CS 240A: Graph and hypergraph partitioning

Thanks to Aydin Buluc, Umit Catalyurek,
Alan Edelman, and Kathy Yelick
for some of these slides.

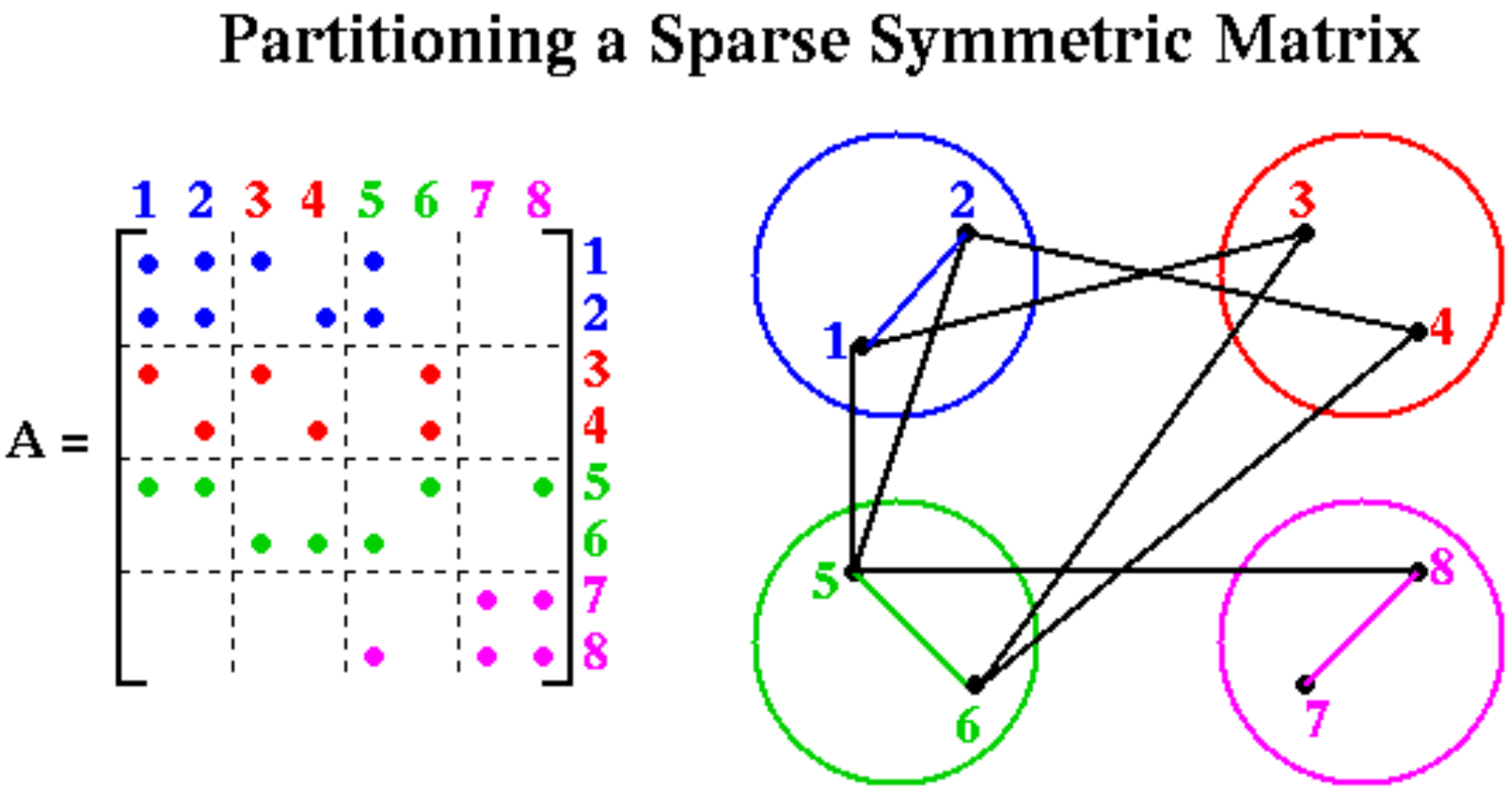
CS 240A: Graph and hypergraph partitioning

- Motivation and definitions
  - Motivation from parallel computing
  - Theory of graph separators
- Heuristics for graph partitioning
  - Iterative swapping
  - Spectral
  - Geometric
  - Multilevel
- Beyond graphs
  - Shortcomings of the graph partitioning model
  - Hypergraph models of communication in MatVec
- Recent ideas: parallel partitioning, scale-free graphs, etc.

CS 240A: Graph and hypergraph partitioning

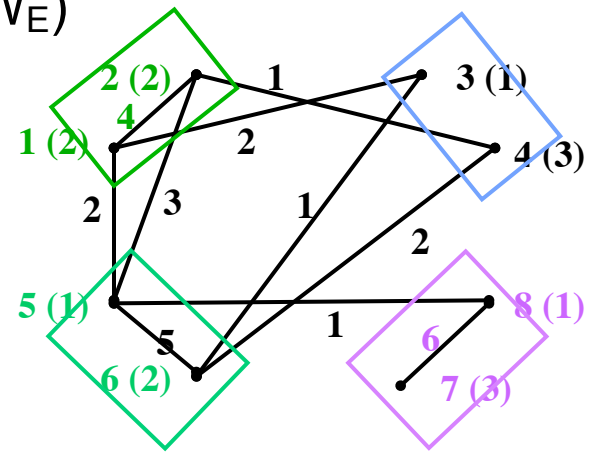
- Motivation and definitions
  - Motivation from parallel computing
  - Theory of graph separators
- Heuristics for graph partitioning
  - Iterative swapping
  - Spectral
  - Geometric
  - Multilevel
- Beyond graphs
  - Shortcomings of the graph partitioning model
  - Hypergraph models of communication in MatVec
- Recent ideas: parallel partitioning, scale-free graphs, etc.

Sparse Matrix Vector Multiplication



Definition of Graph Partitioning

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



- Often nodes are tasks, edges are communication, weights are costs
- Choose a partition N = N_1 U N_2 U … U N_P such that
  - Total weight of nodes in each part is "about the same"
  - Total weight of edges connecting nodes in different parts is small
- Balance the work load, while minimizing communication

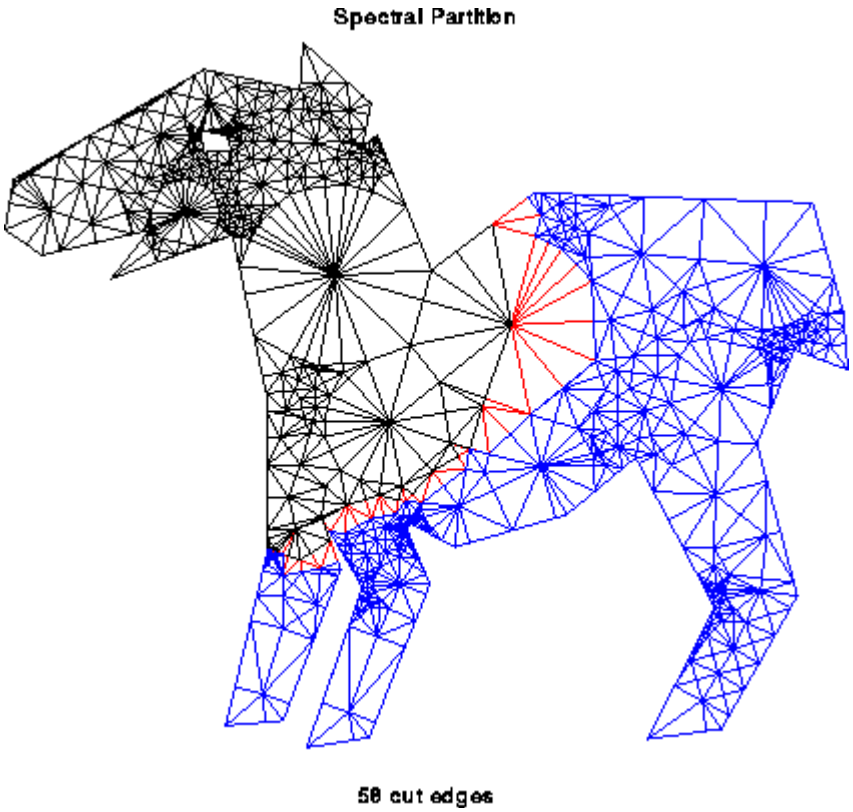
- Special case of N = N_1 U N_2: Graph Bisection

Applications

- Telephone network design
  - Original application, algorithm due to Kernighan
- Load Balancing while Minimizing Communication
- Sparse Matrix times Vector Multiplication
  - 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

Partitioning by Repeated Bisection

- To partition into 2^k parts, bisect graph recursively k times



Separators in theory

- If G is a planar graph with n vertices, there exists a set of at most sqrt(6n) vertices whose removal leaves no connected component with more than 2n/3 vertices. ("Planar graphs have sqrt(n)-separators.")
- "Well-shaped" finite element meshes in 3 dimensions have $n^{2/3}$ - separators.
- Also some others – trees, graphs of bounded genus, chordal graphs, bounded-excluded-minor graphs, …
- Mostly these theorems come with efficient algorithms, but they aren't used much.
- "Random graphs" don't have good separators.
  - e.g. Erdos-Renyi random graphs have only n - separators.

CS 240A: Graph and hypergraph partitioning

- Motivation and definitions
  - Motivation from parallel computing
  - Theory of graph separators
- Heuristics for graph partitioning
  - Iterative swapping
  - Spectral
  - Geometric
  - Multilevel
- Beyond graphs
  - Shortcomings of the graph partitioning model
  - Hypergraph models of communication in MatVec
- Recent ideas: parallel partitioning, scale-free graphs, etc.

Separators in practice

- Graph partitioning heuristics have been an active research area for many years, often motivated by partitioning for parallel computation.

- Some techniques:
  - Iterative-swapping (Kernighan-Lin, Fiduccia-Matheysses)
  - Spectral partitioning (uses eigenvectors of Laplacian matrix of graph)
  - Geometric partitioning (for meshes with specified vertex coordinates)
  - Breadth-first search (fast but dated)

- Many popular modern codes (e.g. Metis, Chaco, Zoltan) use multilevel iterative swapping.

- New ideas in: *parallelism*; partitioning *scale-free graphs.*

Iterative swapping:
Kernighan/Lin, Fiduccia/Mattheyses

- Take a initial partition and iteratively improve it
  - Kernighan/Lin (1970), cost = $O(|N|^3)$ but simple
  - Fiduccia/Mattheyses (1982), cost = $O(|E|)$ but more complicated

- Start with a weighted graph and a partition A U B, where |A| = |B|
  - T = cost(A,B) = Σ {weight(e): e connects nodes in A and B}
  - Find subsets X of A and Y of B with |X| = |Y|
  - Swapping X and Y should decrease cost:
    - newA = A - X U Y and newB = B - Y U X
    - newT = cost(newA , newB) < cost(A,B)

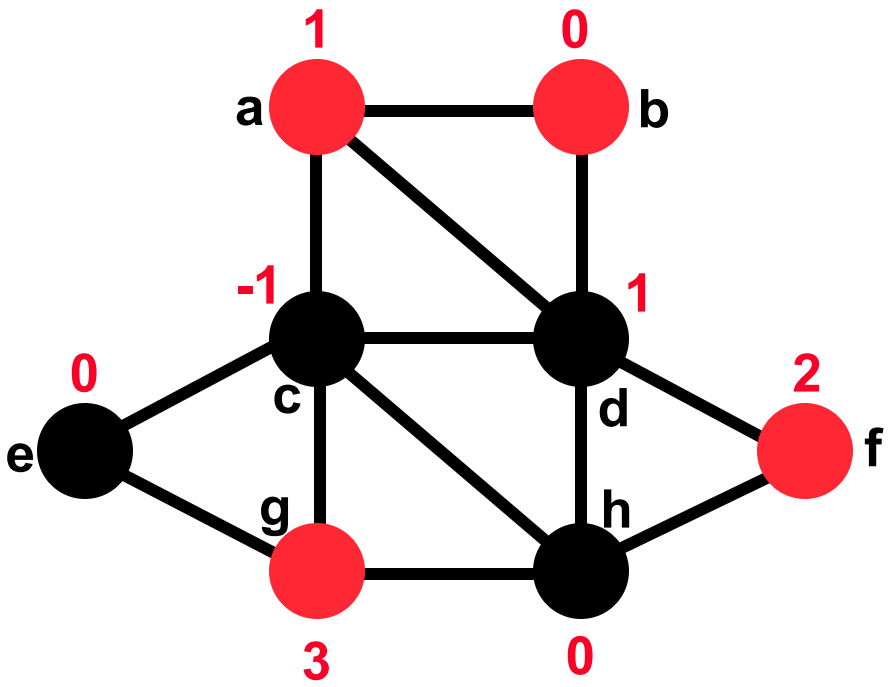
- Compute newT efficiently for many possible X and Y, (not time to do all possible), then choose smallest

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.



