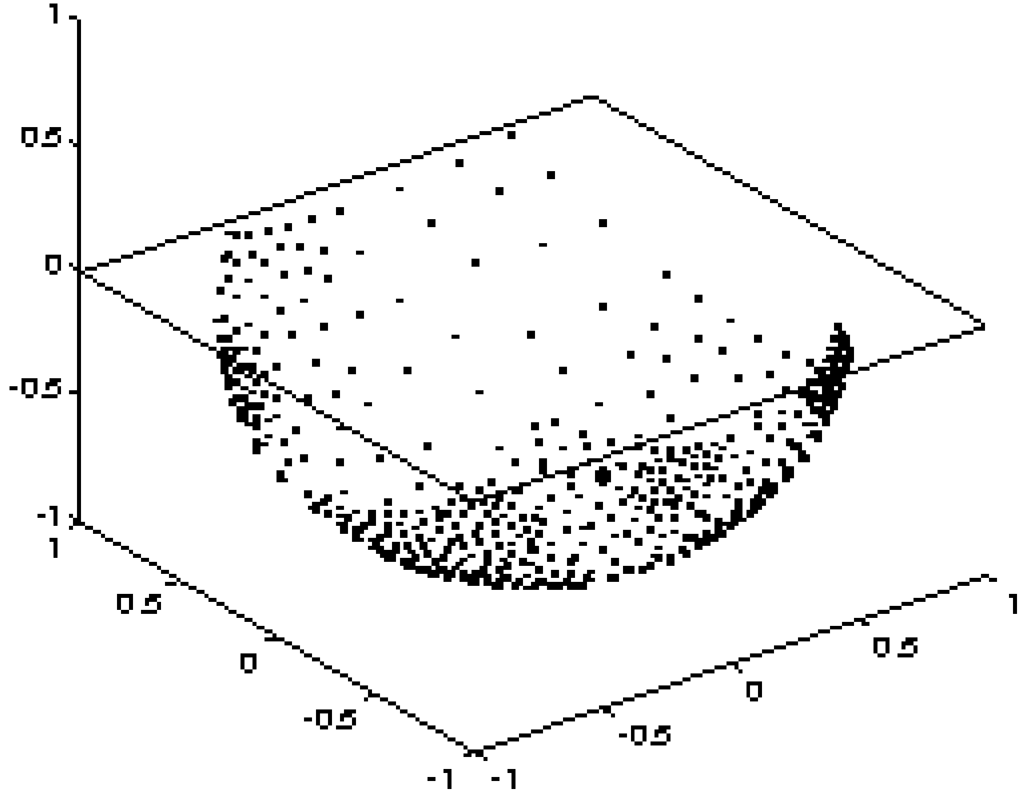
Random Sphere Algorithm (Gilbert)

Mesh Points in the Plane



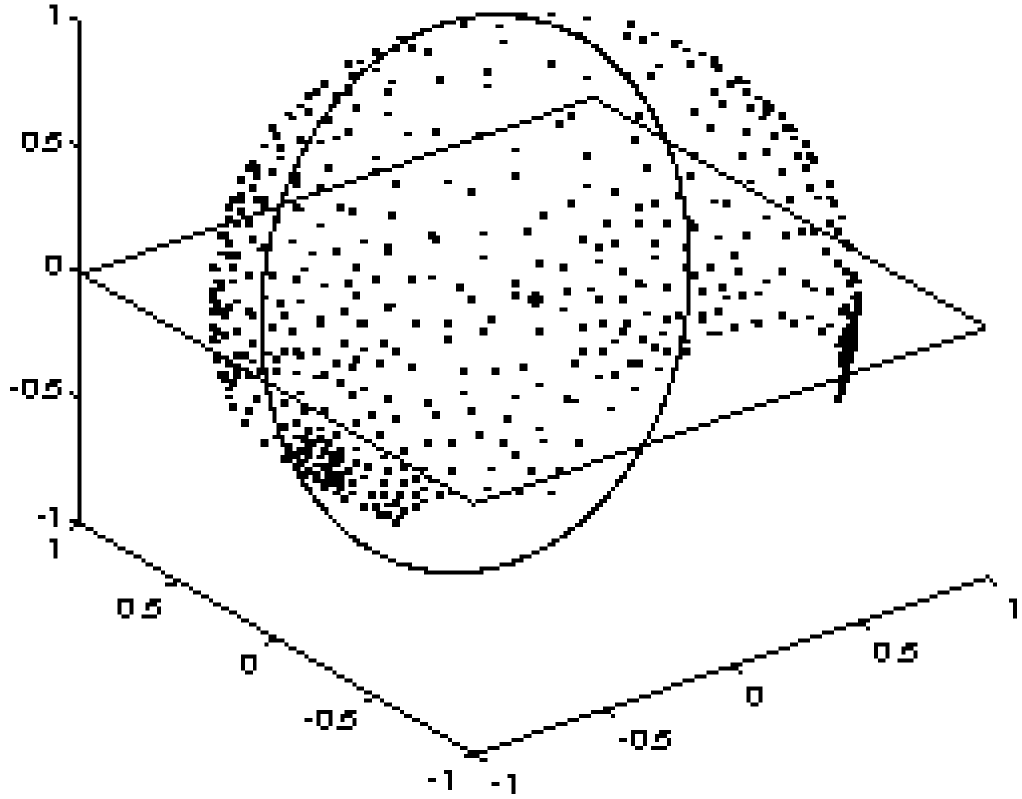
Random Sphere Algorithm (Gilbert)

Points Projected onto the Sphere

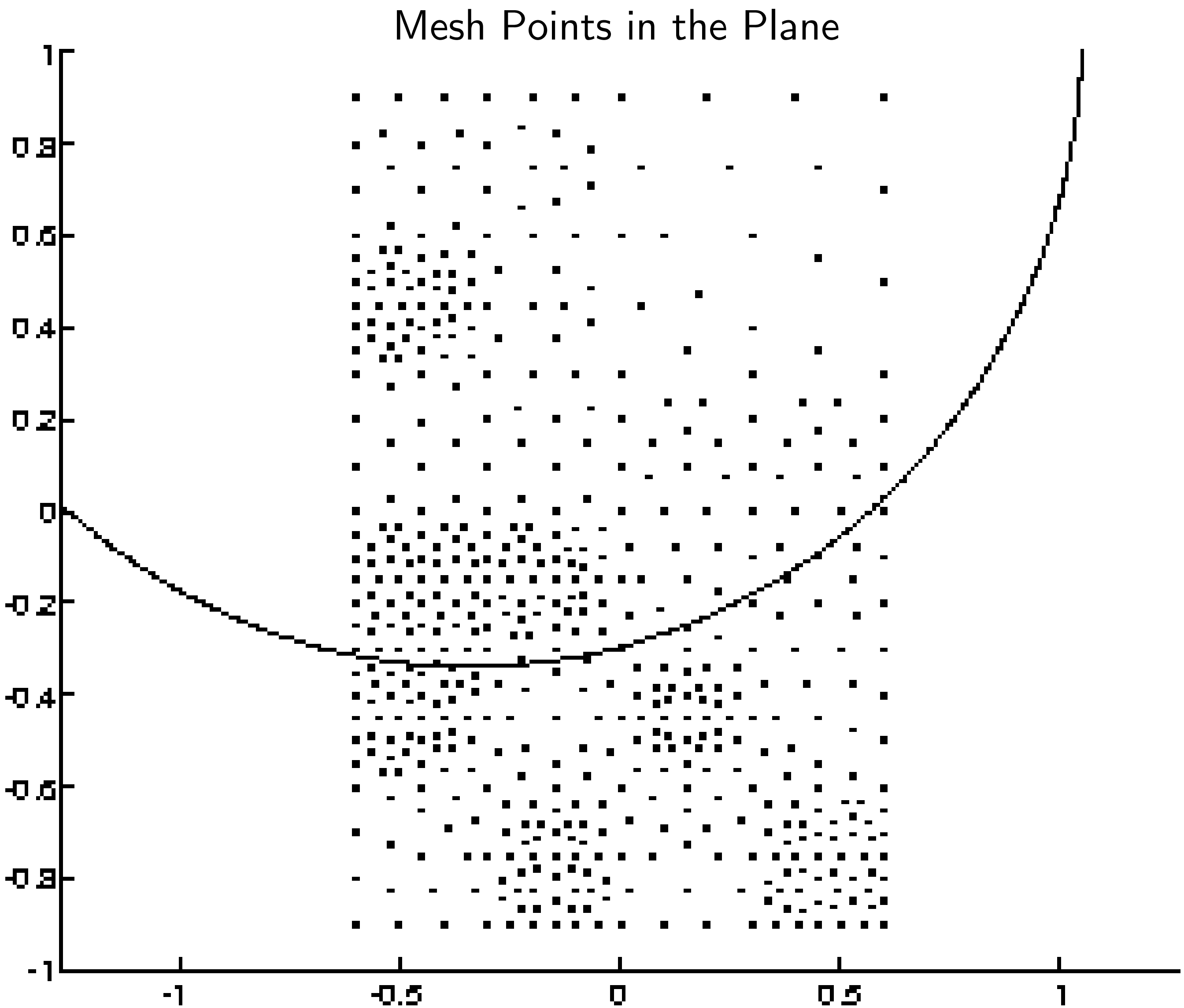


Random Sphere Algorithm (Gilbert)

Conformally Mapped Projected Points

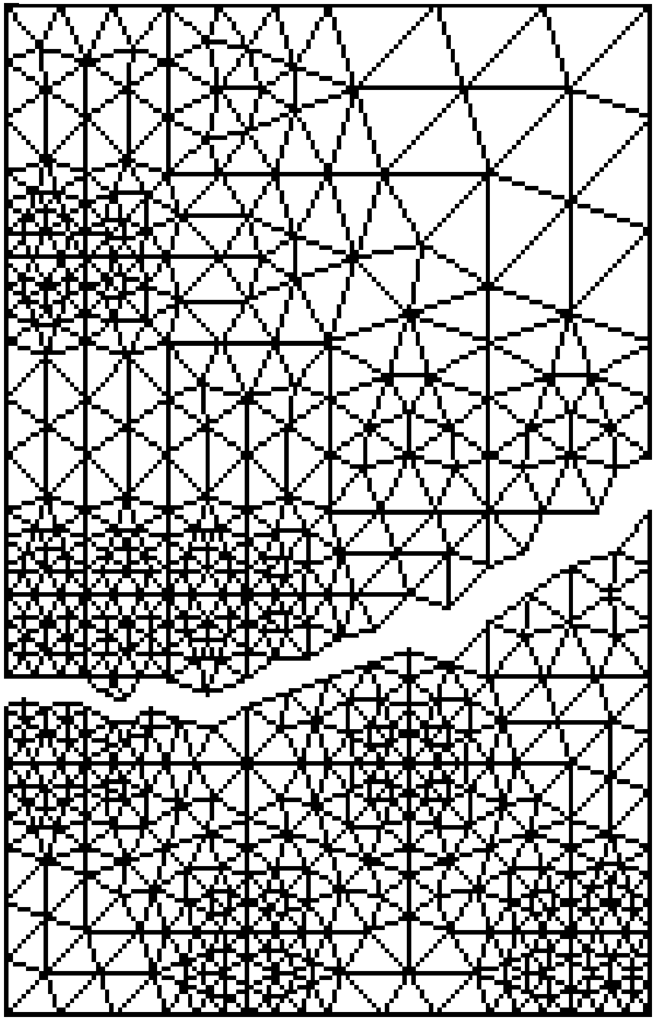


Random Sphere Algorithm (Gilbert)



Random Sphere Algorithm (Gilbert)

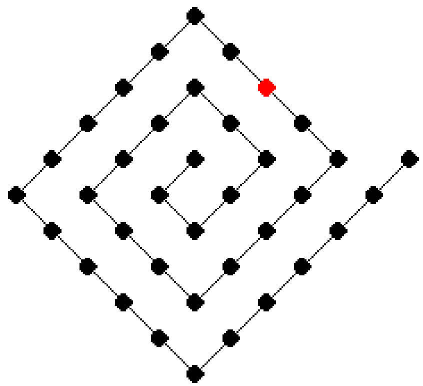
Partition of the Original Mesh



42 cut edges

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:

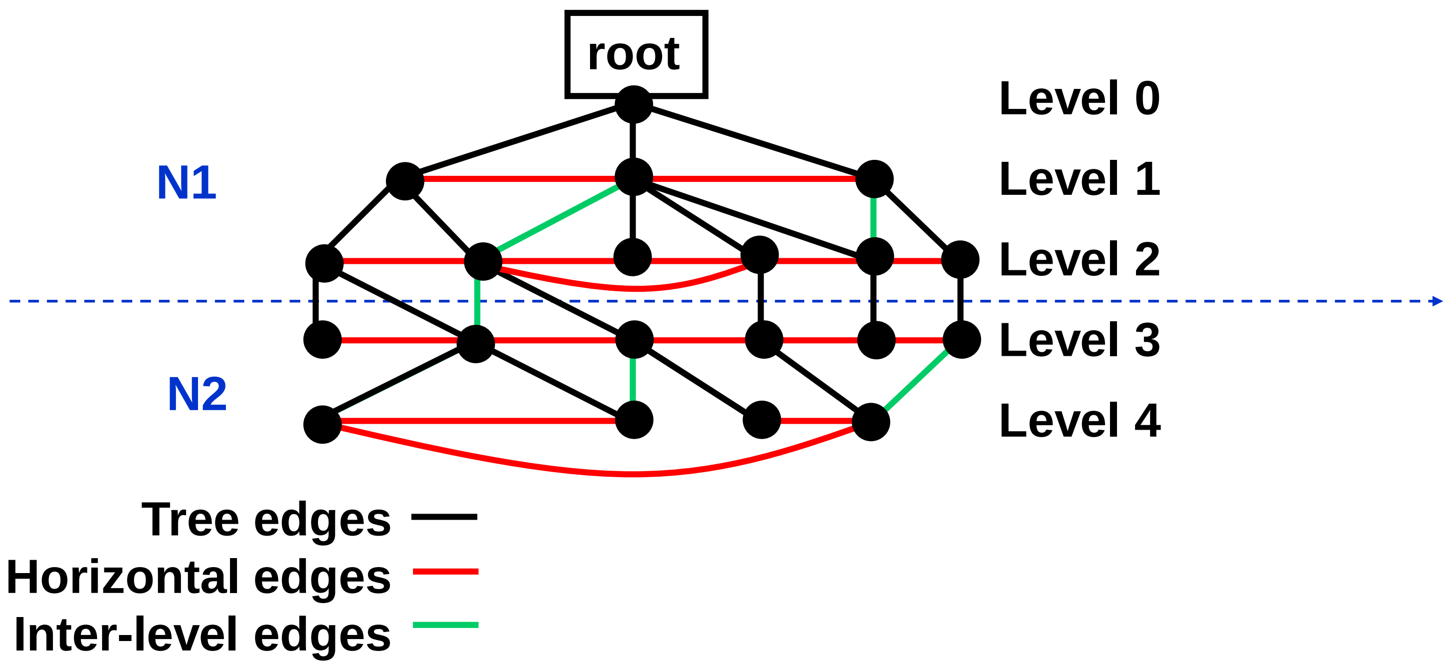


Outline

- 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
- Graph algorithms in sparse direct methods

Coordinate-Free: Breadth First Search (BFS)

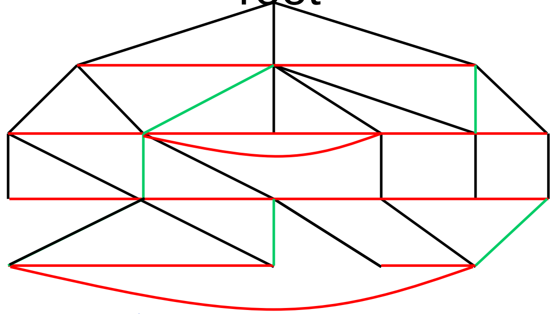
- Given $G = (N, E)$ and a root node r in N, BFS produces
  - **A subgraph T of G (same nodes, subset of edges)**
  - **T is a tree rooted at r**
  - **Each node assigned a level = distance from r**