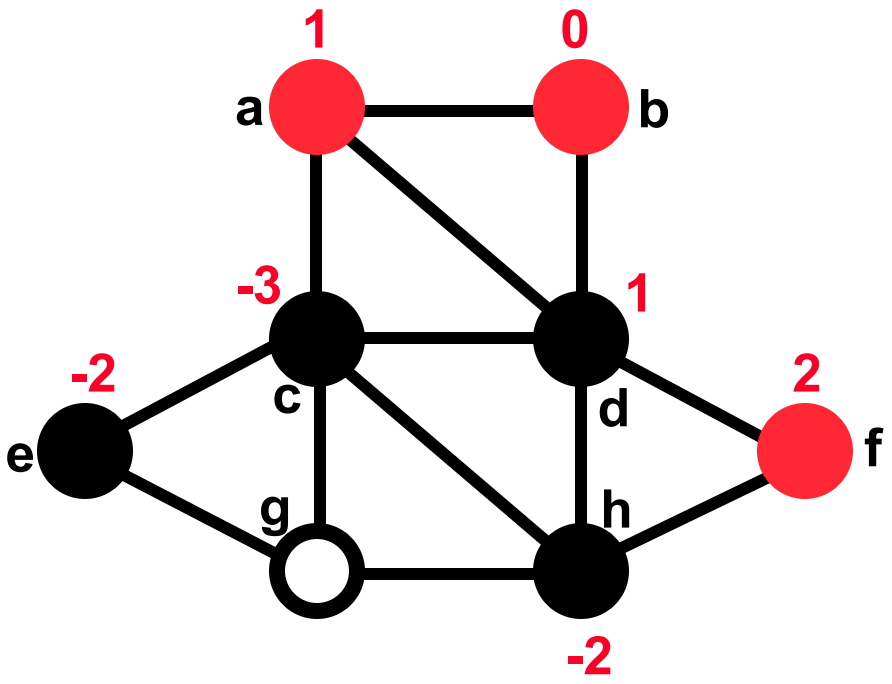
Nodes tentatively moved (and cut size after each pair):

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.



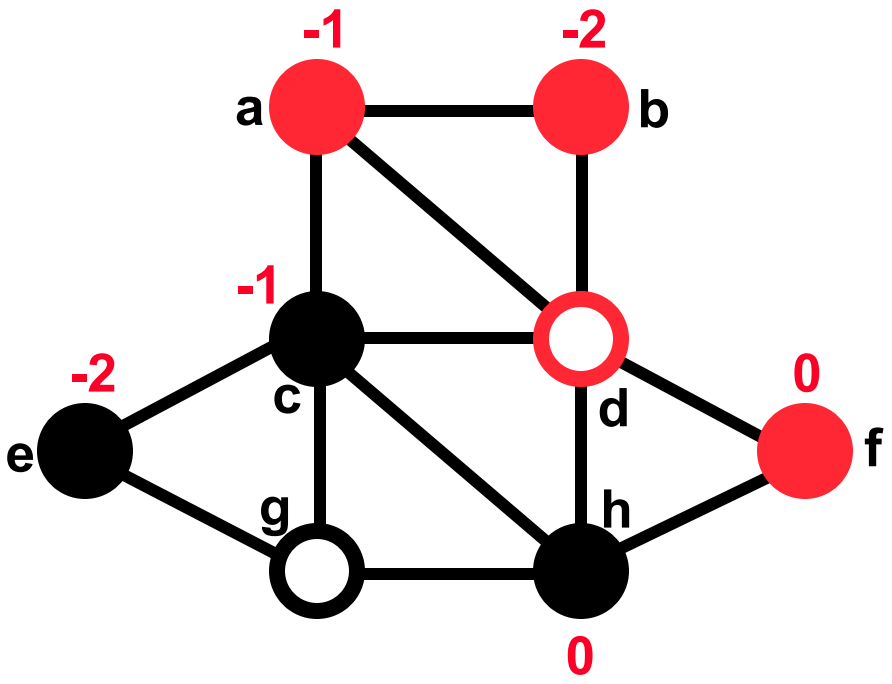
Nodes tentatively moved (and cut size after each pair):

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.

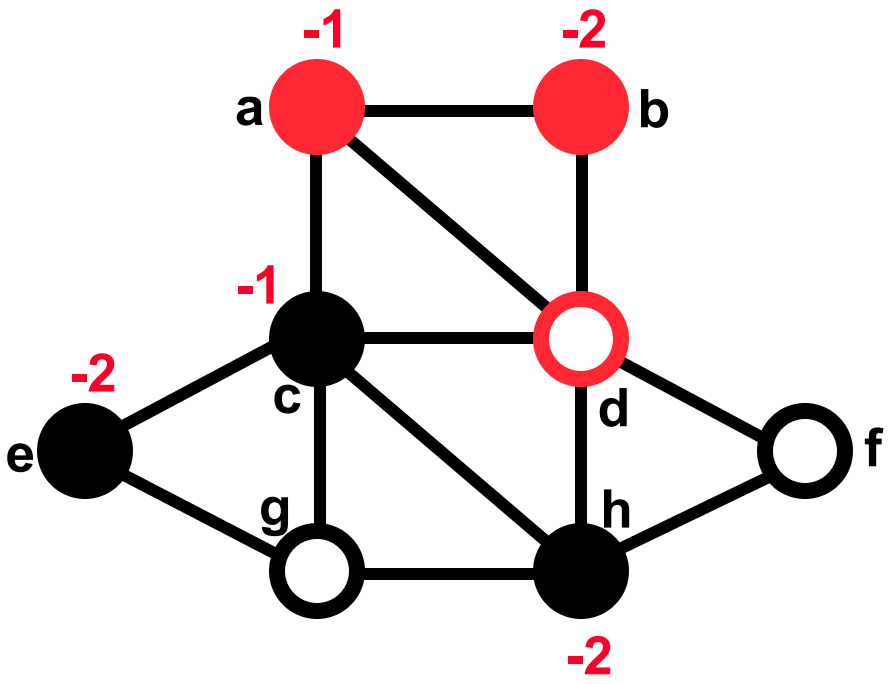


Nodes tentatively moved (and cut size after each pair):

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.



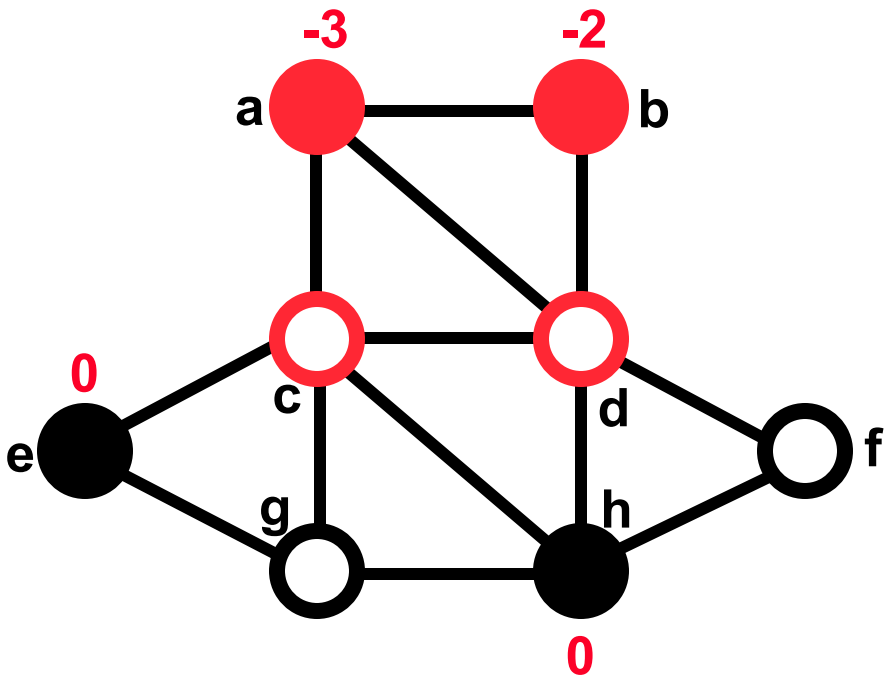
Nodes tentatively moved (and cut size after each pair):

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.

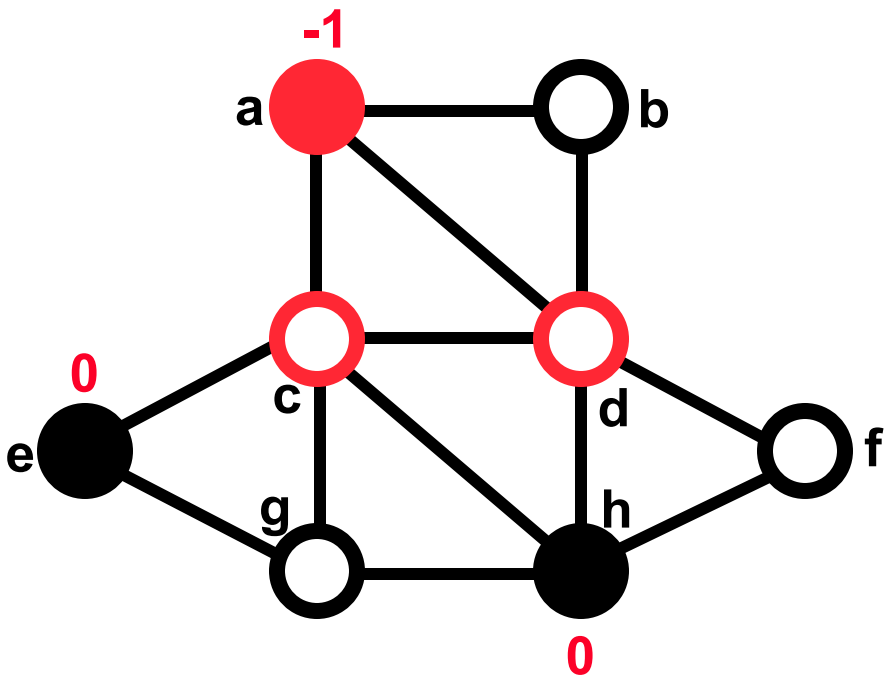


Nodes tentatively moved (and cut size after each pair):

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.



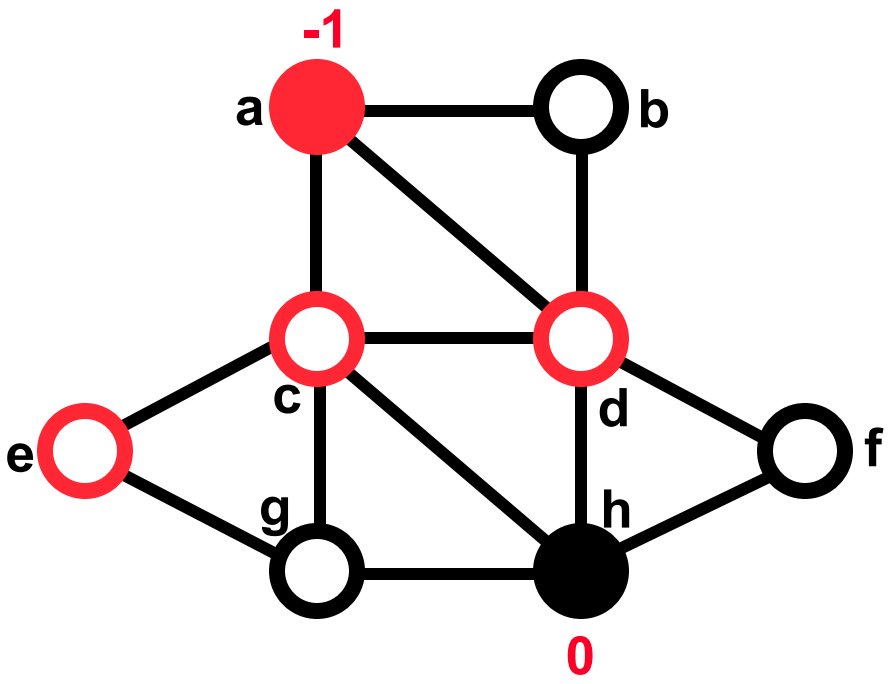
Nodes tentatively moved (and cut size after each pair):

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.

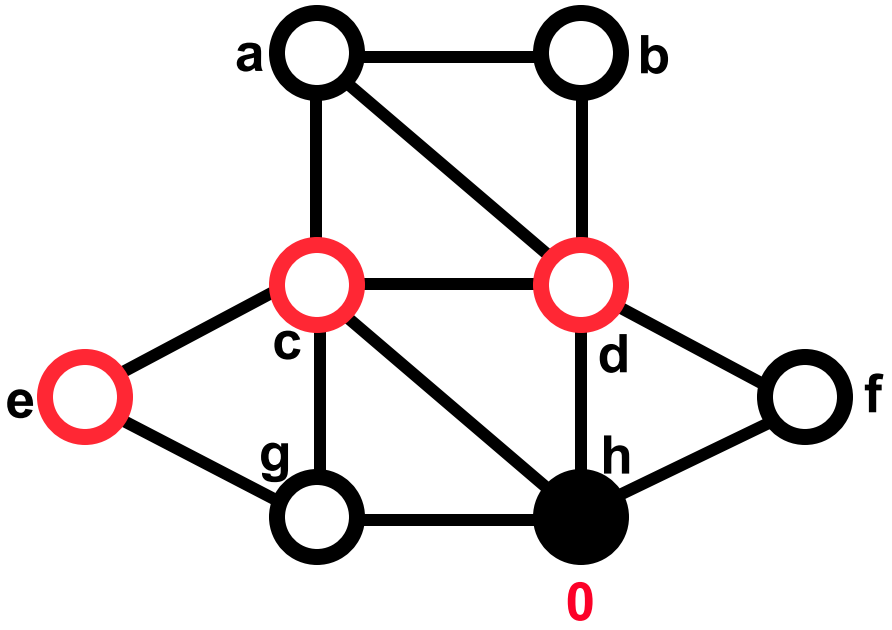


Nodes tentatively moved (and cut size after each pair):

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.



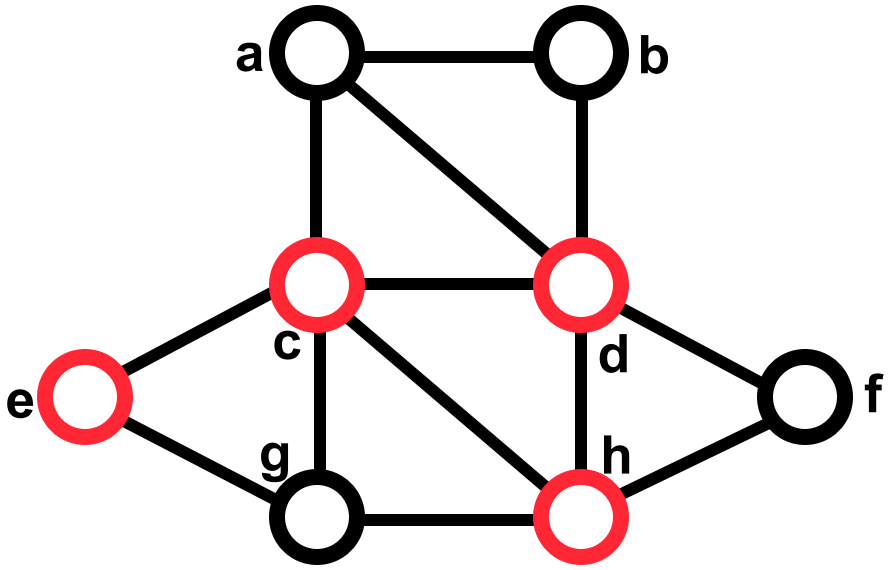
Nodes tentatively moved (and cut size after each pair):

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.



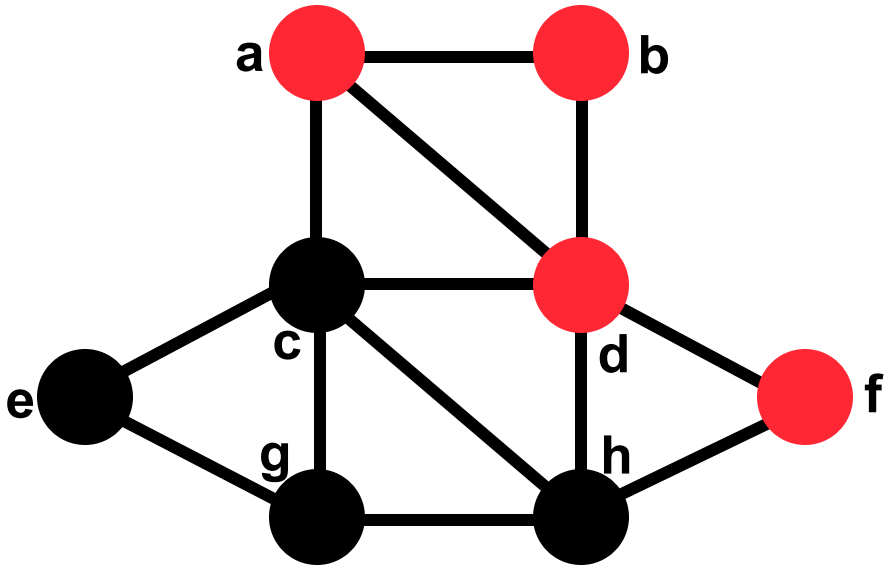
Nodes tentatively moved (and cut size after each pair):

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):

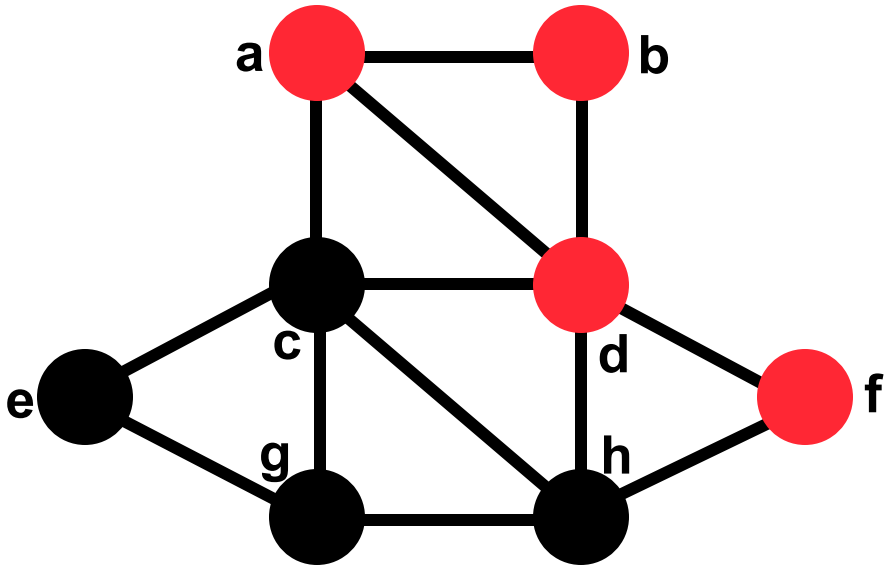
none (8); **g, d (4);** f, c (5); b, e (7); a, h (8)

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.



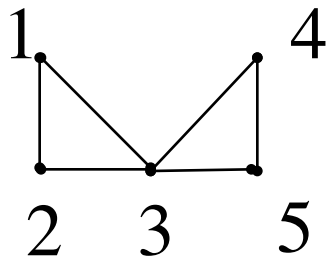
Spectral Bisection

- Based on theory of Fiedler (1970s), rediscovered several times in different communities
  - Motivation I: analogy to a vibrating string
  - Motivation II: continuous relaxation of discrete optimization
- Implementation: eigenvectors via Lanczos algorithm
  - To optimize sparse-matrix-vector multiply, we graph partition
  - To graph partition, we find an eigenvector of a matrix
  - To find an eigenvector, we do sparse-matrix-vector multiply
  - No free lunch ...

Laplacian Matrix

- *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

G =



$$\mathbf{L(G) =} \begin{pmatrix} 2 & -1 & -1 & 0 & 0 \\ -1 & 2 & -1 & 0 & 0 \\ -1 & -1 & 4 & -1 & -1 \\ 0 & 0 & -1 & 2 & -1 \\ 0 & 0 & -1 & -1 & 2 \end{pmatrix}$$

Properties of Laplacian Matrix

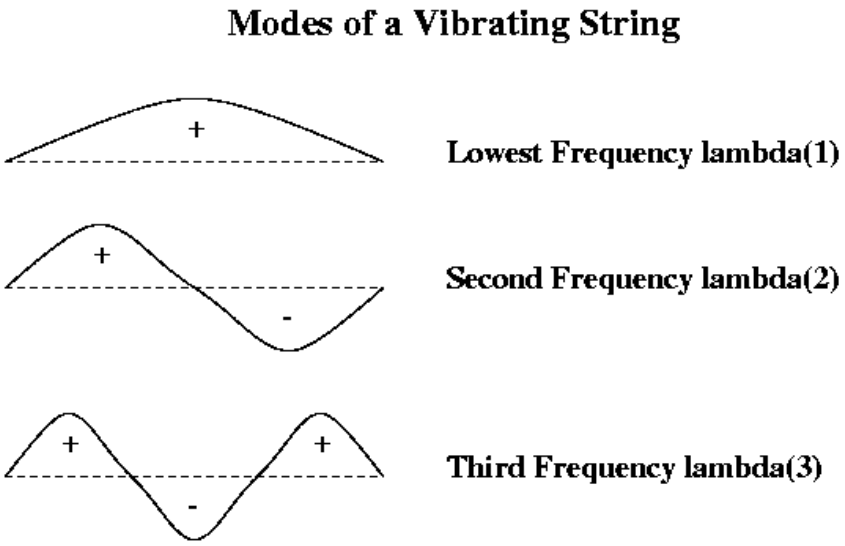
- *Theorem:* L(G) has the following properties
  - L(G) is symmetric.
    - This implies the eigenvalues of L(G) are real, and its eigenvectors are real and orthogonal.
  - Rows of L sum to zero:
    - Let e = $[1,\ldots,1]^T$, i.e. the column vector of all ones. Then L(G)*e=0.
  - The eigenvalues of L(G) are nonnegative:
    - $0 = \lambda_1 <= \lambda_2 <= \ldots <= \lambda_n$
  - The number of connected components of G is equal to the number of λ_i that are 0.