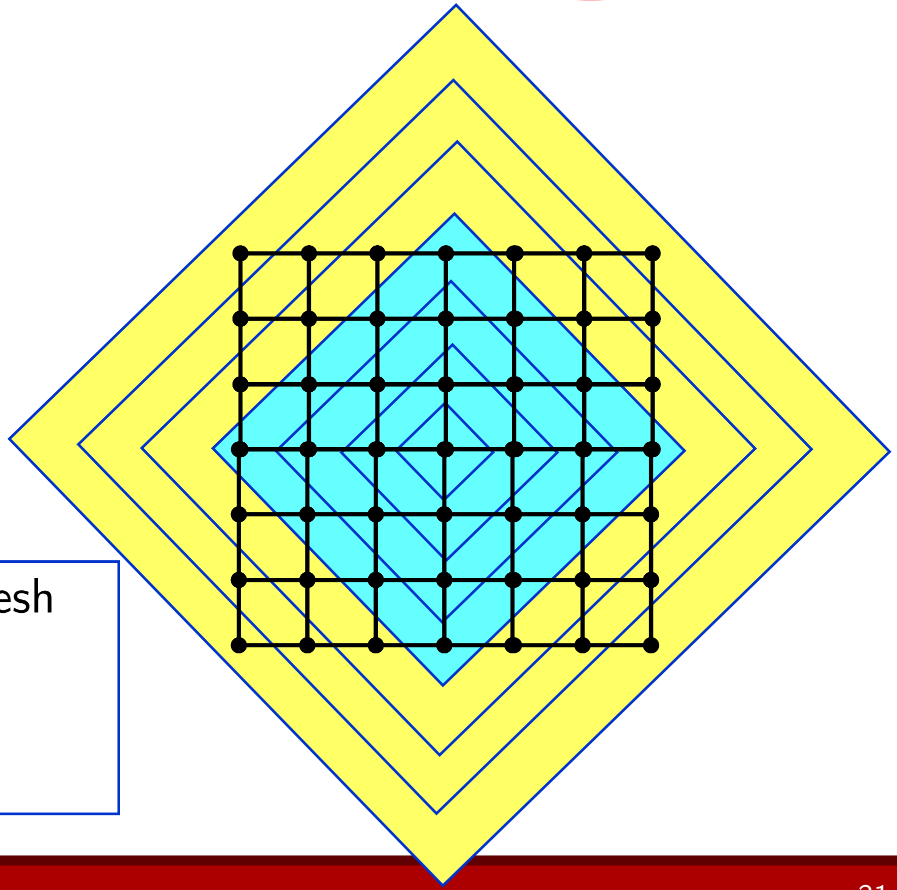


Partitioning via Breadth First Search

- 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
  - $N = N_1 \cup N_2$, where
    - $N_1 = \{\text{nodes at level} \leq L\}$,
    - $N_2 = \{\text{nodes at level} > L\}$
  - Choose L so $|N_1|$ close to $|N_2|$





BFS partition of a 2D Mesh
using center as root:
N_1 = levels 0, 1, 2, 3
N_2 = levels 4, 5, 6

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 $N = A \cup B$, where $|A| = |B|$
  - T = cost(A, B) = $\sum\{W(e)$ 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

- $T = \text{cost}(A, B)$, 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 $= (A - \{a\}) \cup \{b\}$, newB $= (B - \{b\}) \cup \{a\}$
- newT $= T - \big(D(a) + D(b) - 2w(a, b)\big) \equiv T -$ gain(a, b)
  - gain(a, b) measures improvement from swapping a and b
- Update formulas
  - newD$(a') = D(a') + 2w(a', a) - 2w(a', b)$ for a' in A, $a' \neq a$
  - newD$(b') = D(b') + 2w(b', b) - 2w(b', a)$ for b' in B, $b' \neq b$

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 ($|N| \leq 360$) 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:

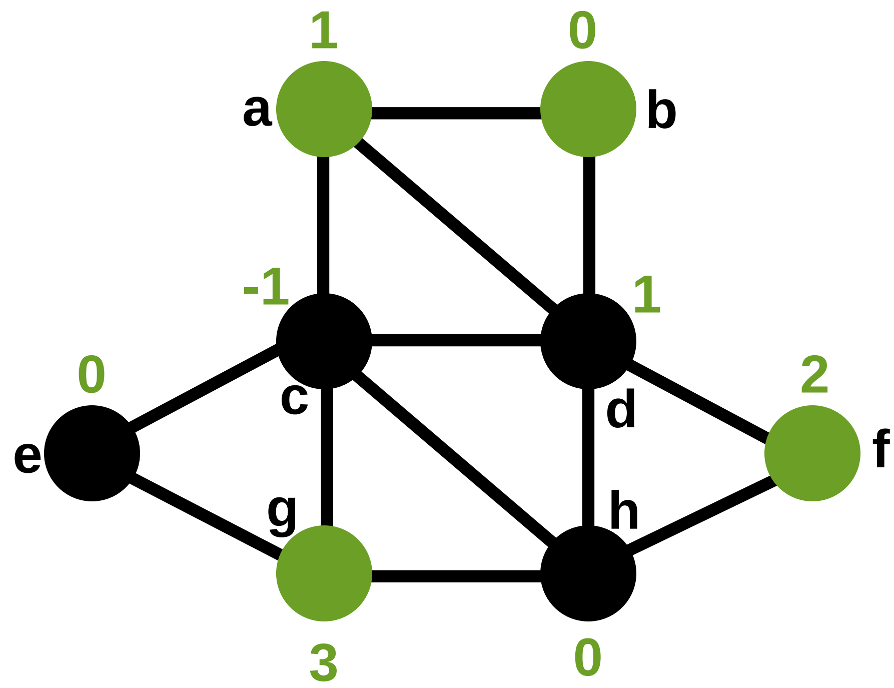
Buluç, A., Meyerhenke, H., Safro, I., Sanders, P., & Schulz, C. (2016). Recent advances in graph partitioning. In Algorithm Engineering.

Simplified Fiduccia-Mattheyses: Example (1)

Green 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 green.



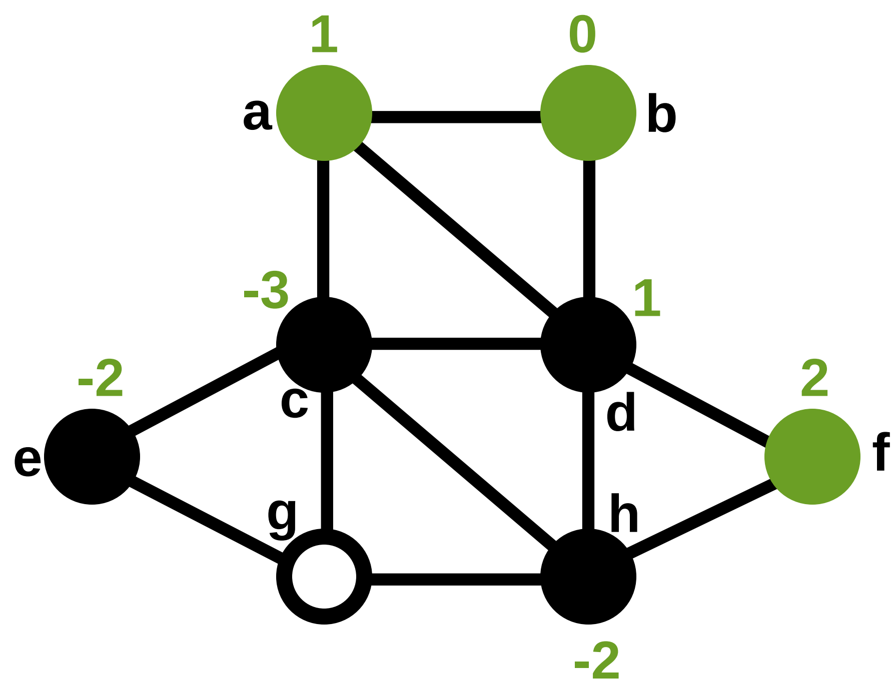
Nodes tentatively moved (and cut size after each pair):

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.



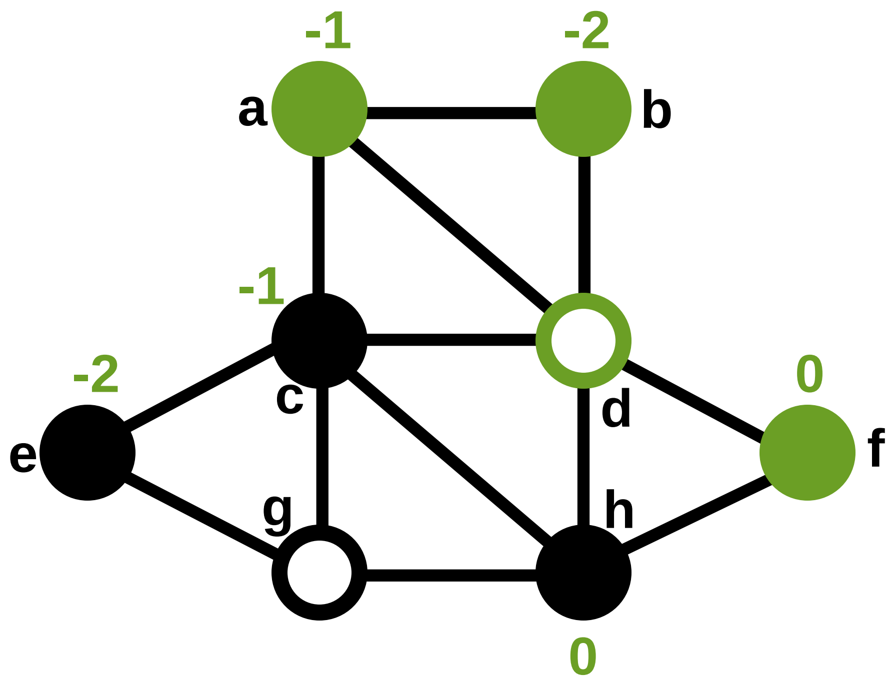
Nodes tentatively moved (and cut size after each pair):

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.

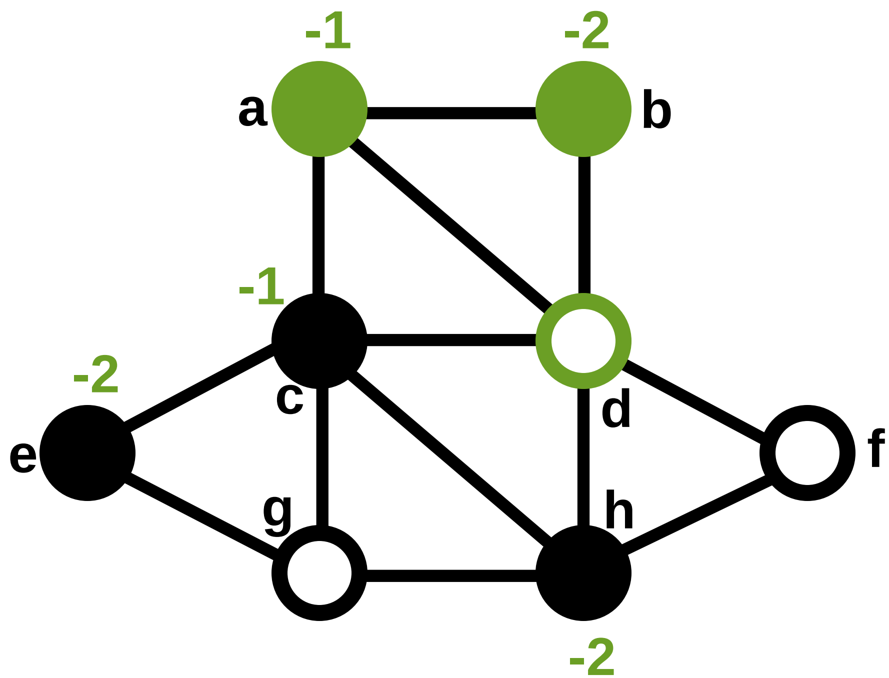


Nodes tentatively moved (and cut size after each pair):

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.



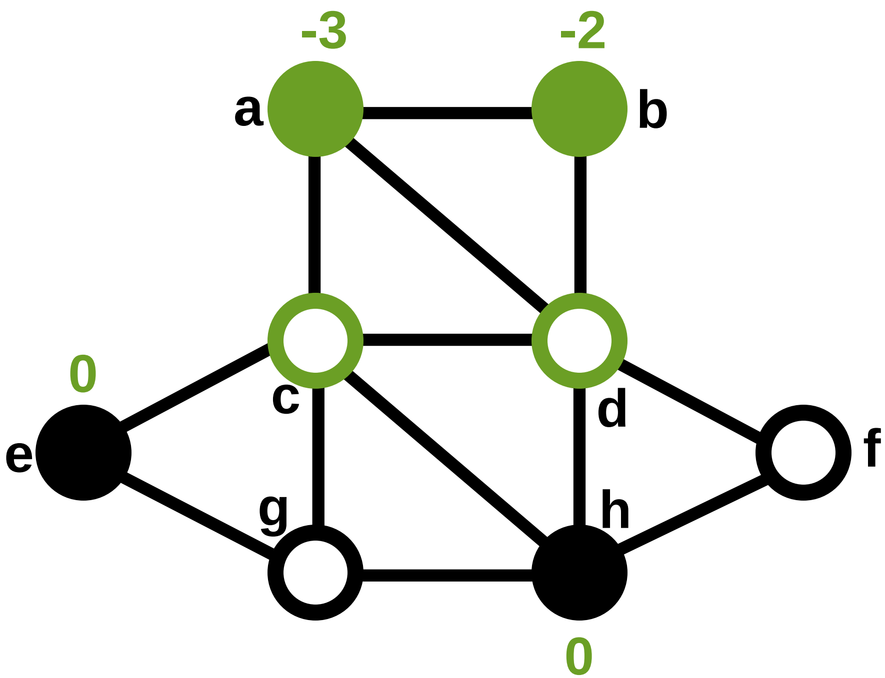
Nodes tentatively moved (and cut size after each pair):

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.

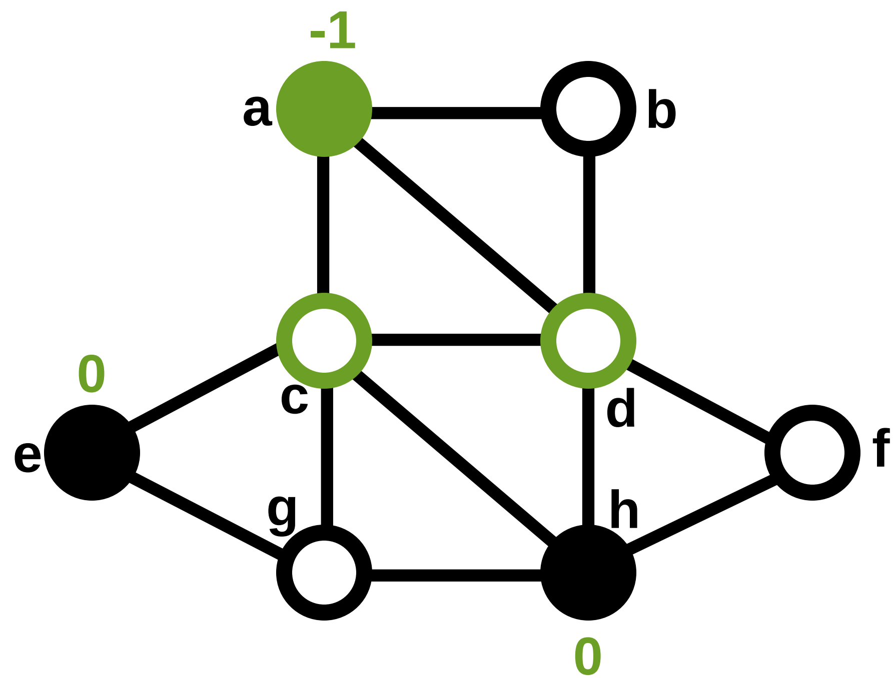


Nodes tentatively moved (and cut size after each pair):

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.



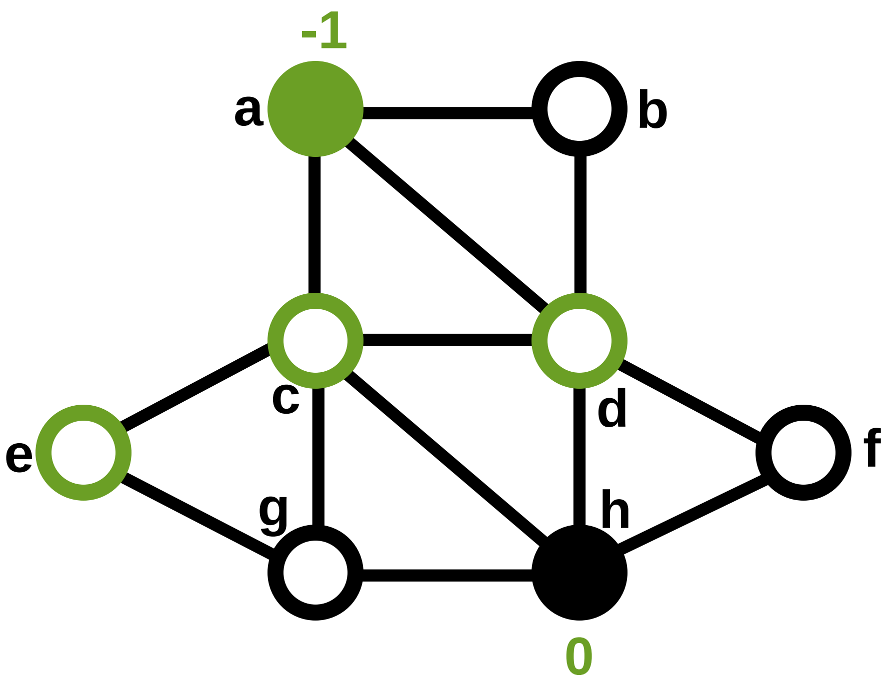
Nodes tentatively moved (and cut size after each pair):

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.