Spectral Bisection Algorithm

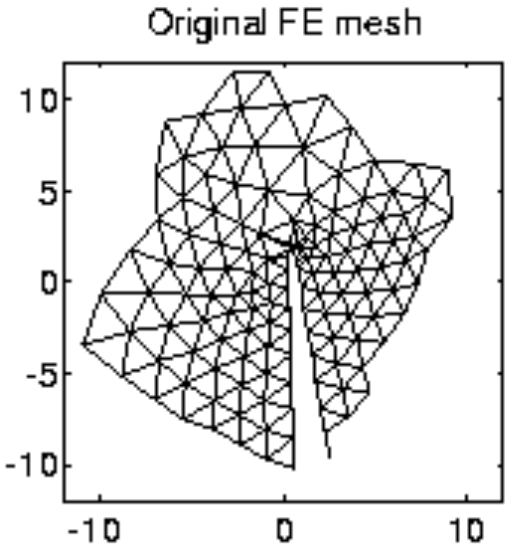
- Spectral Bisection Algorithm:
  - Compute eigenvector v_2 corresponding to $\lambda_2(L(G))$
  - Partition nodes around the median of $v_2(n)$
- Why in the world should this work?
- Intuition: vibrating string or membrane
- Heuristic: continuous relaxation of discrete optimization

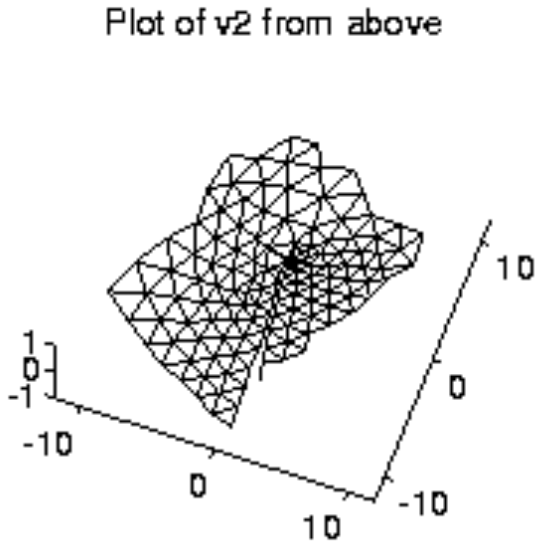
Motivation for Spectral Bisection

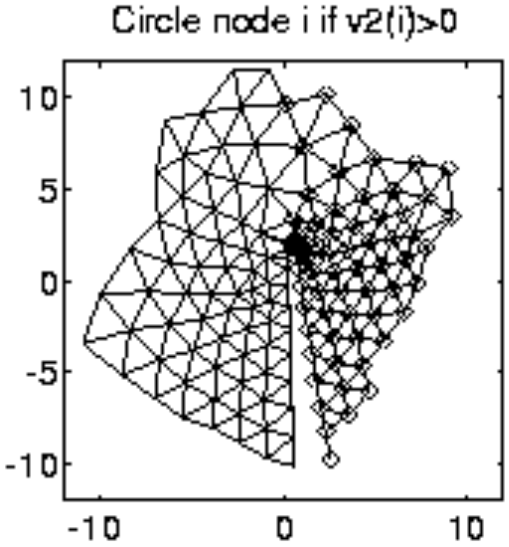
- Vibrating string
- Think of G = 1D mesh as masses (nodes) connected by springs (edges), i.e. a string that can vibrate
- Vibrating string has modes of vibration, or harmonics
- Label nodes by whether mode - or + to partition into N- and N+
- Same idea for other graphs (eg planar graph ~ trampoline)

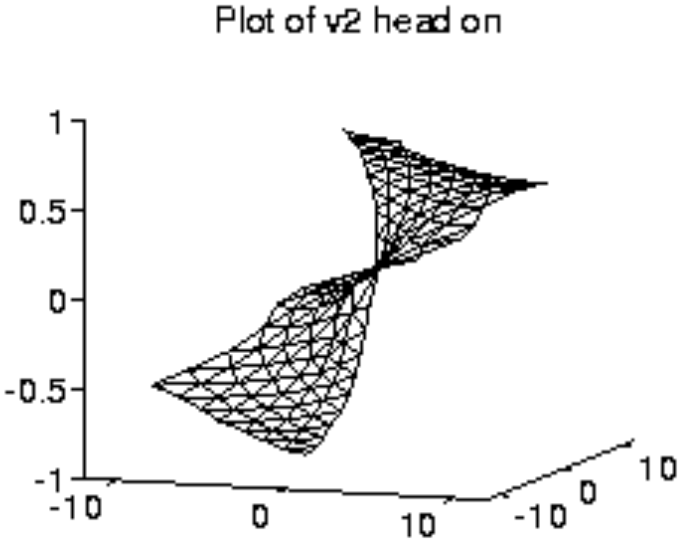


2nd eigenvector of L(planar mesh)



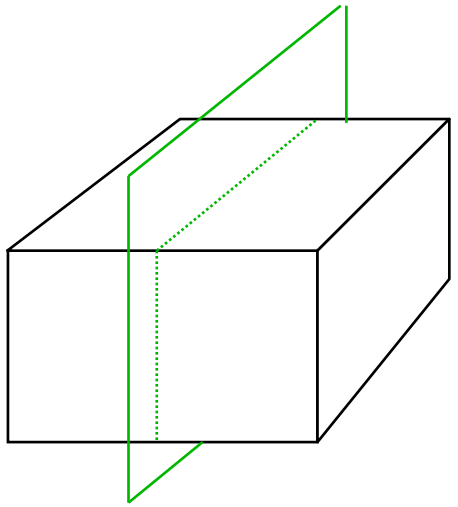






Geometric Partitioning (for meshes in space)

- Use "near neighbor" idea of planar graphs in higher dimension
- Intuition from regular 3D mesh:
- k by k by k mesh of $n = k^3$ nodes
  - Edges to 6 nearest neighbors
  - Partition by taking plane parallel to axes
  - Cuts $k^2 = n^{2/3}$ edges
- For general "3D" graphs
  - Need a notion of well-shaped
  - (Any graph fits in 3D without crossings!)



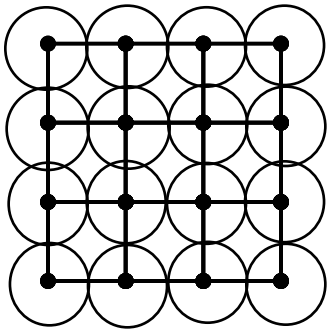
Generalizing planar separators to higher dimensions

- *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 $O(\alpha * k^{1/d} * n^{(d-1)/d})$ nodes
- When d=2, same as Lipton/Tarjan
- Algorithm:
  - Choose a sphere S in R^d
  - Edges that S "cuts" form edge separator E_S
  - Build N_S from E_S
  - Choose "randomly", so that it satisfies Theorem with high probability

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 R^d such that no point in R^d is strictly interior to more than k disks
- Def: An (α,k) overlap graph is a graph defined in terms of $\alpha >= 1$ 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

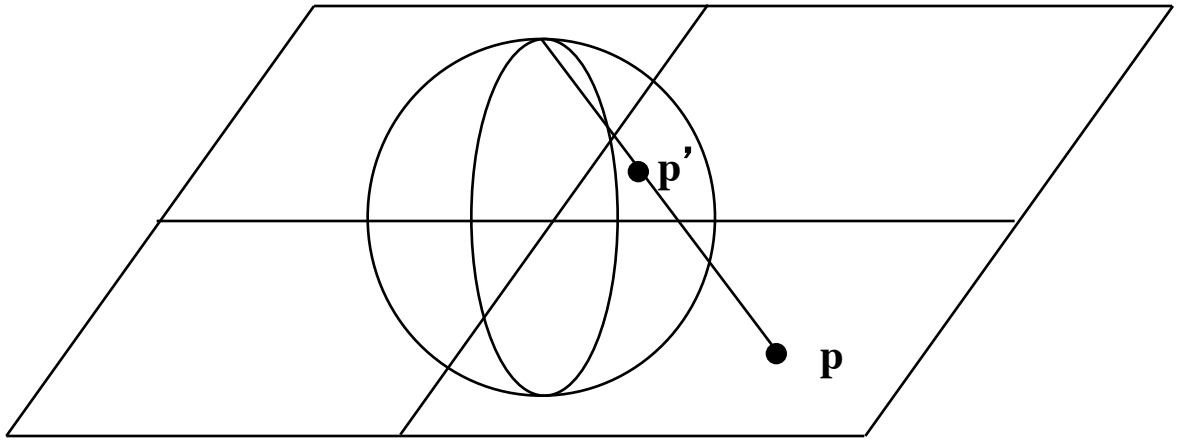
An n-by-n mesh is a (1,1) overlap graph
Every planar graph is (a,k) overlap for some a,k



2D Mesh is (1,1) overlap graph

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



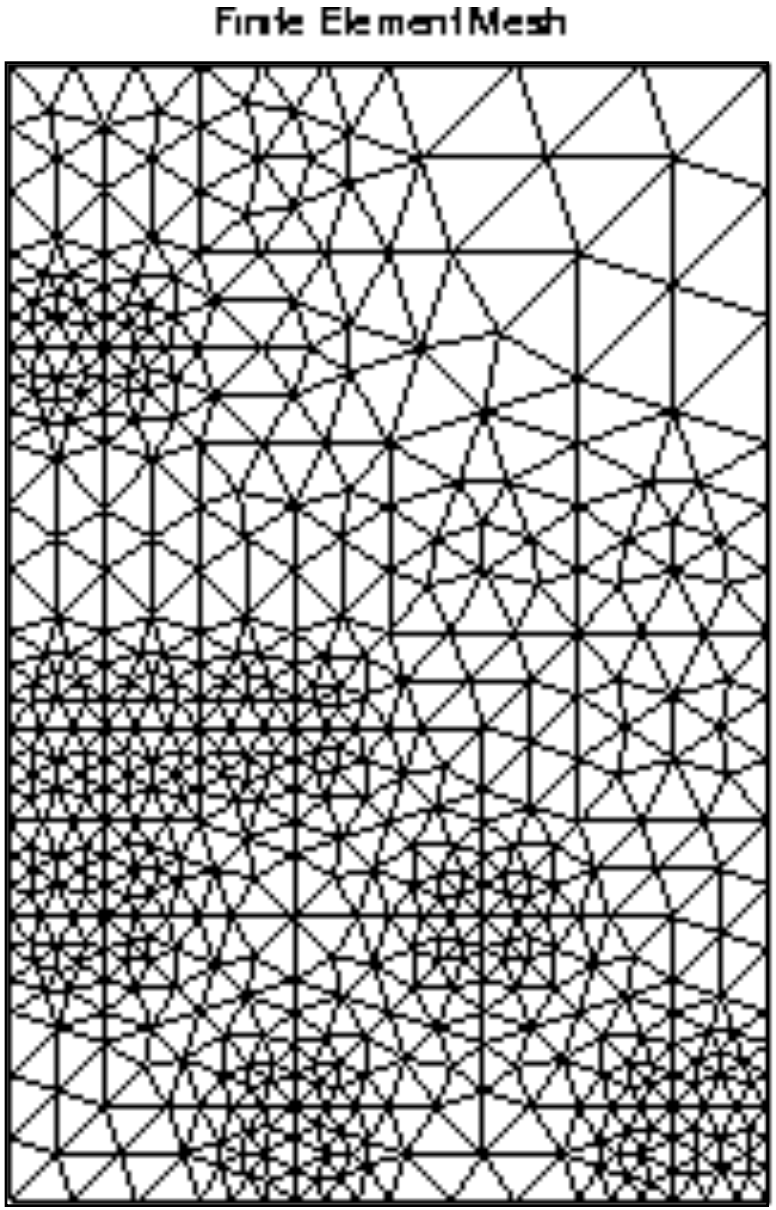
$$p = (x,y) \qquad p' = (2x,2y,x^2 + y^2 - 1) / (x^2 + y^2 + 1)$$