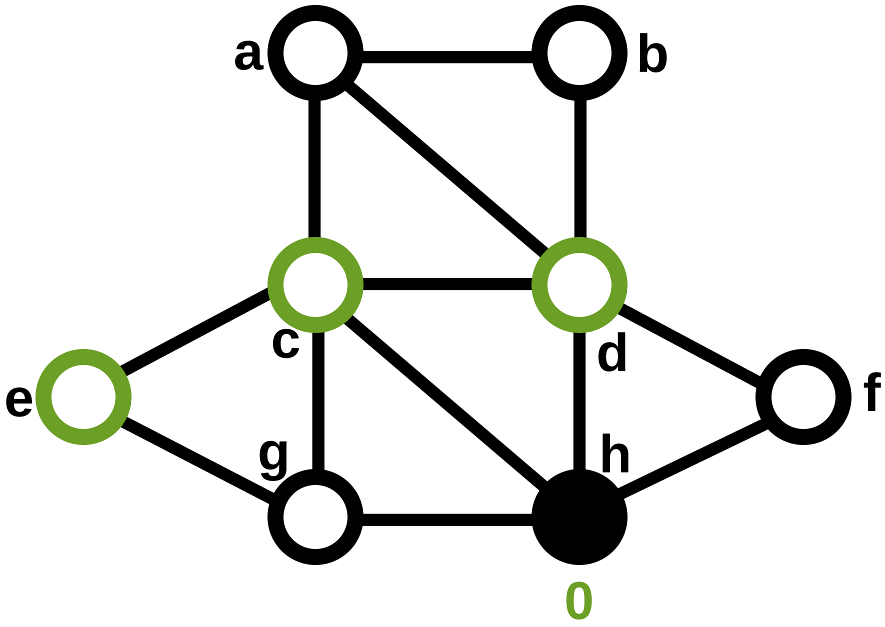


Nodes tentatively moved (and cut size after each pair):

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.



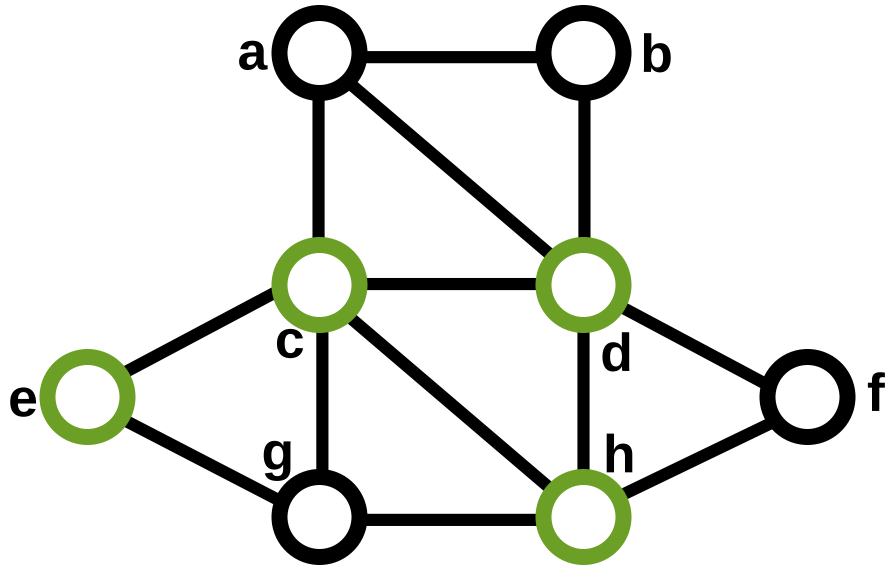
Nodes tentatively moved (and cut size after each pair):

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.



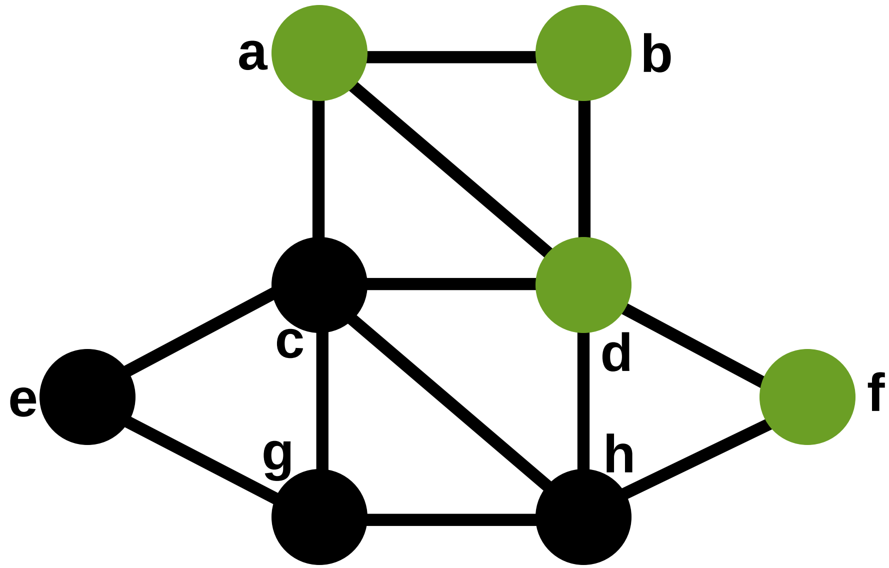
Nodes tentatively moved (and cut size after each pair):

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):

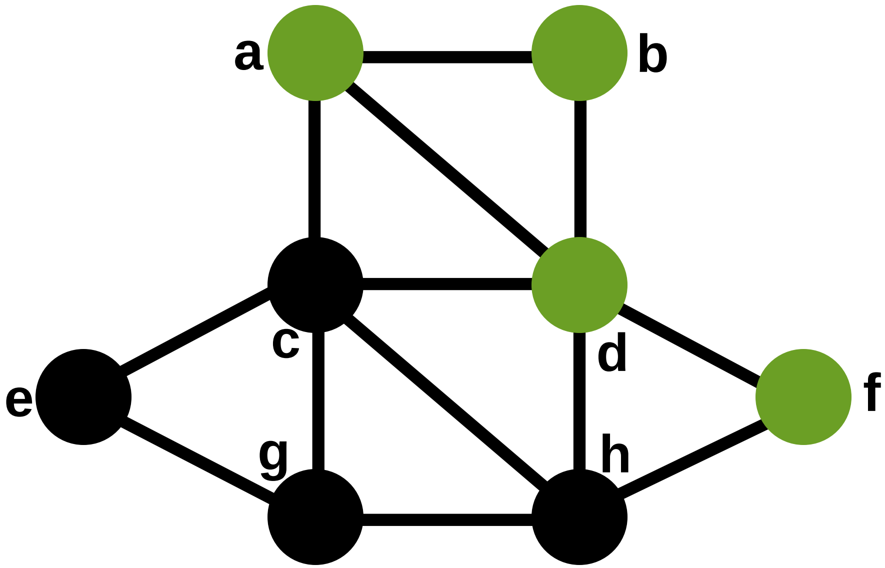
none (8); **g, d** **(4);** f, c (5); b, e (7); a, h (8)

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.

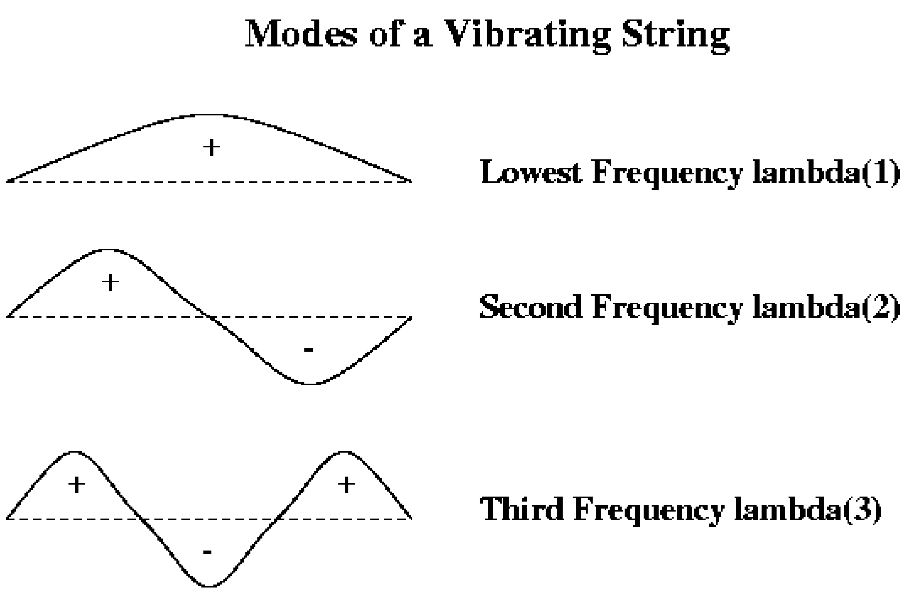


Coordinate-Free: Spectral Bisection

- 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)

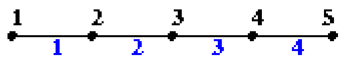


Basic Definitions

- *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 $i \neq j$ and there is an edge (i, j)
  - $L(G)(i, j) = 0$ otherwise

Example of In(G) and L(G) for Simple Meshes

Graph G

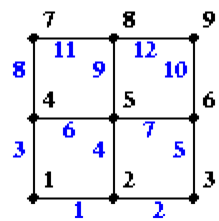


Incidence Matrix $In(G)$

$$\begin{array}{cc} & \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} & \begin{bmatrix} -1 & & & \\ 1 & -1 & & \\ & 1 & -1 & \\ & & 1 & -1 \\ & & & 1 \end{bmatrix} \end{array}$$

Laplacian Matrix $L(G)$

$$\begin{array}{cc} & \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} & \begin{bmatrix} 1 & -1 & & & \\ -1 & 2 & -1 & & \\ & -1 & 2 & -1 & \\ & & -1 & 2 & -1 \\ & & & -1 & 1 \end{bmatrix} \end{array}$$



$$\begin{array}{cc} & \begin{array}{cccccccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 \end{array} \\ \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \end{array} & \begin{bmatrix} -1 & & 1 & & & & & & & & & \\ 1 & -1 & & 1 & & & & & & & & \\ & 1 & & & 1 & & & & & & & \\ & & -1 & & & -1 & & 1 & & & & \\ & & & -1 & & 1 & -1 & & 1 & & & \\ & & & & -1 & & 1 & & & 1 & & \\ & & & & & & & -1 & & & -1 & \\ & & & & & & & & -1 & & 1 & -1 \\ & & & & & & & & & -1 & & 1 \end{bmatrix} \end{array}$$

$$\begin{array}{cc} & \begin{array}{ccccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \end{array} \\ \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \end{array} & \begin{bmatrix} 2 & -1 & & -1 & & & & & \\ -1 & 3 & -1 & & -1 & & & & \\ & -1 & 2 & & & -1 & & & \\ -1 & & & 3 & -1 & & -1 & & \\ & -1 & & -1 & 4 & -1 & & -1 & \\ & & -1 & & -1 & 3 & & & -1 \\ & & & -1 & & & 2 & -1 & \\ & & & & -1 & & -1 & 3 & -1 \\ & & & & & -1 & & -1 & 2 \end{bmatrix} \end{array}$$

Nodes numbered in black

Edges numbered in blue

Properties of Laplacian Matrix

- *Theorem 1:* Given G, $L(G)$ has the following properties
  - $L(G)$ is symmetric.
    - This means the eigenvalues of $L(G)$ are real and its eigenvectors are real and orthogonal.
  - $In(G) \times \left(In(G)\right)^T = L(G)$
  - The eigenvalues of $L(G)$ are nonnegative:
    - $0 = \lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n$
  - 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? 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.
  - 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.

Details for Vibrating String Analogy

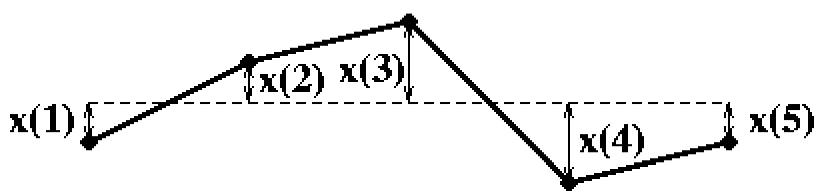
- Force on mass j $= k[x(j-1) - x(j)] + k[x(j+1) - x(j)]$

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

- $F = ma$ yields $m \cdot x''(j) = -k[-x(j-1) + 2x(j) - x(j+1)]$ $(*)$
- Writing $(*)$ for $j = 1,2,\dots,n$ yields

$$m = \frac{d^2}{dx^2}\begin{bmatrix} x(1) \\ x(2) \\ \vdots \\ x(j) \\ \vdots \\ x(n) \end{bmatrix} = -k\begin{bmatrix} 2x(1) - x(2) \\ -x(1) + 2x(2) - x(3) \\ \vdots \\ -x(j-1) + 2x(j) - x(j+1) \\ \vdots \\ -x(n-1) + 2x(n) \end{bmatrix} = -k\begin{bmatrix} 2 & -1 & & \\ -1 & 2 & \ddots & \\ & \ddots & \ddots & -1 \\ & & -1 & 2 \end{bmatrix}\begin{bmatrix} x(1) \\ x(2) \\ \vdots \\ x(j) \\ \vdots \\ x(n) \end{bmatrix} = -kL\begin{bmatrix} x(1) \\ x(2) \\ \vdots \\ x(j) \\ \vdots \\ x(n) \end{bmatrix}$$

$$(-\boldsymbol{m}/\boldsymbol{k})\,\boldsymbol{x}'' = \boldsymbol{L}\boldsymbol{x}$$



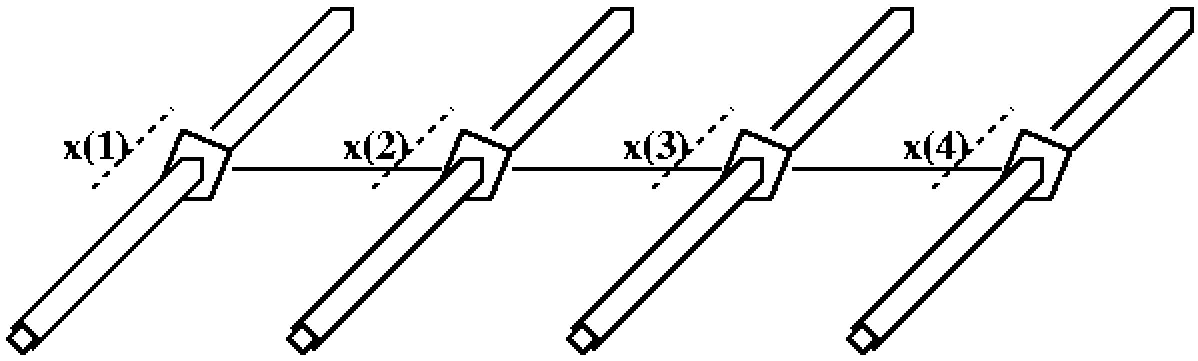
Details for Vibrating String (continued)

- $-(m/k)\, x'' = Lx$, where $x = [x_1, x_2, \dots, x_n]^T$
- Seek solution of form $x(t) = \sin(\alpha t)\, x_0$
  - $Lx_0 = (m/k)\alpha^2 x_0 = \lambda x_0$
  - For each integer i, get $\lambda = 2(1 - \cos(i\pi/(n+1))$, $x_0 = \begin{bmatrix} \sin(1i\pi/(n+1)) \\ \sin(2i\pi/(n+1)) \\ \vdots \\ \sin(ni\pi/(n+1)) \end{bmatrix}$
  - Thus x_0 is a sine curve with frequency proportional to i
  - Thus $\alpha^2 = 2(k/m)(1 - \cos(i\pi/(n+1))$ or $\alpha \sim (k/m)^{1/2}\pi i/(n+1)$
- $L = \begin{pmatrix} 2 & -1 & & \\ -1 & 2 & -1 & \\ & \dots & & \\ & & -1 & 2 \end{pmatrix}$ not quite Laplacian of 1D mesh, but we can fix that …