  - Similar in higher dimensions

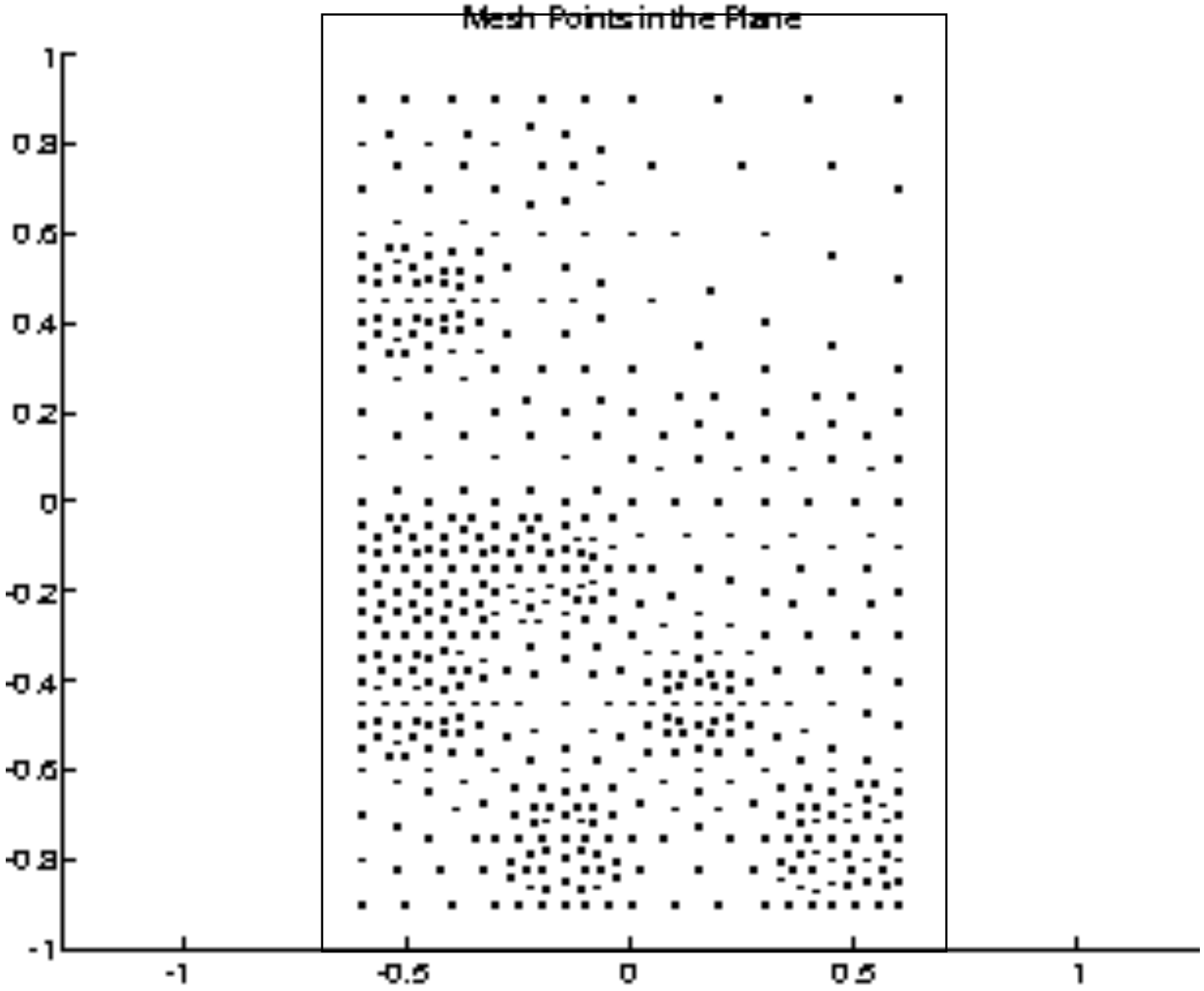
Choosing a Random Sphere

- Do stereographic projection from R^d to sphere in R^{d+1}
- Find centerpoint of projected points
  - Any plane through centerpoint divides points ~evenly
  - There is a linear programming algorithm, cheaper heuristics
- *Conformally map* points on sphere
  - *Rotate* points around origin so centerpoint at (0,…0,r) for some r
  - *Dilate* points (unproject, multiply by sqrt((1-r)/(1+r)), project)
    - this maps centerpoint to origin (0,…,0)
- Pick a random plane through origin
  - Intersection of plane and sphere is circle
- Unproject circle
  - yields desired circle C in R^d
- Create N_s: j belongs to N_s if $\alpha * D_j$ intersects C

Random Sphere Algorithm



Random Sphere Algorithm



Random Sphere Algorithm

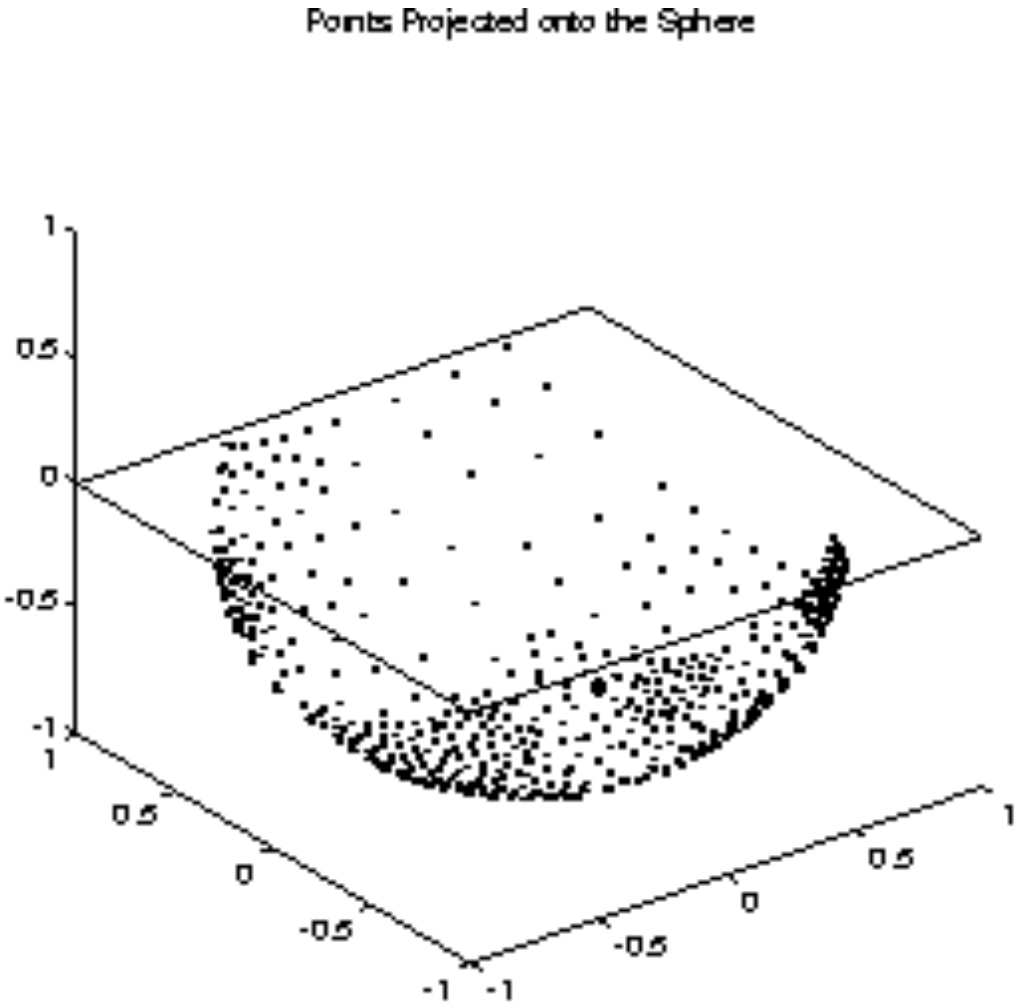
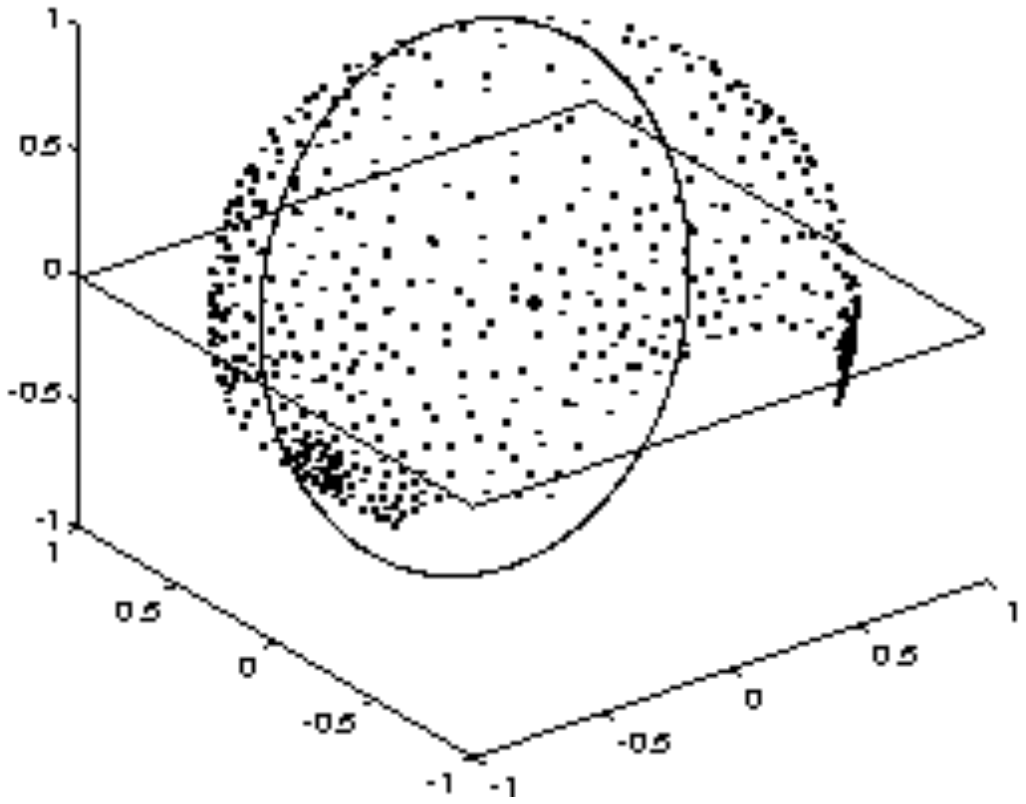
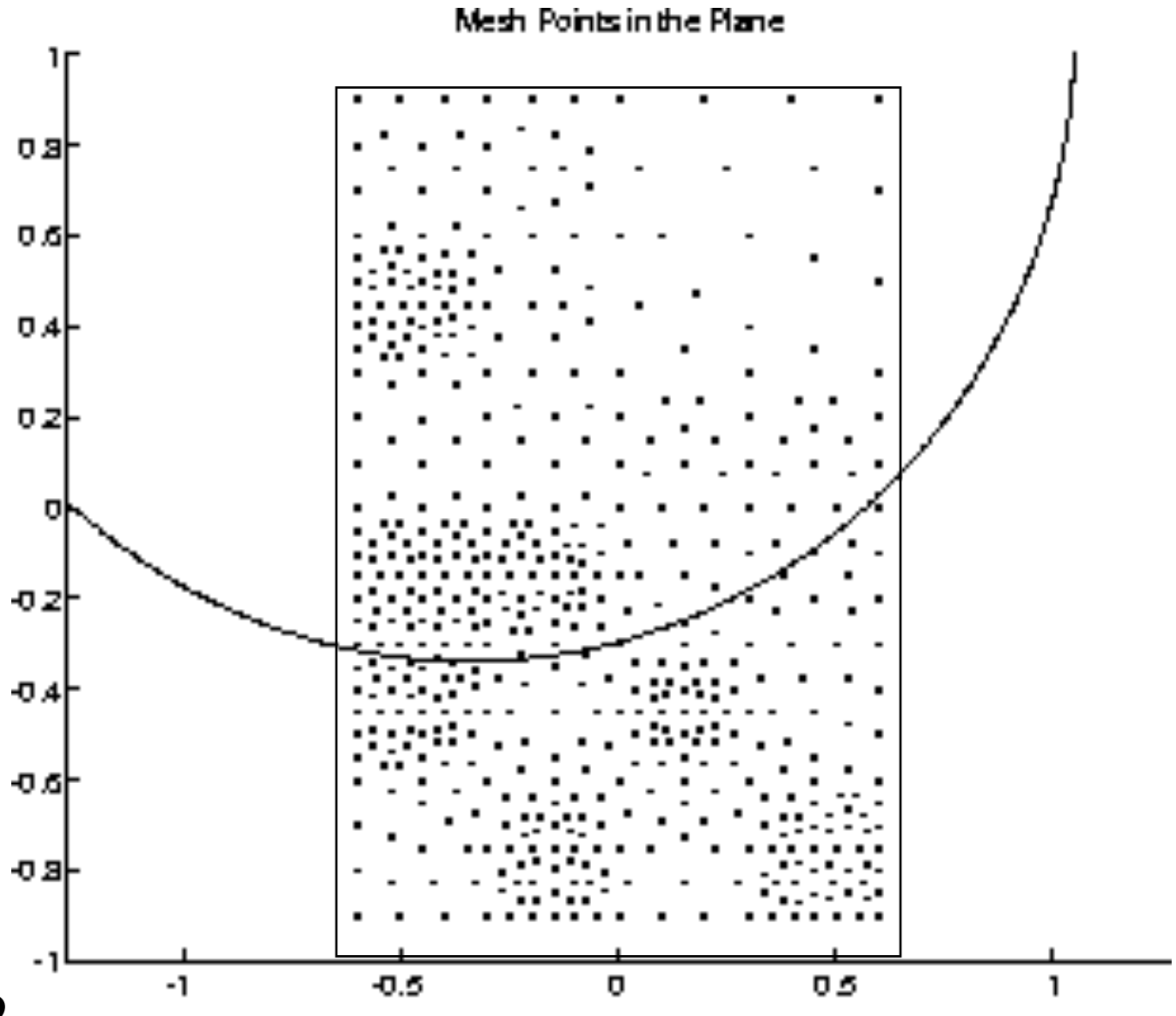


Figure 3: Projected mesh points. The large dot is the centerpoint.

Random Sphere Algorithm

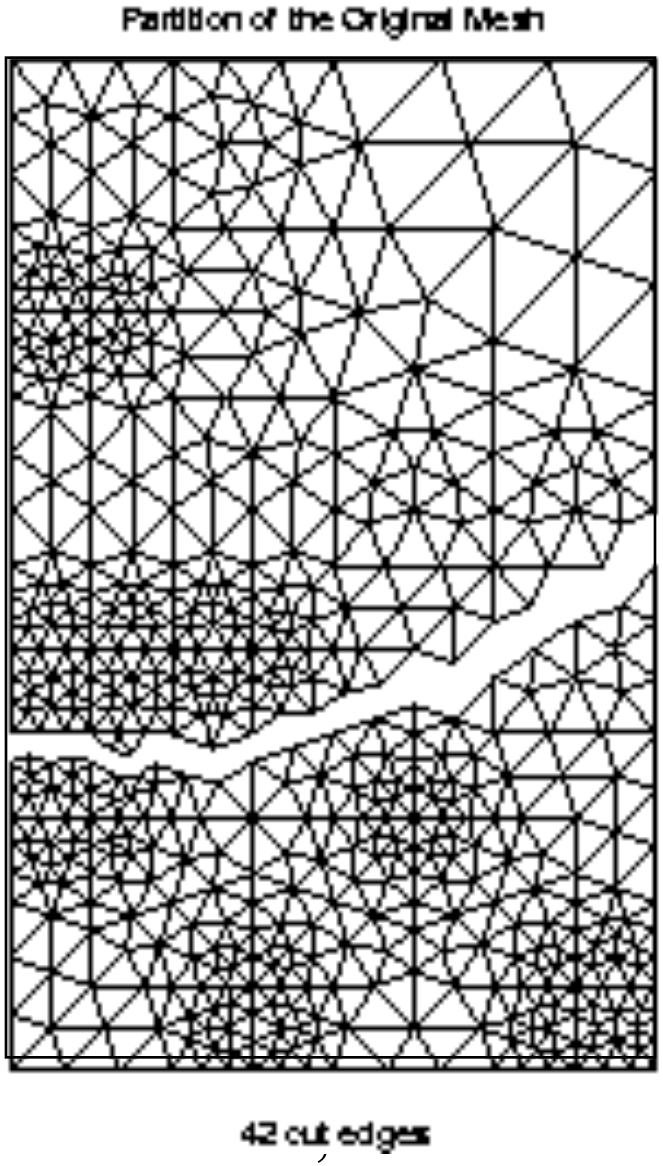


Random Sphere Algorithm



2

Figure 5: The separating circle projected back to the plane.



Multilevel Partitioning

- If G(N,E) is too big for our algorithms, what can we do?

  (1) Replace G(N,E) by a coarse approximation $G_C(N_C,E_C)$, and partition G_C instead

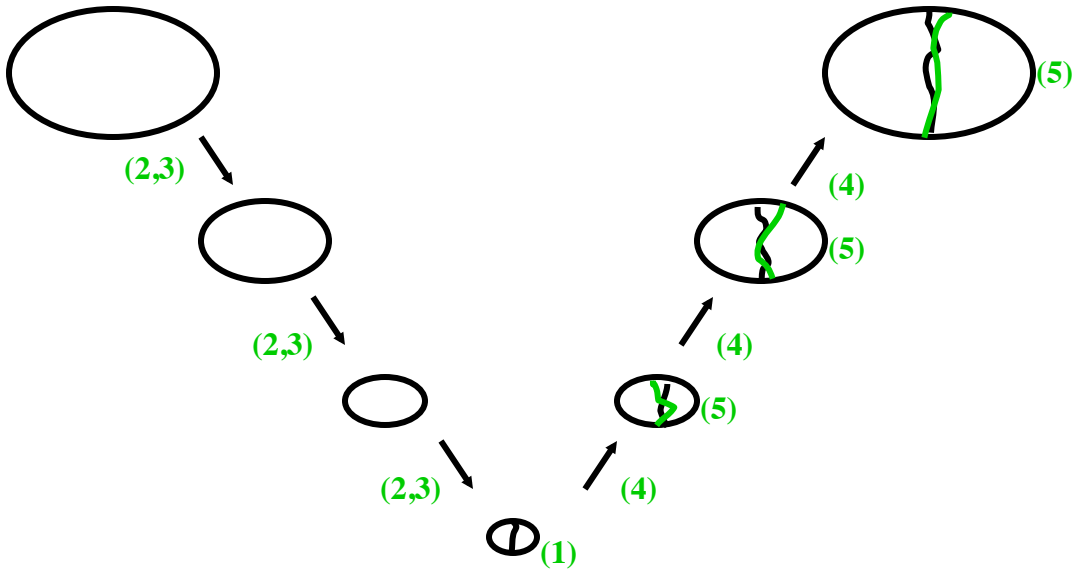
  (2) Use partition of G_C to get a rough partitioning of G, and then iteratively improve it