Details for Vibrating String (continued)

- Write down F=ma for "vibrating string" below
- Get Graph Laplacian of 1D mesh

"Vibrating String" for Spectral Bisection

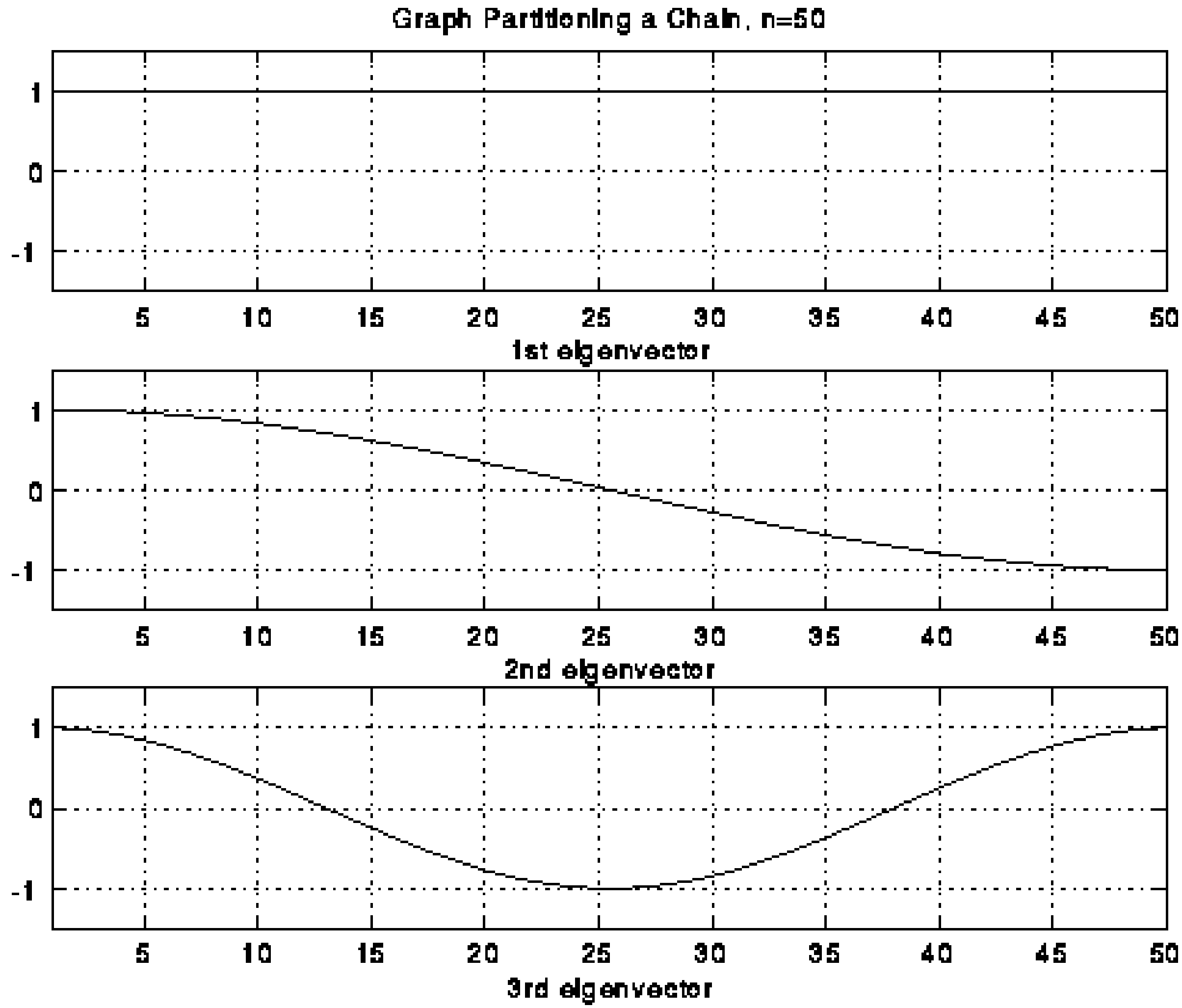


Eigenvectors of L (1D mesh)

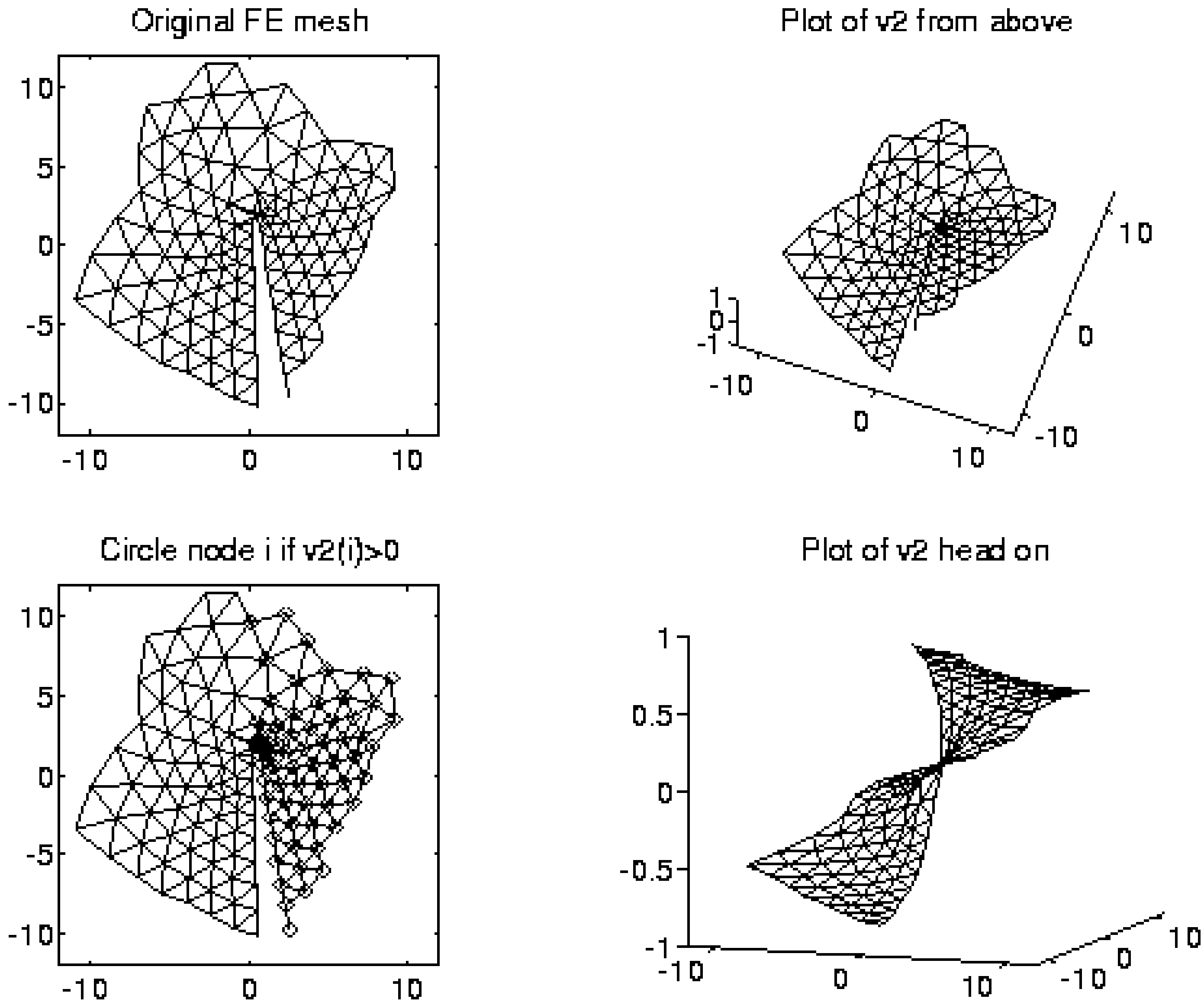
Eigenvector 1
(all ones)

Eigenvector 2

Eigenvector 3



2nd eigenvector of L (planar mesh)



Computing v_2 and λ_2 of $L(G)$ using Lanczos

- Given any $n \times n$ symmetric matrix A (such as $L(G)$) Lanczos computes a $k \times k$ "approximation" T by doing k matrix-vector products, $k \ll n$

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)
    $r = Aq(j)$ ... matrix vector multiplication, 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), expensive step
    $r = r - a(j)v(j)$ ... "axpy" (BLAS1)
    $b(j) = ||r||$ ... compute vector norm (BLAS1), expensive step
until convergence ... details omitted

$$
T = \begin{pmatrix}
a(1) & b(1) & & & & & 0 \\
b(1) & a(2) & b(2) & & & & \\
& b(2) & a(3) & b(3) & & & \\
& & \cdots & \cdots & \cdots & & \\
0 & & & & b(k-2) & a(k-1) & b(k-1) \\
& & & & & b(k-1) & a(k)
\end{pmatrix}
$$

- 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

- 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
- Graph algorithms in sparse direct methods

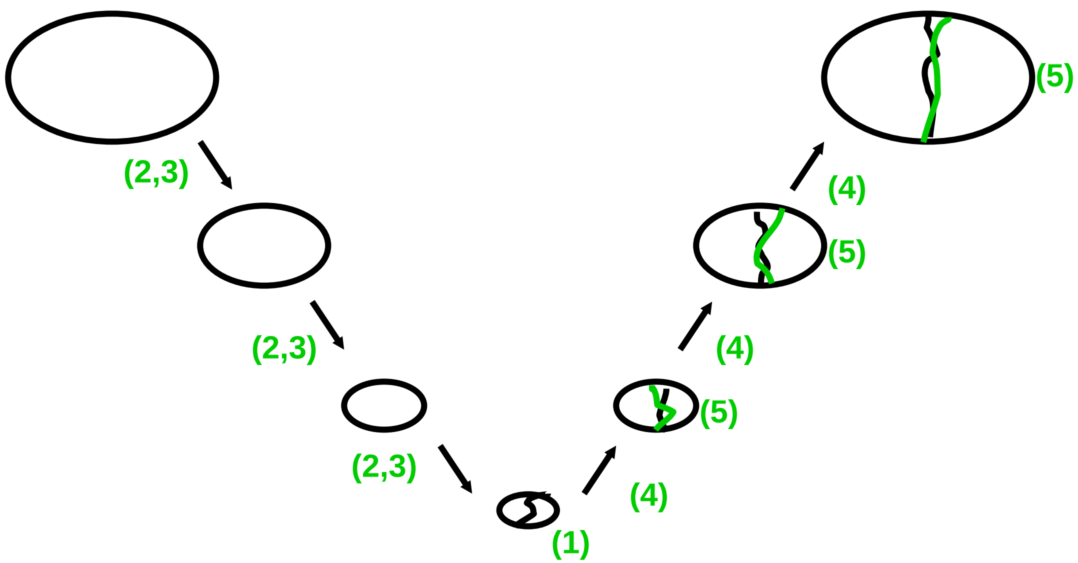
Introduction to Multilevel Partitioning

- 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

$(N+,N-) = \text{Multilevel\_Partition}(N,E)$
... recursive partitioning routine returns $N+$ and $N-$ where $N = N+ \cup N-$
if $|N|$ is small
(1) Partition $G = (N,E)$ directly to get $N = N+ \cup N-$
Return $(N+, N-)$
else
(2) Coarsen G to get an approximation $G_c = (N_c, E_c)$
(3) $(N_c+ , N_c-) = \text{Multilevel\_Partition}(N_c, E_c)$
(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

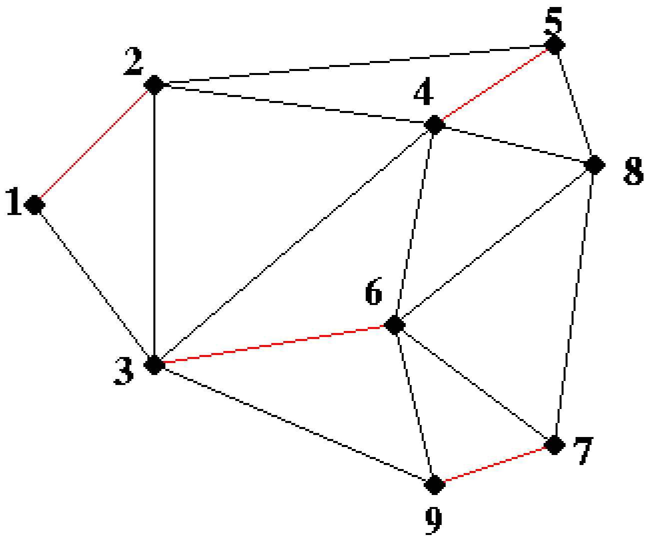
- Coarsen graph and expand partition using maximal matchings
- Improve partition using Kernighan-Lin

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:

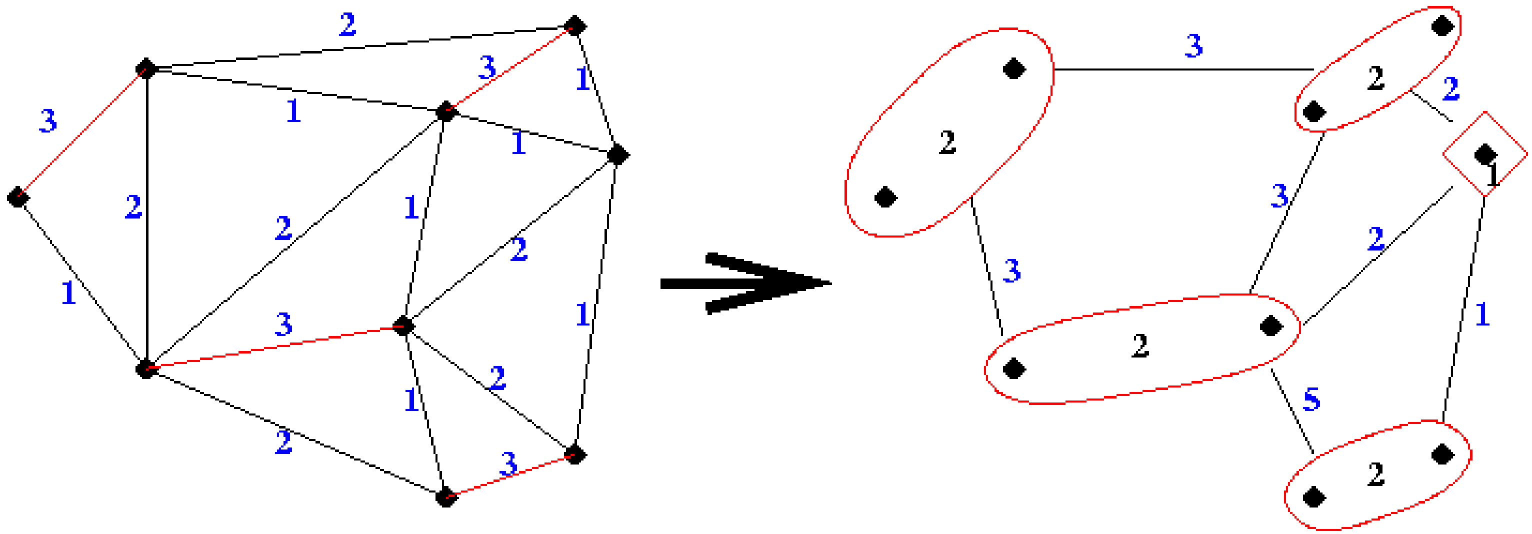
```
let E_m be empty
mark all nodes in N as unmatched
for i = 1 to |N|     ... visit the nodes in any order
    if i has not been matched
        mark i as matched
        if there is an edge e = (i, j) where j is also unmatched,
            add e to E_m
            mark j as matched
        endif
 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

Maximal Matching: Example



Example of Coarsening

How to coarsen a graph using a maximal matching



$G = (N, E)$

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 N_c 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

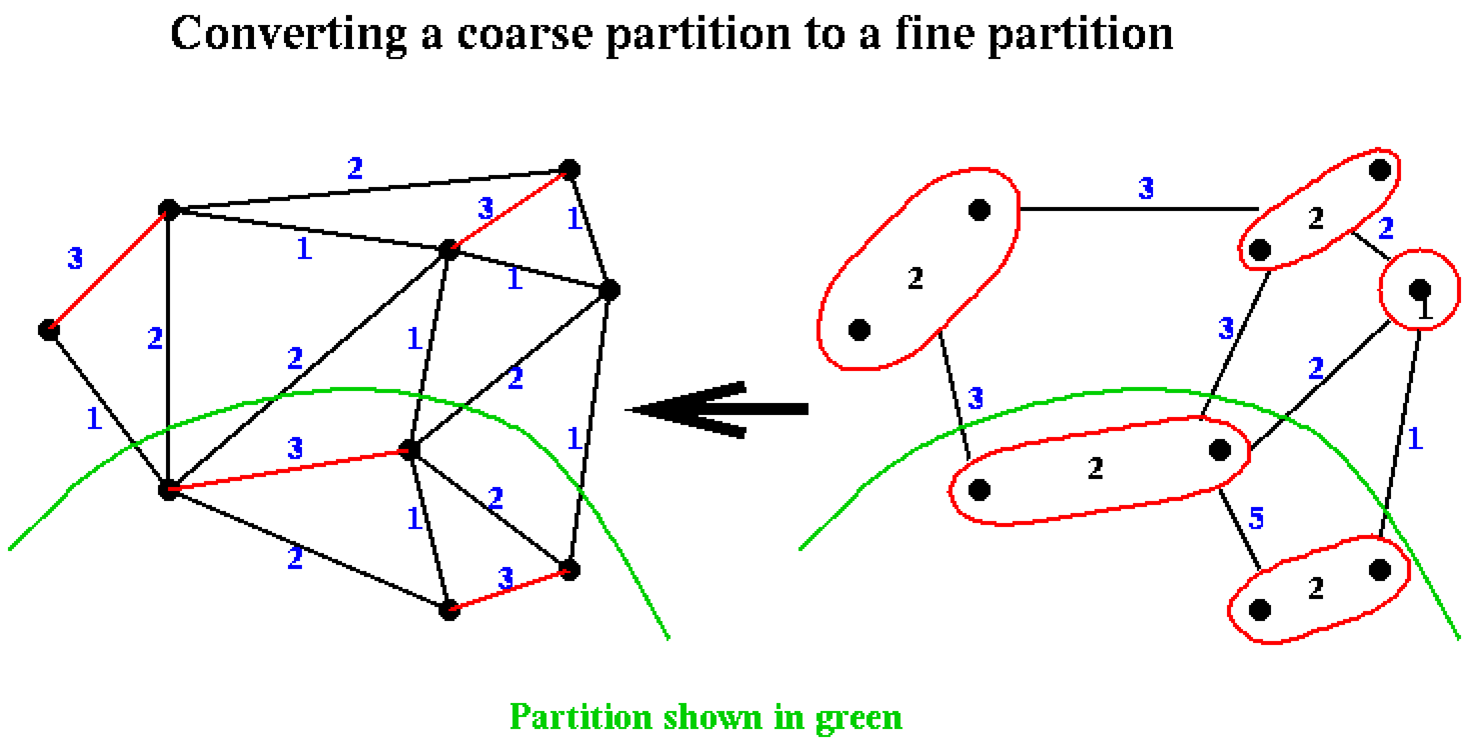
Put edge $ee = (n(e), n(r))$ in E_c

$W(ee) = W(e')$

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



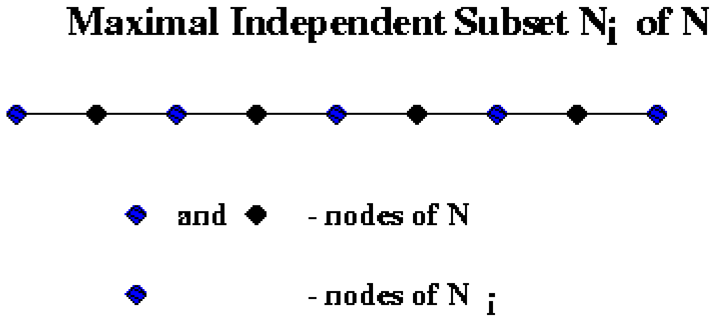
Multilevel Spectral Bisection

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

Maximal Independent Sets

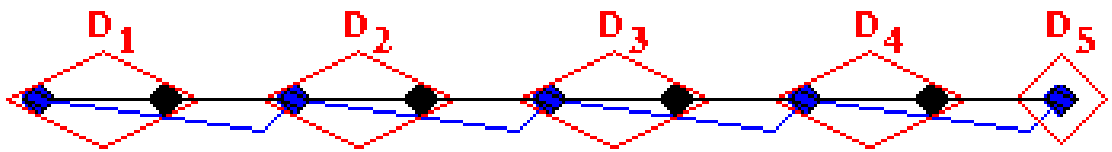
- *Definition*: An independent set of a graph $G = (N, E)$ is a subset N_i of N such that no two nodes in N_i are connected by an edge
- *Definition:* A maximal independent set (MIS) of a graph $G = (N, E)$ is an independent set N_i to which no more nodes can be added and remain an independent set
-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

let 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

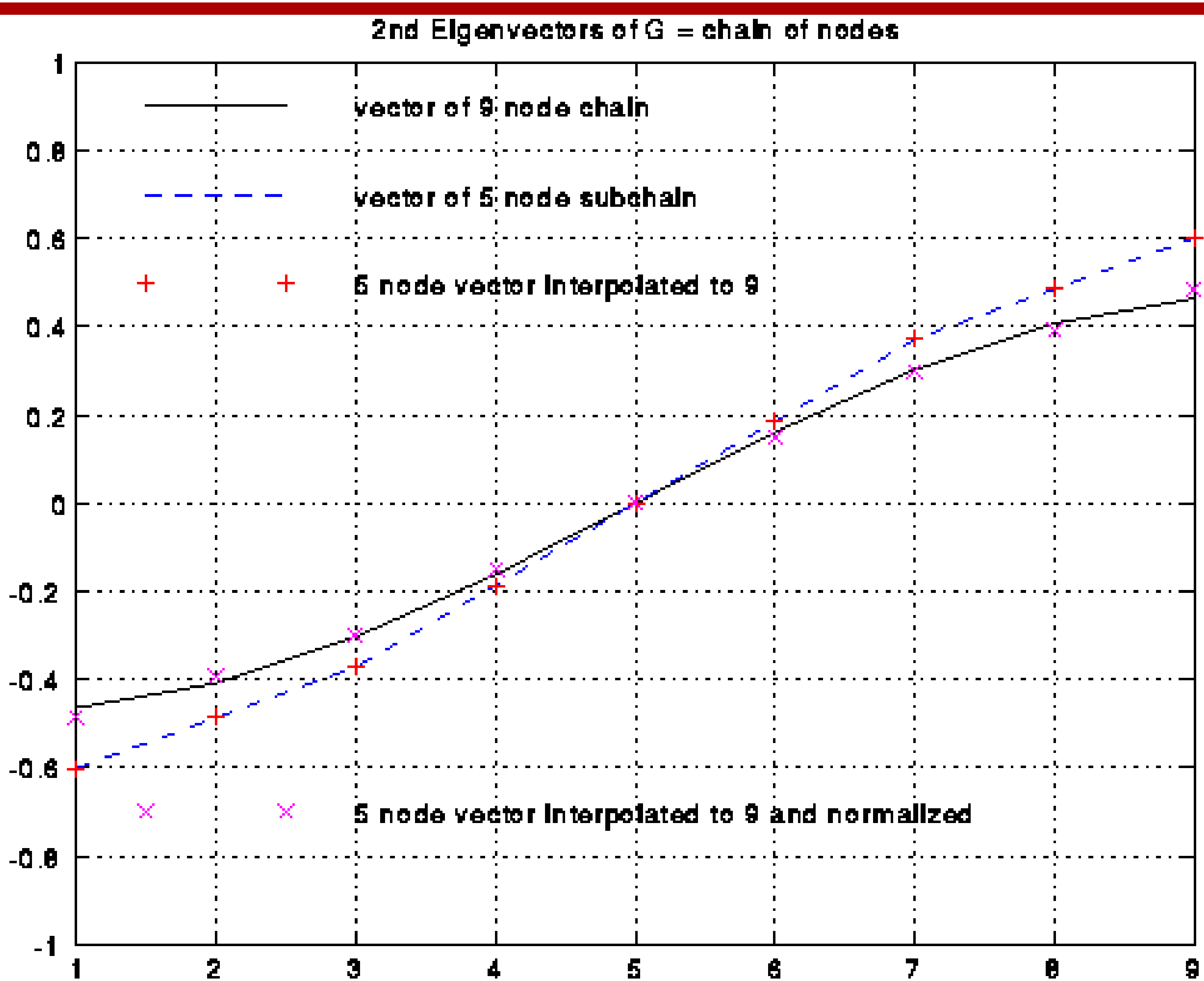
```
E_c = empty set
for all nodes k in N_i
    D(k) = ( {k}, empty set )
    ... first set contains nodes in D(k), second set contains edges in D(k)
unmark all edges in E
repeat
    choose an unmarked edge e = (k,j) from E
    if exactly one of k and j (say k) is in some D(m)
        mark e
        add 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

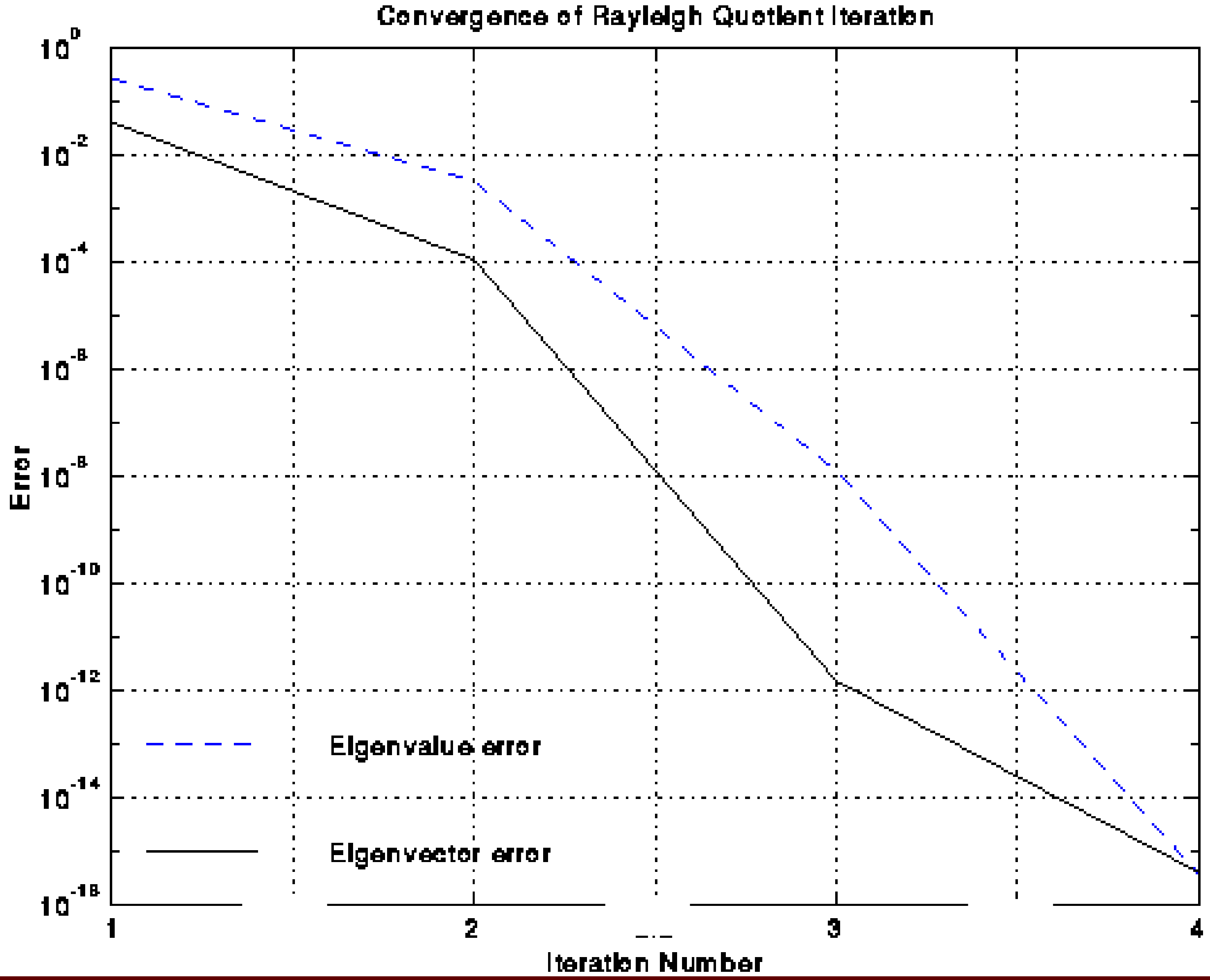
Example: 1D mesh of 9 nodes



Improve eigenvector: Rayleigh Quotient Iteration

$j = 0$
pick starting vector $v(0)$ ... from expanding v_c
repeat
  $j = j + 1$
  $r(j) = v^T(j-1) \cdot L(G) \cdot v(j-1)$
  ... $r(j) =$ *Rayleigh Quotient* of $v(j-1)$
  ... $=$ good approximate eigenvalue
  $v(j) = (L(G) - r(j) \cdot I)^{-1} \cdot v(j-1)$
  ... 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

- 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
- Graph algorithms in sparse direct methods

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/)

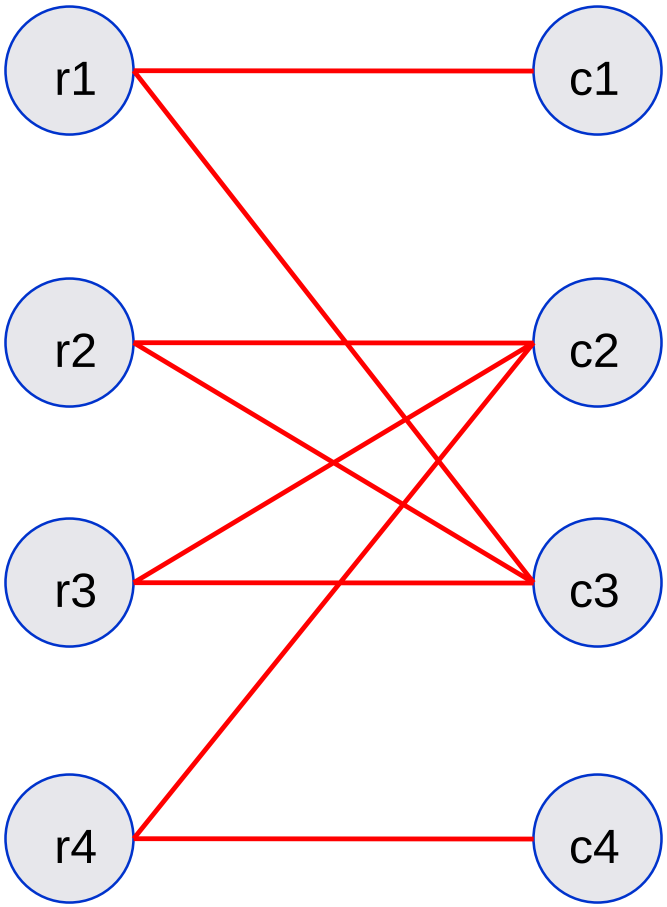
Outline

- 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
- Graph algorithms in sparse direct methods

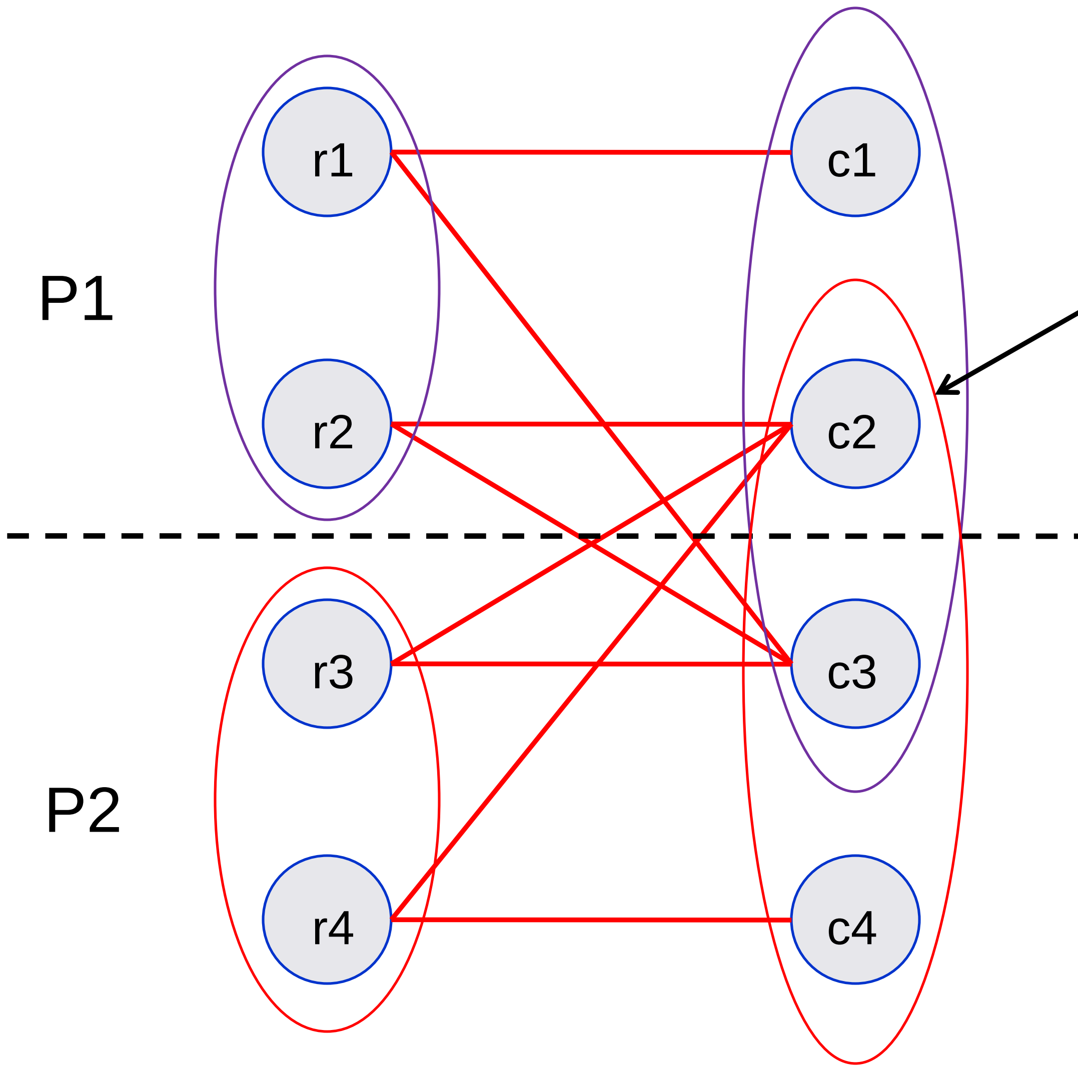
Beyond simple graph partitioning:

Representing a sparse matrix as a hypergraph

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



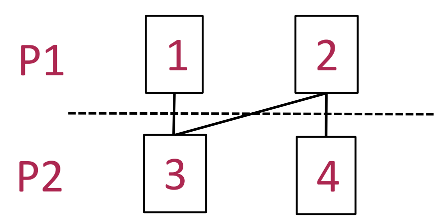
Using a graph to partition, versus a hypergraph



Source vector entries corresponding to c2 and c3 are needed by both partitions – so total volume of communication is 2

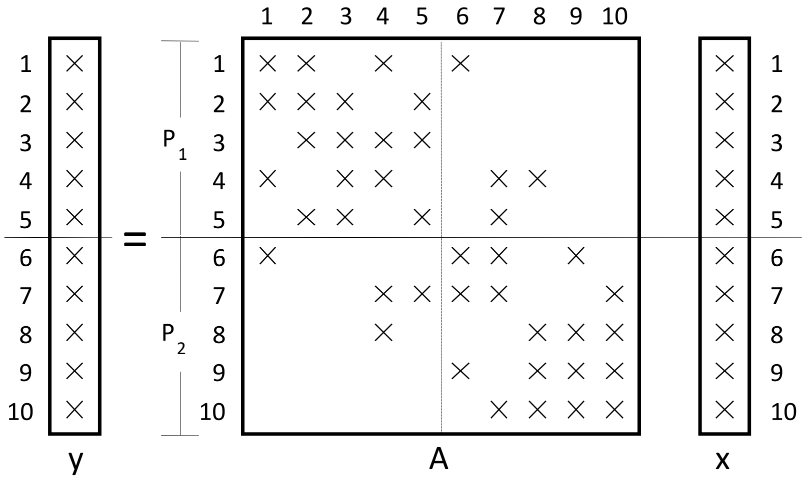
$$\begin{bmatrix} \times & 0 & \times & 0 \\ 0 & \times & \times & 0 \\ 0 & \times & \times & 0 \\ 0 & \times & 0 & \times \end{bmatrix}$$

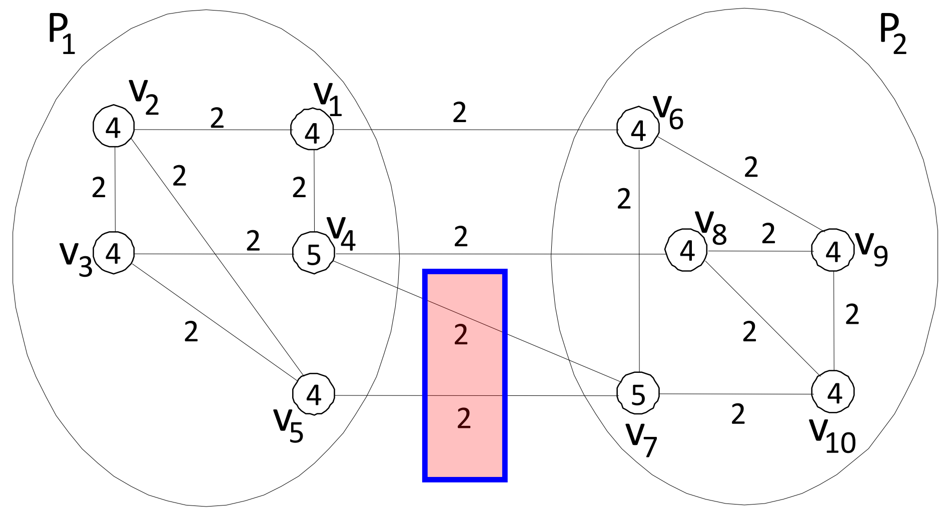
But graph cut is 3!



⇒ Cut size of graph partition may not accurately count communication volume

A sparse matrix in the graph model





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

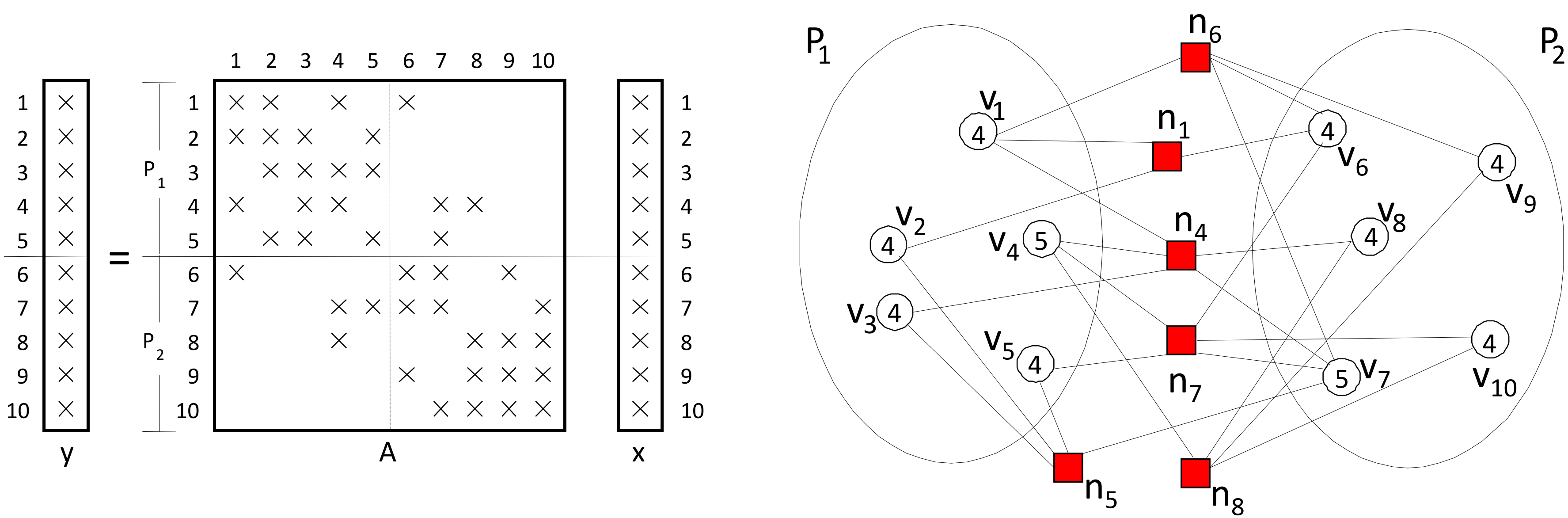
$$y(i) \leftarrow y(i) + A(i,j)\,x(j) \text{ and } y(j) \leftarrow y(j) + A(j,i)\,x(i)$$

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

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

$x(7)$ 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)

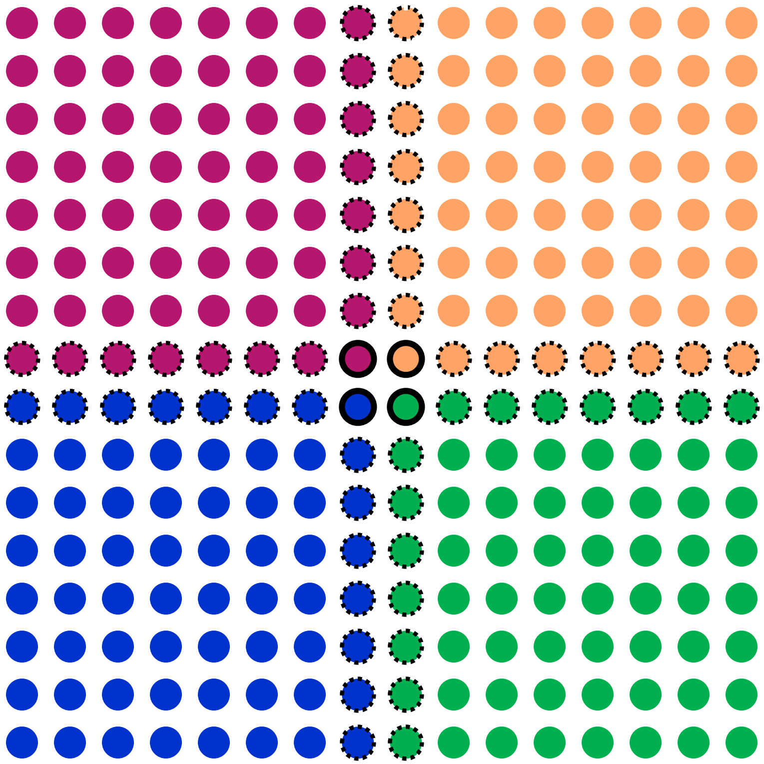
Each {vertex, net} pair represents unique nonzero net-cut metric:

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

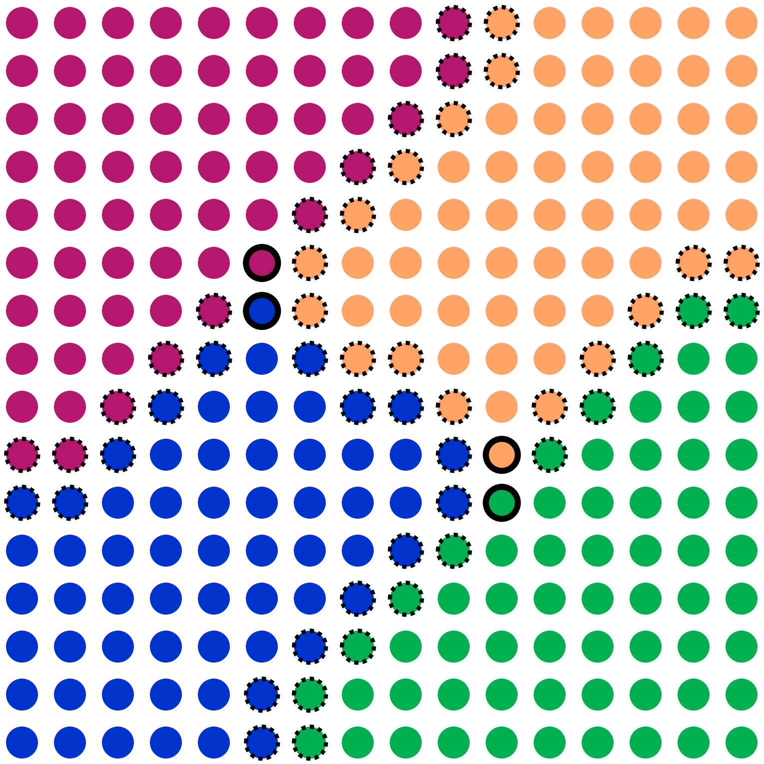
connectivity-1 metric: $\text{cutsize}(\Pi) = \sum_{n \in NE} w(n_i)(c(n_j) - 1)$

Two Different 2D Mesh Partitioning Strategies

Graph:
Cartesian Partitioning



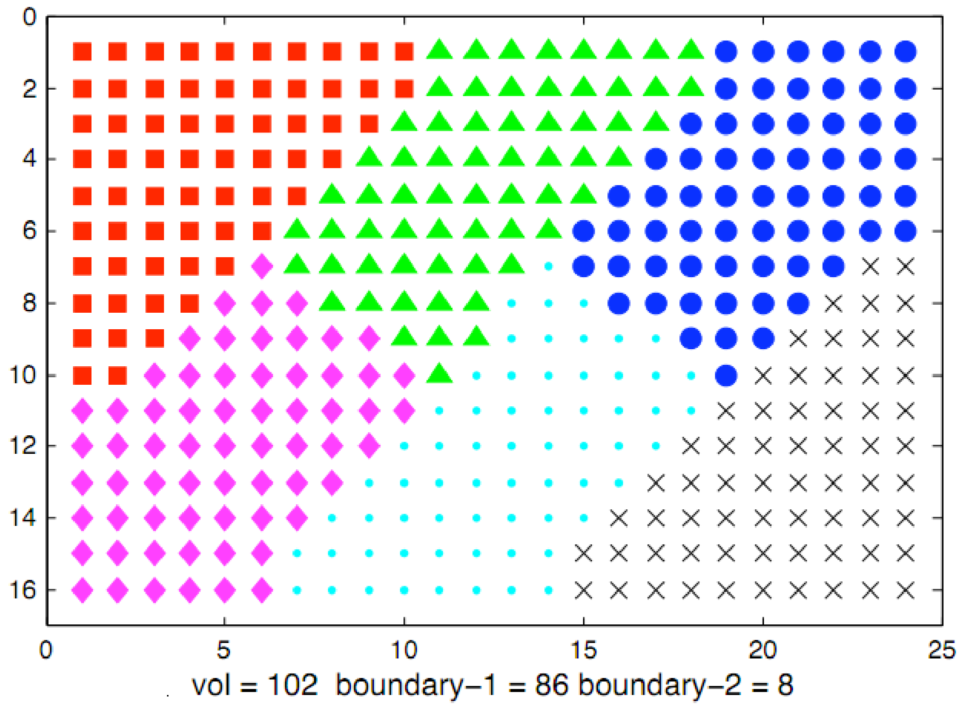
Hypergraph:
MeshPart Algorithm [Ucar, Catalyurek, 2010]