- What if G_C is 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
```

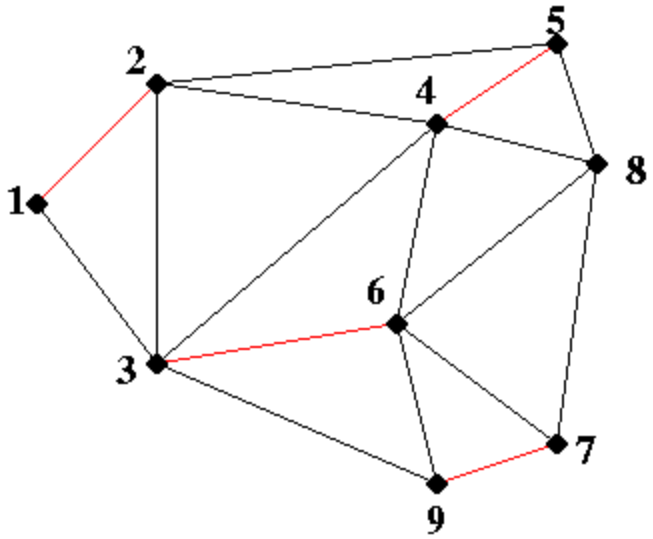
"V - cycle:"



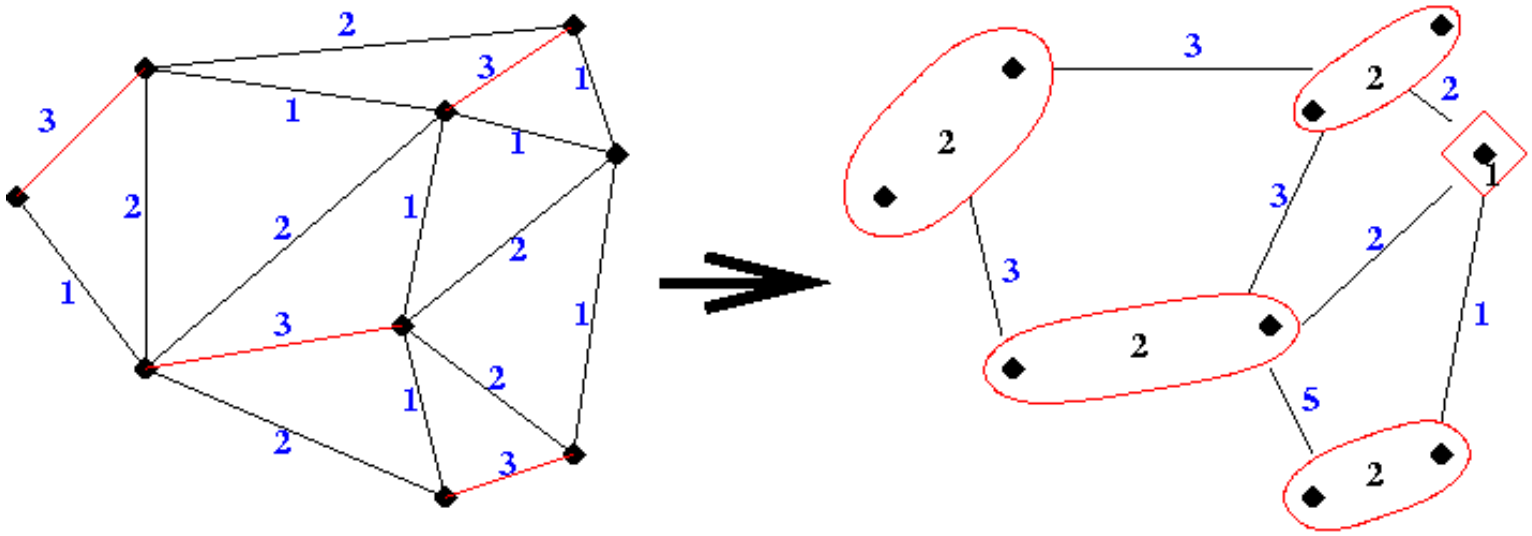
How do we

- **Coarsen?**
- **Expand?**
- **Improve?**

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

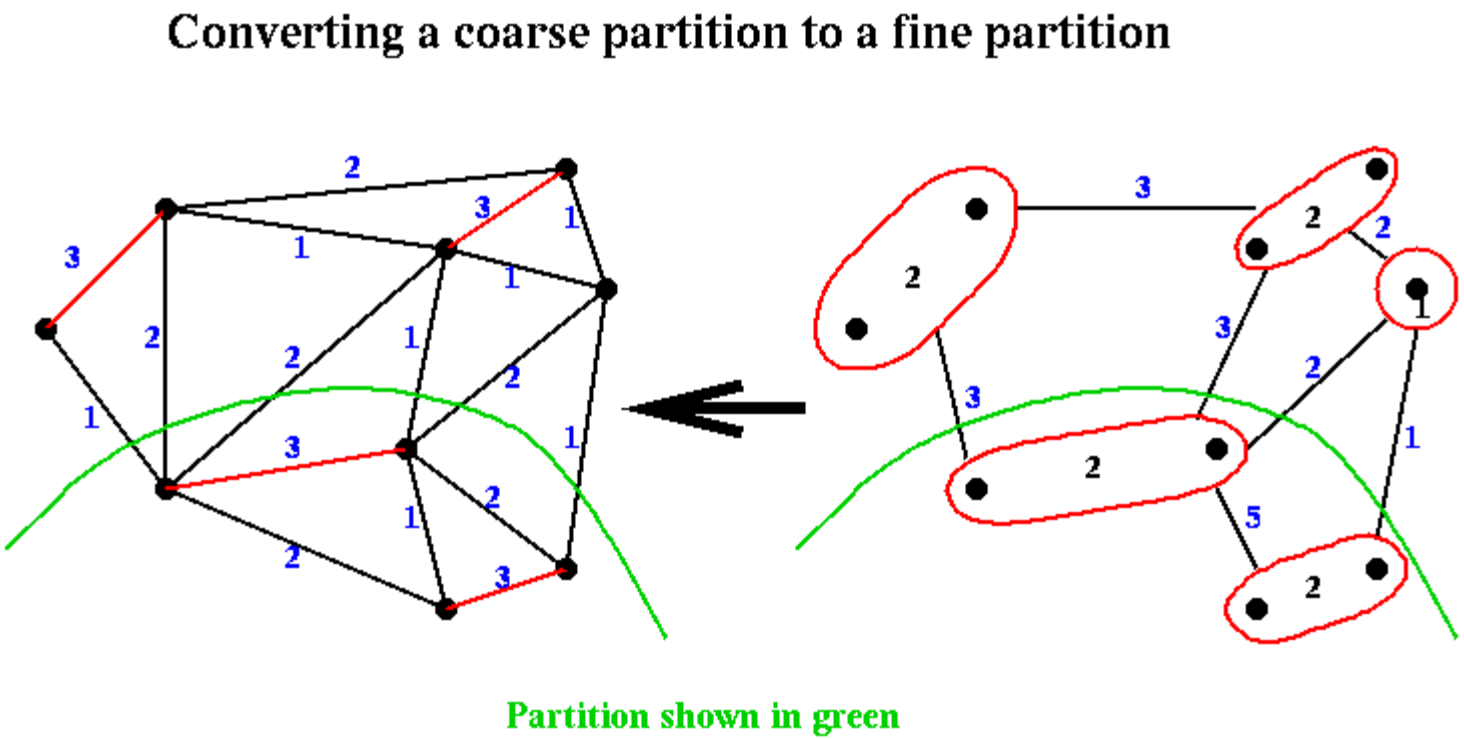
$G_c = (N_c, E_c)$

N_c is shown in red

Edge weights shown in blue

Node weights shown in black

Expanding a partition of G_c to a partition of G



CS 240A: Graph and hypergraph partitioning

- Motivation and definitions
  - Motivation from parallel computing
  - Theory of graph separators
- Heuristics for graph partitioning
  - Iterative swapping
  - Spectral
  - Geometric
  - Multilevel
- Beyond graphs
  - Shortcomings of the graph partitioning model
  - Hypergraph models of communication in MatVec
- Recent ideas: parallel partitioning, scale-free graphs, etc.

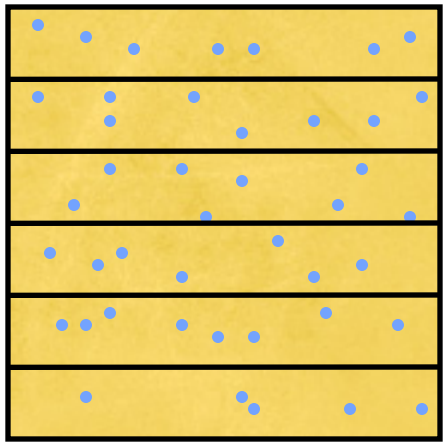
Most of the following slides adapted from:

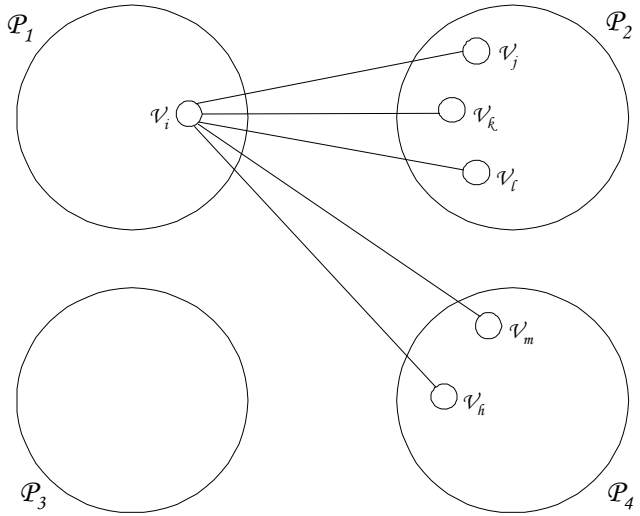
Dynamic Load Balancing for Adaptive Scientific Computations via Hypergraph Repartitioning

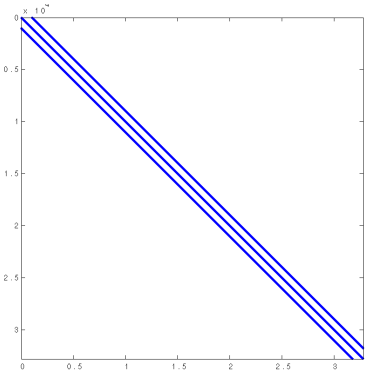
Ümit V. Çatalyürek
Department of Biomedical Informatics
The Ohio State University

Graph Models: Approximate Communication Metric for SpMV

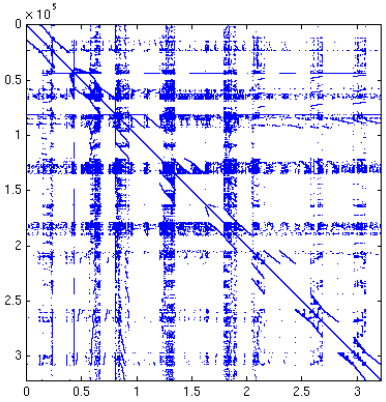
- Graph models assume
  - Weight of edge cuts = Communication volume.
- But edge cuts only *approximate* communication volume.
- Good enough for many PDE applications.
- Not good enough for irregular problems





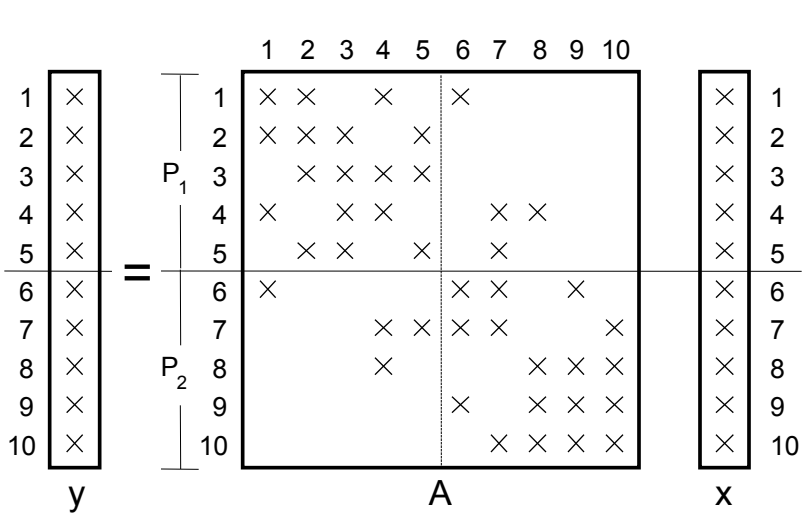


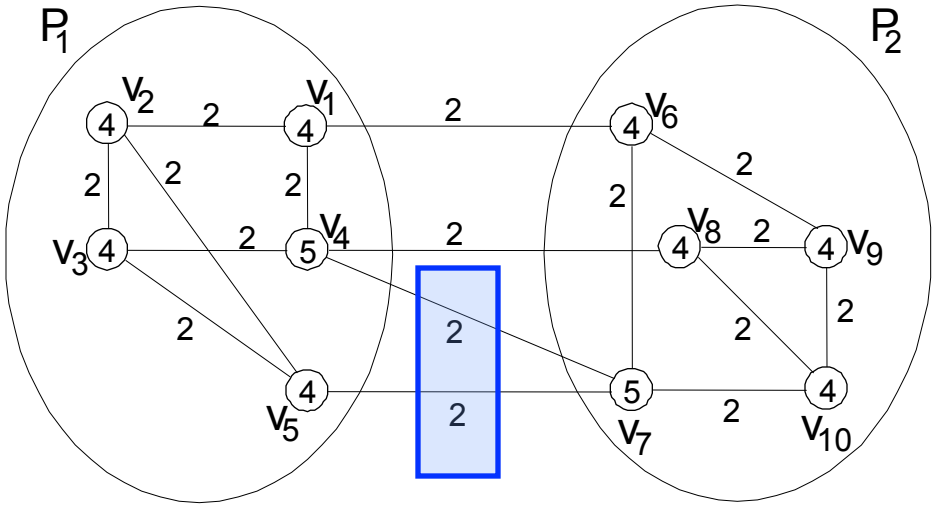
Hexahedral finite element matrix



Xyce ASIC matrix

Graph Model for Sparse Matrices





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

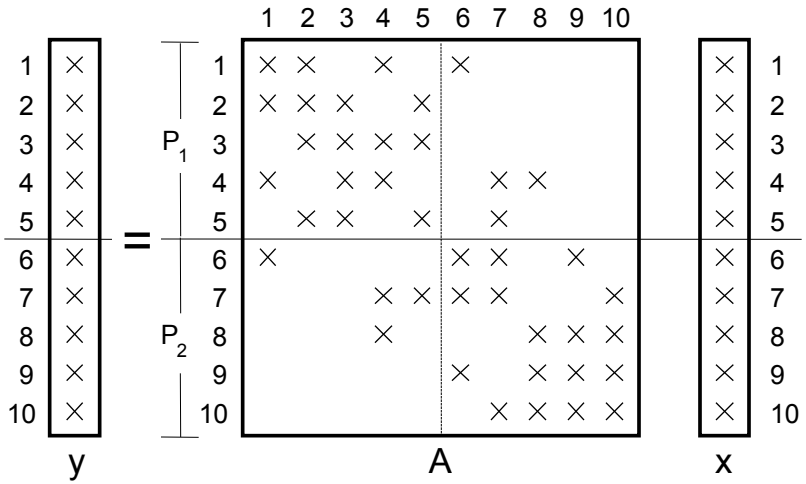
$y(i) \leftarrow y(i) + A(i,j)\, x(j)$ and $y(j) \leftarrow y(j) + A(j,i)\, x(i)$

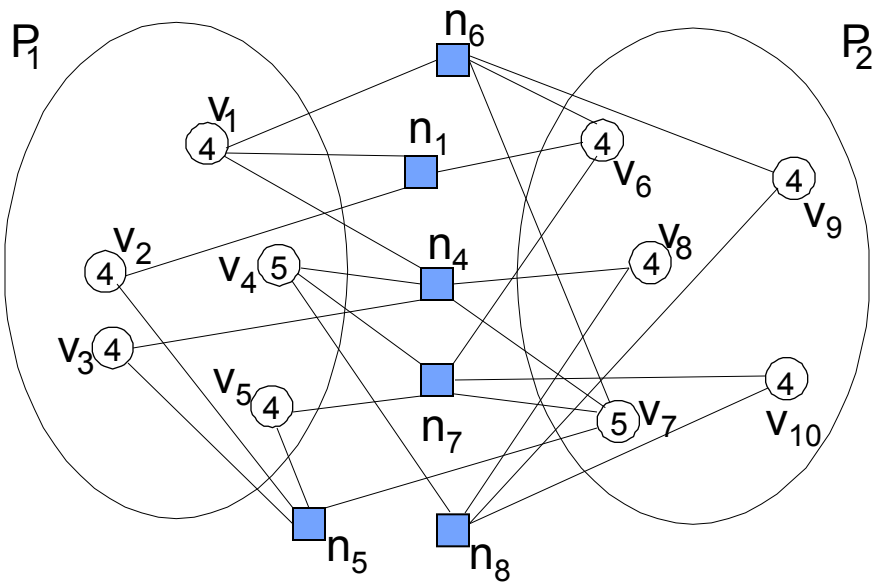
P_1 performs: $y(4) \leftarrow y(4) + A(4,7)\, x(7)$ and

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

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

Hypergraph Model for Sparse Matrices





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