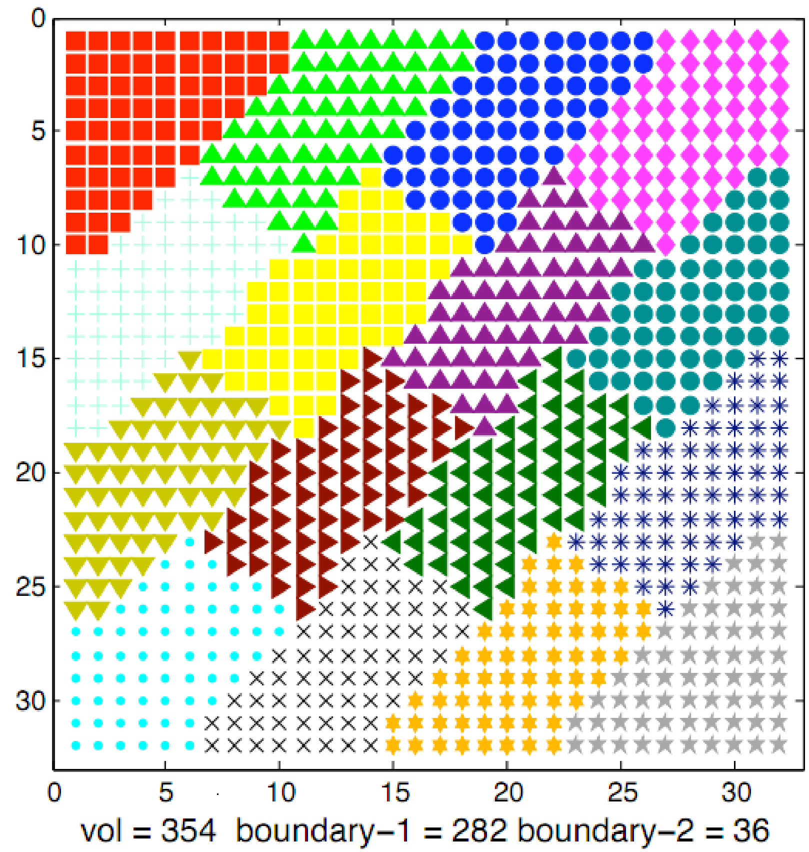
Total SpMV communication volume = 64

Total SpMV communication volume = 58

Generalization of the MeshPart Algorithm



(a) 2×3-way partitioning of the 16×24 mesh

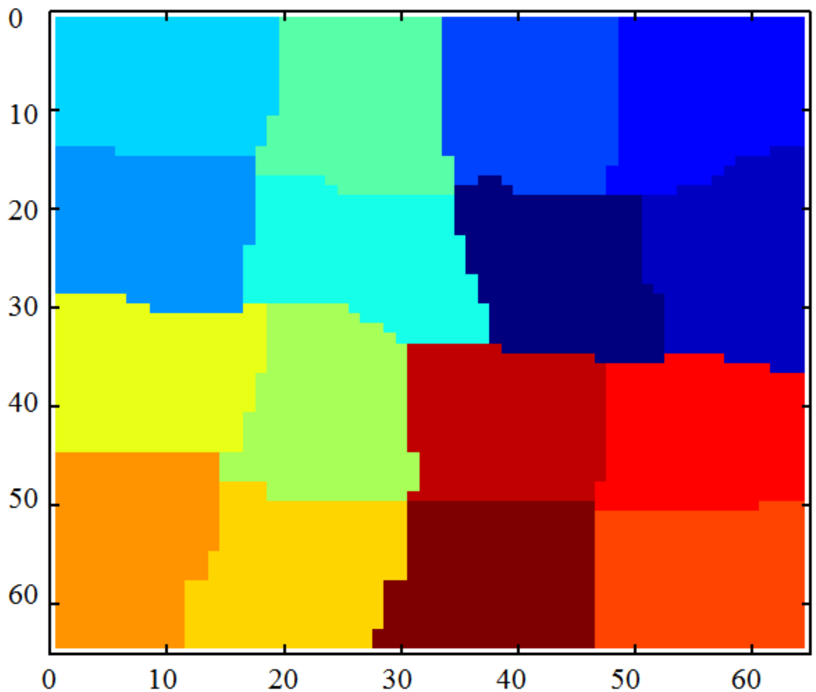


(c) 16-way partitioning of the 32×32 mesh

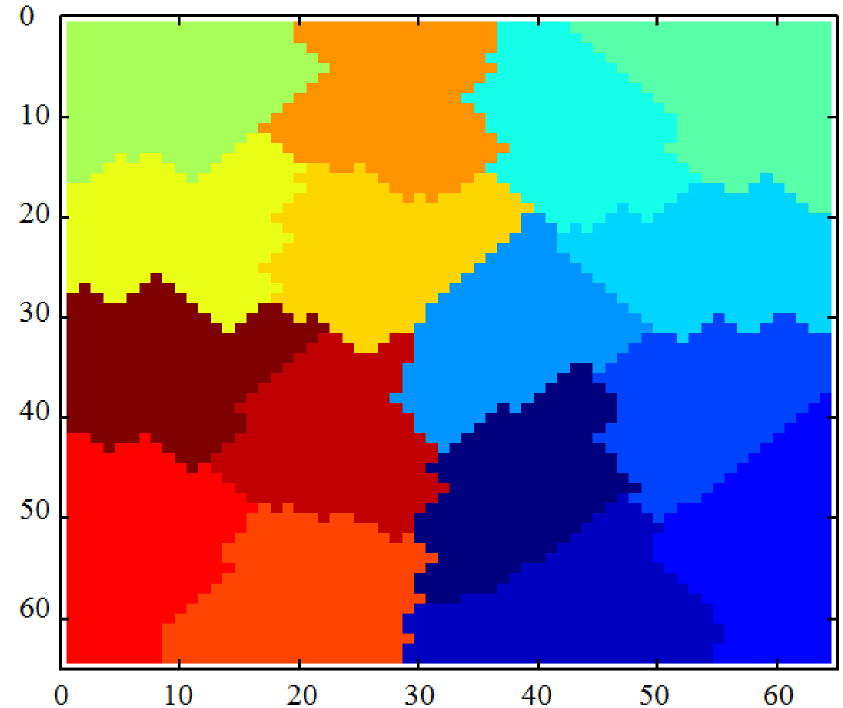
For $N \times N$ mesh on $P \times P$ processor grid:
Usual Cartesian partitioning costs $\approx 4NP$ words moved
MeshPart costs $\approx 3NP$ words moved, 25% savings

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

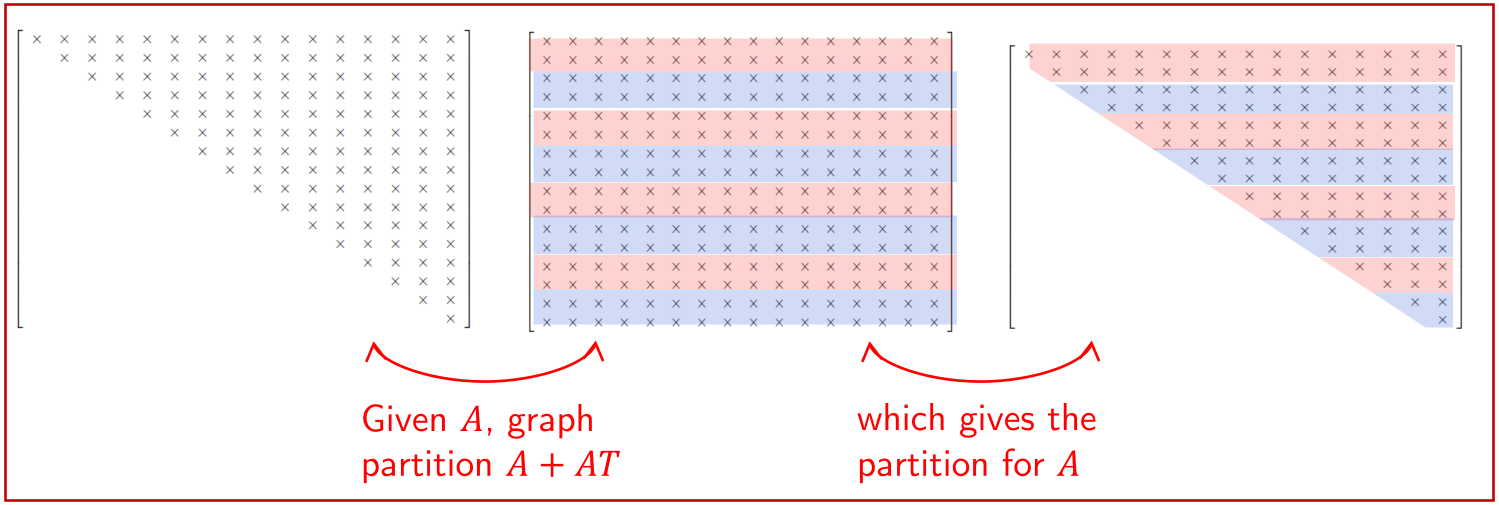
~8% reduction in total communication volume
using hypergraph partitioning (PaToH)
versus graph partitioning (METIS)

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?

Illustrative Bad Example: triangular matrix

Whereas the hypergraph model can capture nonsymmetry, the graph partitioning model deals with nonsymmetry by partitioning the graph of $A + A^T$ (which in this case is a dense matrix).

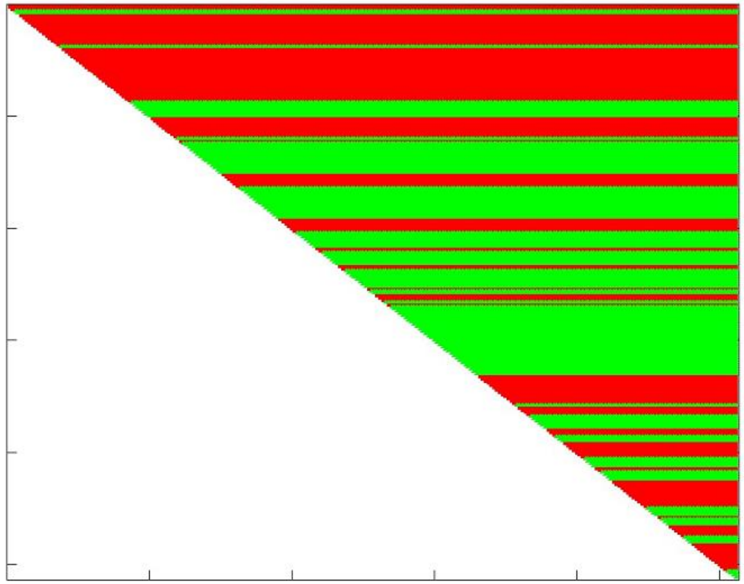


This results in a suboptimal partition in terms of both communication and load balancing. In this case,

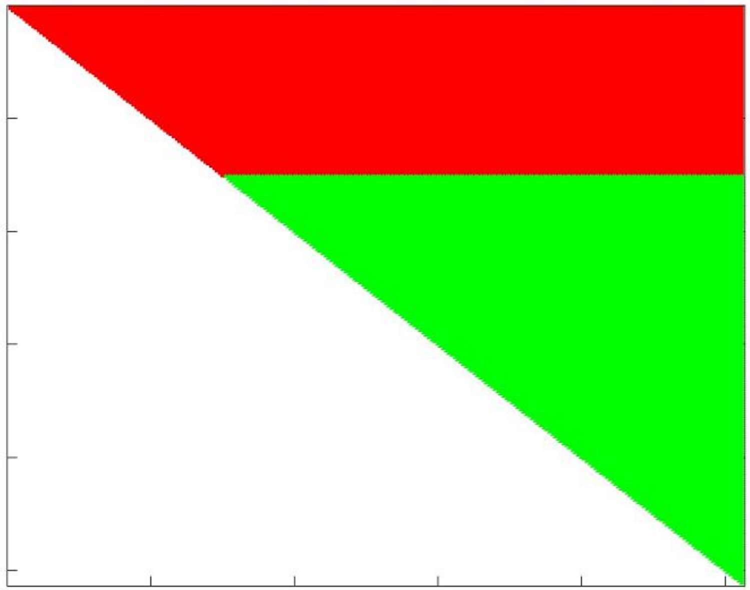
Total Communication Volume = 60 (optimal is ~12 in this case, subject to load balancing)
Proc1: 76 nonzeros, Proc 2: 60 nonzeros (~26% imbalance ratio)

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

Some References

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

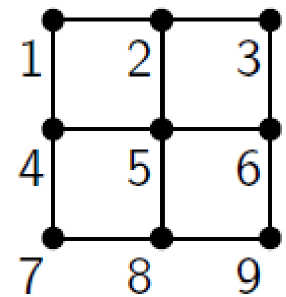
Outline

- 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
- Graph algorithms in sparse direct methods

Symmetric sparse matrices and graphs

- The structure of a square symmetric matrix A with nonzero diagonal can be represented by an undirected graph $G(A) = (V, E)$ with
  - n vertices, one for each row/column of A
  - an edge (i, j) for each nonzero a_{ij}, $i > j$

$$
A = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \end{array}
\begin{array}{c}
\begin{array}{ccccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \end{array} \\
\left(\begin{array}{ccccccccc}
x & x &   & x &   &   &   &   &   \\
x & x & x &   & x &   &   &   &   \\
  & x & x &   &   & x &   &   &   \\
x &   &   & x & x &   & x &   &   \\
  & x &   & x & x & x &   & x &   \\
  &   & x &   & x & x &   &   & x \\
  &   &   & x &   &   & x & x &   \\
  &   &   &   & x &   & x & x & x \\
  &   &   &   &   & x &   & x & x
\end{array}\right)
\end{array}
$$



$G(A)$

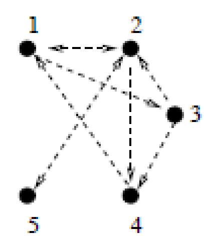
Notation: upper case (A) - matrices; lower case (a_{ij}) - elements

Nonsymmetric sparse matrices and graphs

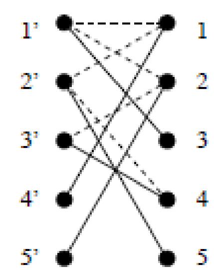
- The structure of a nonsymmetric matrix A of size $n \times n$ can be represented by
  - a directed graph $G(A) = (V, E)$ with
    - n vertices, one for each column of A
    - an edge from i to j for each nonzero a_{ij}
  - a bipartite graph $H(A) = (V, E)$ with
    - $2n$ vertices, for n rows and n columns of A
    - an edge (i', j) for each nonzero a_{ij}
  - a hypergraph

	1	2	3	4	5
1'	x	x	x		
2'	x			x	x
3'		x		x	
4'	x				
5'		x			

A



G(A)



H(A)

Sparse Linear Solvers

Direct methods of factorization

- For solving $Ax = b$, least squares problems
  - Cholesky, LU, QR, LDL^T factorizations
- Limited by fill-in/memory consumption and scalability

Iterative solvers

- For solving $Ax = b$, least squares, $Ax = \lambda x$, SVD
- When only multiplying A by a vector is possible
- Limited by accuracy/convergence

Hybrid methods

As domain decomposition methods

Examples of Sparse Direct Solvers

A non-complete list of solvers and their characteristics:

- PSPASES: for SPD matrices, distributed memory.
  `http://www-users.cs.umn.edu/~mjoshi/pspases/`
- UMFPACK / SuiteSparse (Matlab, Google Ceres) - symmetric/unsymmetric, LU, QR, multicores/GPUs.
  `http://faculty.cse.tamu.edu/davis/suitesparse.html`
- SuperLU: unsymmetric matrices, shared/distributed memory.
  `http://crd-legacy.lbl.gov/~xiaoye/SuperLU/`
- MUMPS: symmetric/unsymmetric, distributed memory.
  `http://mumps.enseeiht.fr/`
- Pardiso (Intel MKL): symmetric/unsymmetric, shared/distributed memory.
  `http://www.pardiso-project.org/`

Review: LU Factorization

LU factorization

Compute the factorization PA = LU

Example

Given the matrix

$$A = \begin{pmatrix} 3 & 0 & 3 \\ 6 & 7 & 0 \\ 9 & 12 & 3 \end{pmatrix}$$

The first step of the LU factorization is performed as

$$M_1 = \begin{pmatrix} 1 & & \\ -2 & 1 & \\ -3 & & 1 \end{pmatrix}, \quad M_1 A = \begin{pmatrix} 3 & 0 & 3 \\ 0 & 7 & -6 \\ 0 & 12 & -6 \end{pmatrix}$$

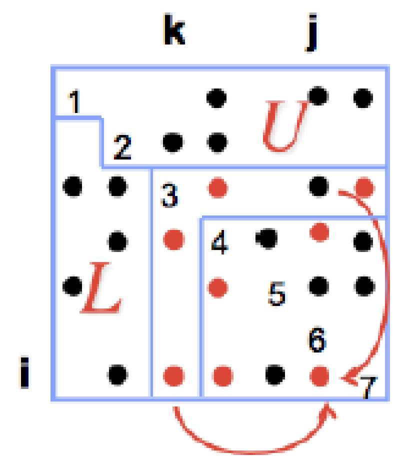
Fill-in elements

Are elements which are zero in A, but become nonzero in L or U (as -6 above).

Sparse LU Factorization

Right looking factorization of A by rows

for $k = 1 : n-1$ do
  Permute row i and row k, where a_{ik} is element of maximum magnitude in $A(k:n,k)$
  for $i = k+1 : n$ st $a_{ik} \neq 0$ do
    /* store in place l_{ik} */
    $a_{ik} = a_{ik} / a_{kk}$
    /* add a multiple of row k to row i */
    for $j = k+1 : n$ st $a_{kj} \neq 0$ do
      $a_{ij} = a_{ij} - a_{ik} * a_{kj}$
    end for
  end for
end for



Observations

- The order of the indices i, j, k can be changed, leading to different algorithms:
  - computing the factorization by rows, by columns, or by sub-matrices,
  - using a left looking, right looking, or multifrontal approach.

Simple case

A is symmetric and positive definite (SPD) if

- $A = A^T$,
- all its eigenvalues are positive,
- or equivalently, A has a Cholesky factorization, $A = LL^T$.

Some properties of an SPD matrix A

- There is no need to pivot for accuracy (just performance) during the Cholesky factorization.
- For any permutation matrix P, PAP^T is also SPD.

Sparse Cholesky

- The algebra can be written as:

$$A = \begin{pmatrix} a_{11} & A_{21}^T \\ A_{21} & A_{22} \end{pmatrix} = \begin{pmatrix} \sqrt{a_{11}} & \\ A_{21}./\sqrt{a_{11}} & L_{22} \end{pmatrix} \cdot \begin{pmatrix} \sqrt{a_{11}} & A_{21}^T./\sqrt{a_{11}} \\ & L_{22}^T \end{pmatrix}$$

- Compute and store only the lower triangular part since $U = L^T$.

Algorithm

for $k = 1 : n-1$ **do**
    $a_{kk} = \sqrt{a_{kk}}$
    /* *factor*(k) */
    for $i = k+1 : n$ st $a_{ik} \neq 0$ **do**
        $a_{ik} = a_{ik}/a_{kk}$
    end for
    for $i = k+1 : n$ st $a_{ik} \neq 0$ **do**
        update(k, i)
        for $j = i : n$ st $a_{kj} \neq 0$ **do**
            $a_{ij} = a_{ij} - a_{ik} a_{jk}$
        end for
    end for
end for

Filled graph $G^+(A)$

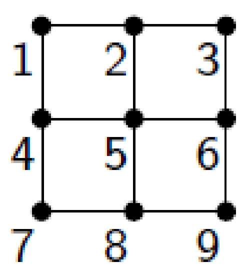
- Given $G(A) = (V, E)$, $G^+(A) = (V, E^+)$ is defined as:
  there is an edge $(i, j) \in G^+(A)$ iff there is a path from i to j in $G(A)$
  going through lower numbered vertices.
- Definition holds also for directed graphs (LU factorization).

- $G(L + L^T) = G^+(A)$, ignoring cancellations.

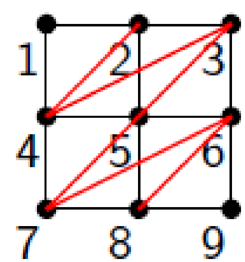
Filled graph $G^+(A)$

$$A = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \end{array} \begin{array}{c} \begin{array}{ccccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \end{array} \\ \left(\begin{array}{ccccccccc} x & x & & x & & & & & \\ x & x & x & & x & & & & \\ & x & x & & & x & & & \\ x & & & x & x & & x & & \\ & x & & x & x & x & & x & \\ & & x & & x & x & & & x \\ & & & x & & & x & x & \\ & & & & x & & x & x & x \\ & & & & & x & & x & x \end{array} \right) \end{array}$$

$$L + L^T = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \end{array} \begin{array}{c} \begin{array}{ccccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \end{array} \\ \left(\begin{array}{ccccccccc} x & x & & x & & & & & \\ x & x & x & \color{red}{x} & x & & & & \\ & x & x & \color{red}{x} & \color{red}{x} & x & & & \\ x & \color{red}{x} & \color{red}{x} & x & x & & x & & \\ & x & \color{red}{x} & x & x & x & \color{red}{x} & x & \\ & & x & & x & x & \color{red}{x} & \color{red}{x} & x \\ & & & x & \color{red}{x} & \color{red}{x} & x & x & \\ & & & & x & \color{red}{x} & x & x & x \\ & & & & & x & & x & x \end{array} \right) \end{array}$$



$G(A)$



$G^+(A)$

Filled graph $G^+(A)$

- $G^+(A)$ is chordal (every cycle of length at least four has a chord, an edge connecting two non-neighboring nodes).
- Conversely, if $G(A)$ is chordal, then there is a perfect elimination order, that is a permutation P such that $G(PAP^T) = G^+(PAP^T)$.

- References: [Parter, 1961, Rose, 1970, Rose and Tarjan, 1978]

Steps of Sparse Cholesky

1. Order rows and columns of A to reduce fill-in

2. Symbolic factorization: based on eliminaton trees

   - Compute the elimination tree (in nearly linear time in $nnz(A)$)
   - Allocate data structure for L
   - Compute the nonzero structure of the factor L, in $O(nnz(L)$

3. Numeric factorization

   - Exploit memory hierarchy
   - Exploit parallelism due to sparsity

4. Triangular solve

Ordering rows and columns of A

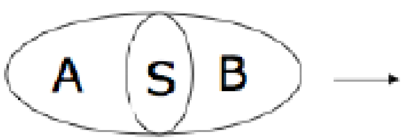
Strategies applied to the graph of A for Cholesky,
to the graph of $A^T A$ for LU with partial pivoting.

Local strategy: minimum degree [Tinney/Walker '67]

- Minimize locally the fill-in.
- Choose at each step (for 1 to n) the node of minimum degree.

Global strategy: graph partitioning approach

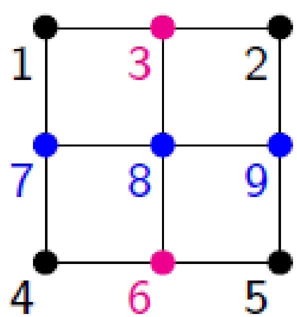
- Nested dissection [George, 1973]
  - First level: find the smallest possible separator S, order last
  - Recurse on A and B
- Multilevel schemes [Barnard/Simon '93, Hendrickson/Leland '95, Karypis/Kumar '95].



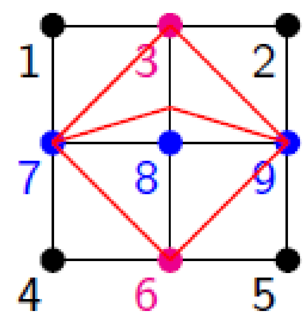
$$\begin{matrix} A & 0 & x \\ 0 & B & x \\ x & x & S \end{matrix}$$

Nested Dissection Example

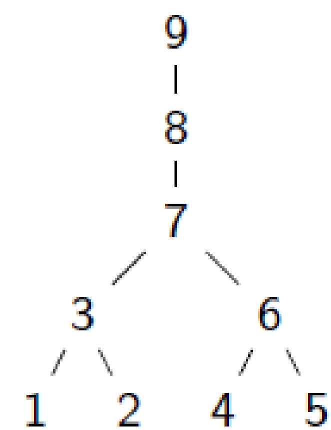
$$
A = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \end{array}
\begin{array}{c}
\begin{array}{ccccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \end{array} \\
\left(\begin{array}{cc|c|cc|c|ccc}
x & & x & & & & x & & \\
 & x & x & & & & & & x \\
\hline
x & x & x & & & & & x & \\
\hline
 & & & x & & x & x & & \\
 & & & & x & x & & & x \\
\hline
 & & & x & x & x & & x & \\
\hline
x & & & x & & & x & x & \\
 & & x & & & x & x & x & x \\
 & x & & & x & & & x & x
\end{array}\right)
\end{array},
\qquad
L + L^T = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \end{array}
\begin{array}{c}
\begin{array}{ccccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \end{array} \\
\left(\begin{array}{cc|c|cc|c|ccc}
x & & x & & & & x & & \\
 & x & x & & & & & & x \\
\hline
x & x & x & & & & x & x & x \\
\hline
 & & & x & & x & x & & \\
 & & & & x & x & & & x \\
\hline
 & & & x & x & x & x & x & x \\
\hline
x & & x & x & & x & x & x & x \\
 & & x & & & x & x & x & x \\
 & x & x & & x & x & x & x & x
\end{array}\right)
\end{array}
$$



$G(A)$



$G^+(A)$



$T(A)$

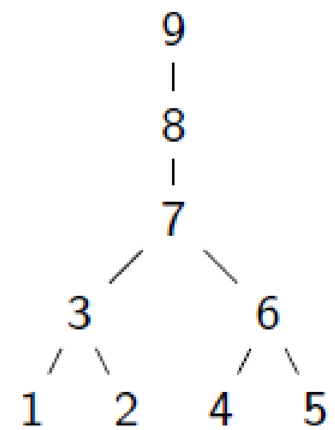
Elimination Tree

Definition ([Schreiber, 1982] and also [Duff, 1982])

Given $A = LL^T$, the etree $T(A)$ has the same node set as $G(A)$, and k is the parent of j iff

$$k = \min\{i > j : l_{ij} \neq 0\}$$

$$L + L^T = \begin{array}{c|c} & \begin{array}{ccccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \end{array} \\ \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \end{array} & \left(\begin{array}{cc|c|cc|c|ccc} x & & x & & & & x & & \\ & x & x & & & & & & x \\ \hline x & x & x & & & & x & x & x \\ \hline & & & x & & x & x & & \\ & & & & x & x & & & x \\ \hline & & & x & x & x & x & x & x \\ \hline x & & x & x & & x & x & x & x \\ & & x & & & x & x & x & x \\ & x & x & & x & x & x & x & x \end{array}\right) \end{array}$$



$T(A)$

Elimination Tree

Definition ([Schreiber, 1982] and also [Duff, 1982])

Given $A = LL^T$, the etree $T(A)$ has the same node set as $G(A)$, and k is the parent of j iff

$$k = \min\{i > j : l_{ij} \neq 0\}$$

Properties (ignoring cancellations), for more details see e.g. [Liu, 1990]

- $T(A)$ is a spanning tree of the filled graph $G^+(A)$.
- $T(A)$ is the result of the transitive reduction of the directed graph $G(L^T)$.
- $T(A)$ of a connected graph $G(A)$ is a depth first search tree of $G^+(A)$ (with specific tie-breaking rules).

Elimination Tree

Complexity

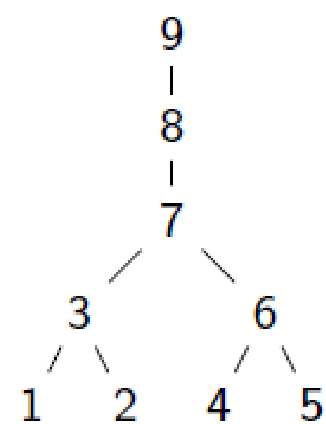
- Can be computed in $O(nnz(A)\alpha(nnz(A), n))$, where $\alpha()$ is the inverse of Ackerman's function.

- Can be used to
  - compute # nonzeros of each column/row of L (same complexity),
  - identify columns with similar structure (supernodes), (same complexity)
  - compute nonzero structure of L, in $O(nnz(L))$

Column dependencies and elimination tree

- If $l_{jk} \neq 0$, then
  - $Factor(k)$ needs to be computed before $Factor(j)$.
  - k is an ancestor of j in $T(A)$.
- Columns belonging to disjoint subtrees can be factored independently.
- Topological orderings of $T(A)$ (that number children before their parent)
  - preserve the amount of fill, the flops of the factorization, the structure of $T(A)$
  - postordering most used in practice

$$
\begin{array}{cc}
 & \begin{array}{ccccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \end{array} \\
L + L^T = \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \end{array} & \left(\begin{array}{cc|c|cc|c|ccc}
x & & x & & & & x & & \\
 & x & x & & & & & & x \\ \hline
x & x & x & & & & {\color{red}x} & x & {\color{red}x} \\ \hline
 & & & x & & x & x & & \\
 & & & & x & x & & & x \\ \hline
 & & & x & x & x & {\color{red}x} & x & {\color{red}x} \\ \hline
x & & {\color{red}x} & x & & {\color{red}x} & x & x & {\color{red}x} \\
 & & x & & & x & x & x & x \\
 & x & {\color{red}x} & & x & {\color{red}x} & {\color{red}x} & x & x
\end{array}\right)
\end{array}
$$

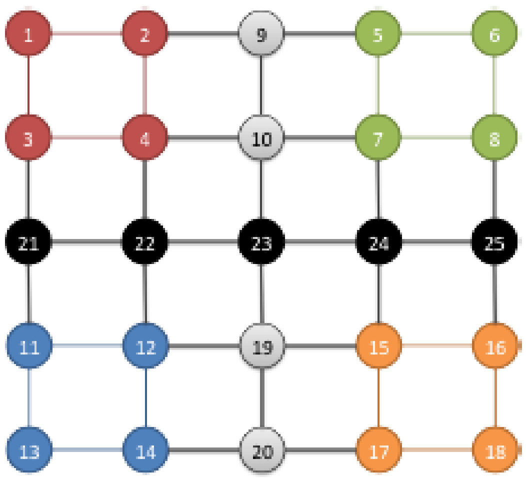


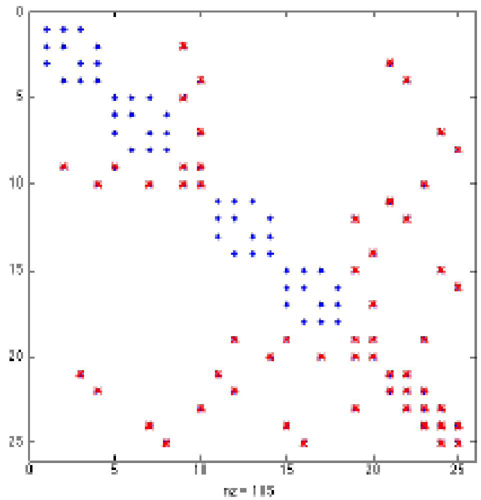
$T(A)$

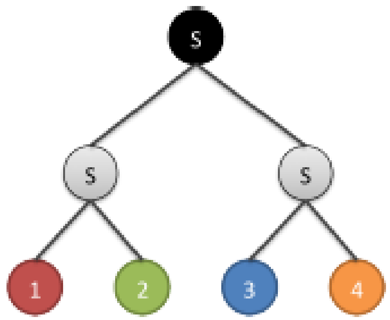
Nested Dissection and Separator Tree

Separator tree:

- Combines together nodes belonging to a same separator, or to a same disjoint graph







- Available packages:
  - Metis, parmetis
  - Scotch, PT-Scotch

Lower Bounds on Communication for Sparse Direct Solvers

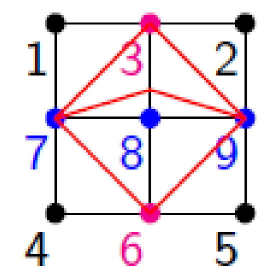
- More difficult than the dense case
  - For example computing the product of two (block) diagonal matrices involves no communication in parallel
- Lower bound on communication from dense linear algebra is loose
- Very few existing results:
  - Lower bounds for parallel multiplication of sparse random matrices [Ballard et al., 2013]
  - Lower bounds for Cholesky factorization of model problems [Grigori et al., 2010]

Lower bounds for Cholesky

- Consider A of size $k^s \times k^s$ results from a finite difference operator on a regular grid of dimension $s \geq 2$ with k^s nodes.

- Its Cholesky L factor contains a dense lower triangular matrix of size $k^{s-1} \times k^{s-1}$.

$$
L + L^T = \begin{array}{c|c}
 & \begin{array}{ccccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \end{array} \\
\begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \end{array} &
\begin{pmatrix}
x & & x & & & & x & & \\
 & x & x & & & & & & x \\
x & x & x & & & & x & x & x \\
 & & & x & & x & x & & \\
 & & & & x & x & & & x \\
 & & & x & x & x & x & x & x \\
x & & x & x & & x & x & x & x \\
 & & x & & & x & x & x & x \\
 & x & x & & x & x & x & x & x
\end{pmatrix}
\end{array}
$$



$G^+(A)$

- Computing the Cholesky factorization of the $k^{s-1} \times k^{s-1}$ matrix dominates the computation and the communication.

Lower bounds for Cholesky

- This result applies more generally to matrix A whose graph $G = (V, E)$, $|V| = n$ has the following property for some l:
  - if every set of vertices $W \subset V$ with $n/3 \leq |W| \leq 2n/3$ is adjacent to at least l vertices in $V - W$,
  - then the Cholesky factor of A contains a dense $l \times l$ submatrix.

Lower Bounds for Cholesky

For the Cholesky factorization of a $k^s \times k^s$ matrix resulting from a finite difference operator on a regular grid of dimension $s \geq 2$ with k^s nodes:

$$\#words \geq \Omega\left(\frac{W}{\sqrt{M}}\right), \qquad \#messages \geq \Omega\left(\frac{W}{M^{3/2}}\right)$$

- Sequential algorithm
  - $W = k^{3(s-1)}/3$ and M is the fast memory size
- Work balanced parallel algorithm executed on P processors
  - $W = \frac{k^{3(s-1)}}{3P}$ and $M \approx nnz(L)/P$