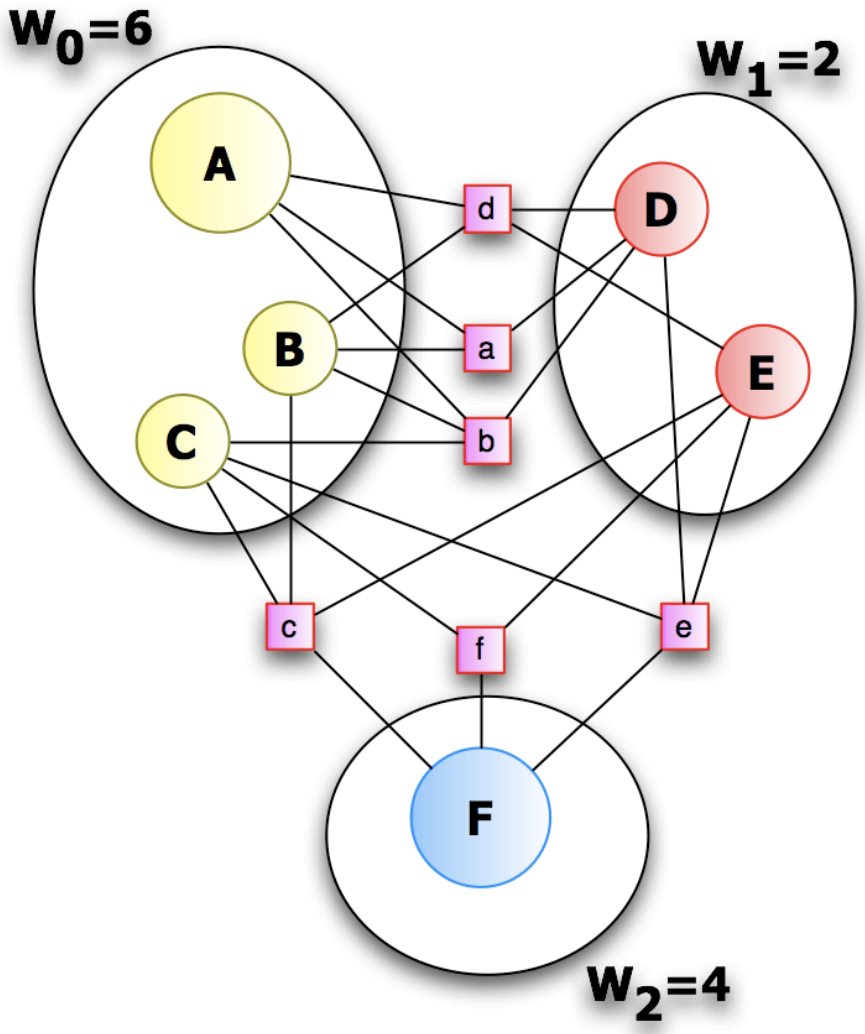
Each {vertex, net} pair represents unique nonzero net-cut metric:

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

connectivity-1 metric:

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

Hypergraph Model



c_i = 1 for all edges

Comm Vol in App = 3 x (2-1) + 3 x (3-1) = 9

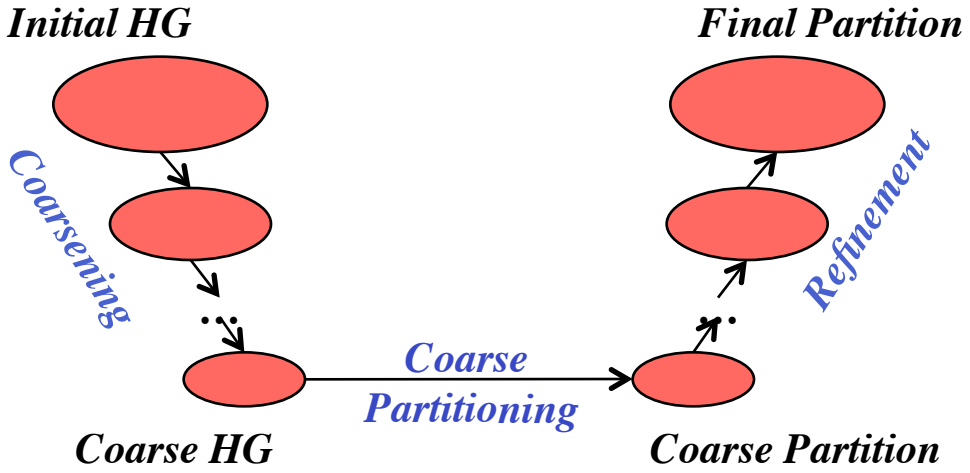
- H=(V, E) is Hypergraph
- w_i vertex weight, c_i edge cost
- P = {V_1, V_2, … , V_k} is k-way partition
- λ_i : #parts edge e_i connects
- cut(P)= $\sum_{e_i \in E} (\lambda_i - 1) \times c_i$
- cut(P) = total comm volume

CS 240A: Graph and hypergraph partitioning

- Motivation and definitions
  - Motivation from parallel computing
  - Theory of graph separators
- Heuristics for graph partitioning
  - Iterative swapping
  - Spectral
  - Geometric
  - Multilevel
- Beyond graphs
  - Shortcomings of the graph partitioning model
  - Hypergraph models of communication in MatVec
- Recent ideas: parallel partitioning, scale-free graphs, etc.

Multilevel Scheme (Serial & Parallel)

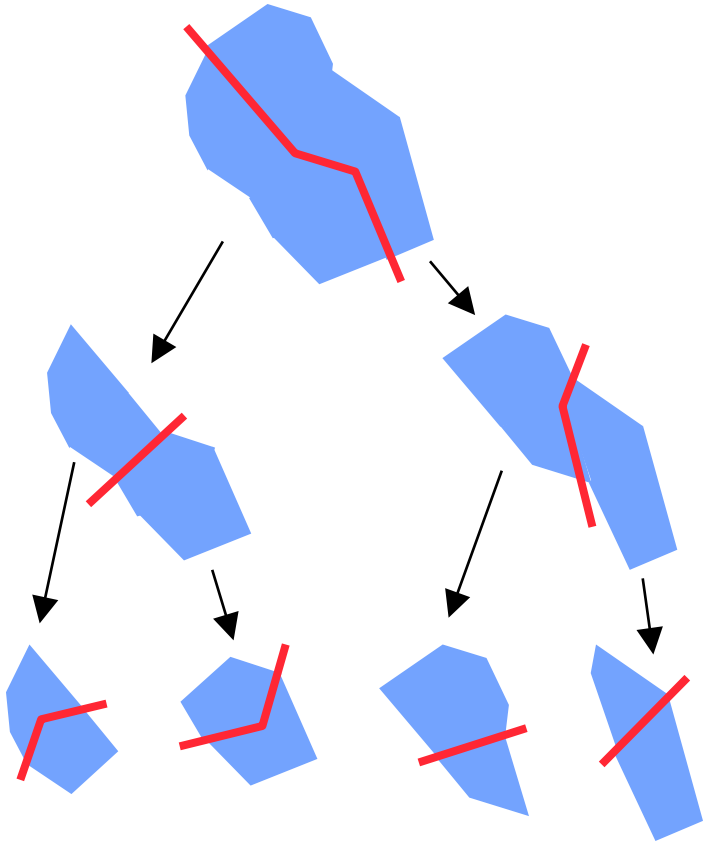
- Multilevel hypergraph partitioning (Çatalyürek, Karypis)
  - Analogous to multilevel graph partitioning (Bui&Jones, Hendrickson&Leland, Karypis&Kumar).
  - Coarsening: reduce HG to smaller representative HG.
  - Coarse partitioning: assign coarse vertices to partitions.
  - Refinement: improve balance and cuts at each level.



Multilevel Partitioning V-cycle

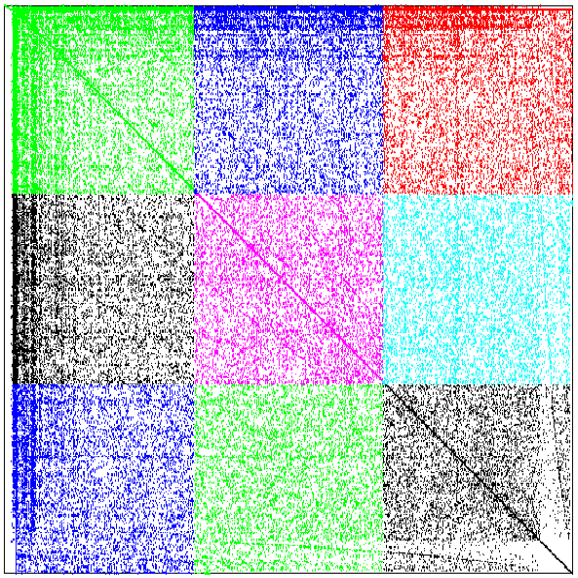
Recursive Bisection

- Recursive bisection approach:
  - Partition data into two sets.
  - Recursively subdivide each set into two sets.
  - Only minor modifications needed to allow $P \neq 2^n$.
- Two split options:
  - Split *only the data* into two sets; use all processors to compute each branch.
  - Split *both the data and processors* into two sets; solve branches in parallel.



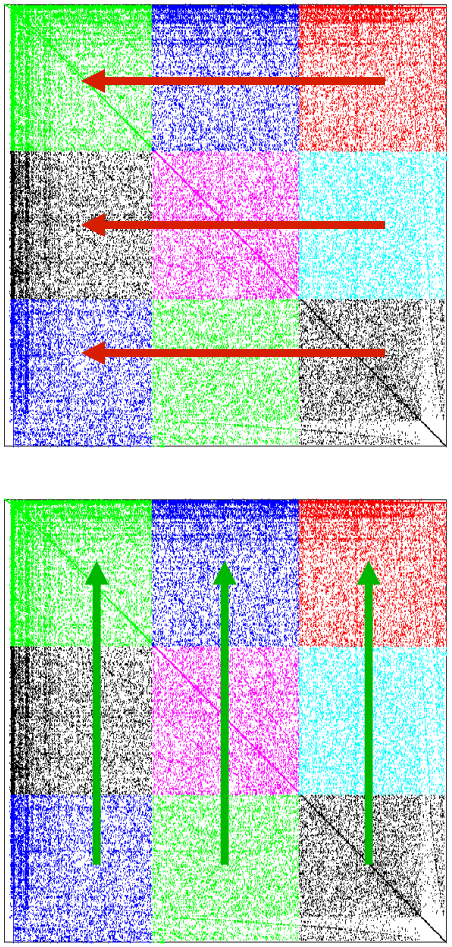
Data Layout

- Matrix representation of Hypergraphs (Çatalyürek & Aykanat)
  - vertices == columns
  - nets (hyperedges) == rows
- 2D data layout within partitioner
- Vertex/hyperedge broadcasts to only *sqrt*(*P*) processors.



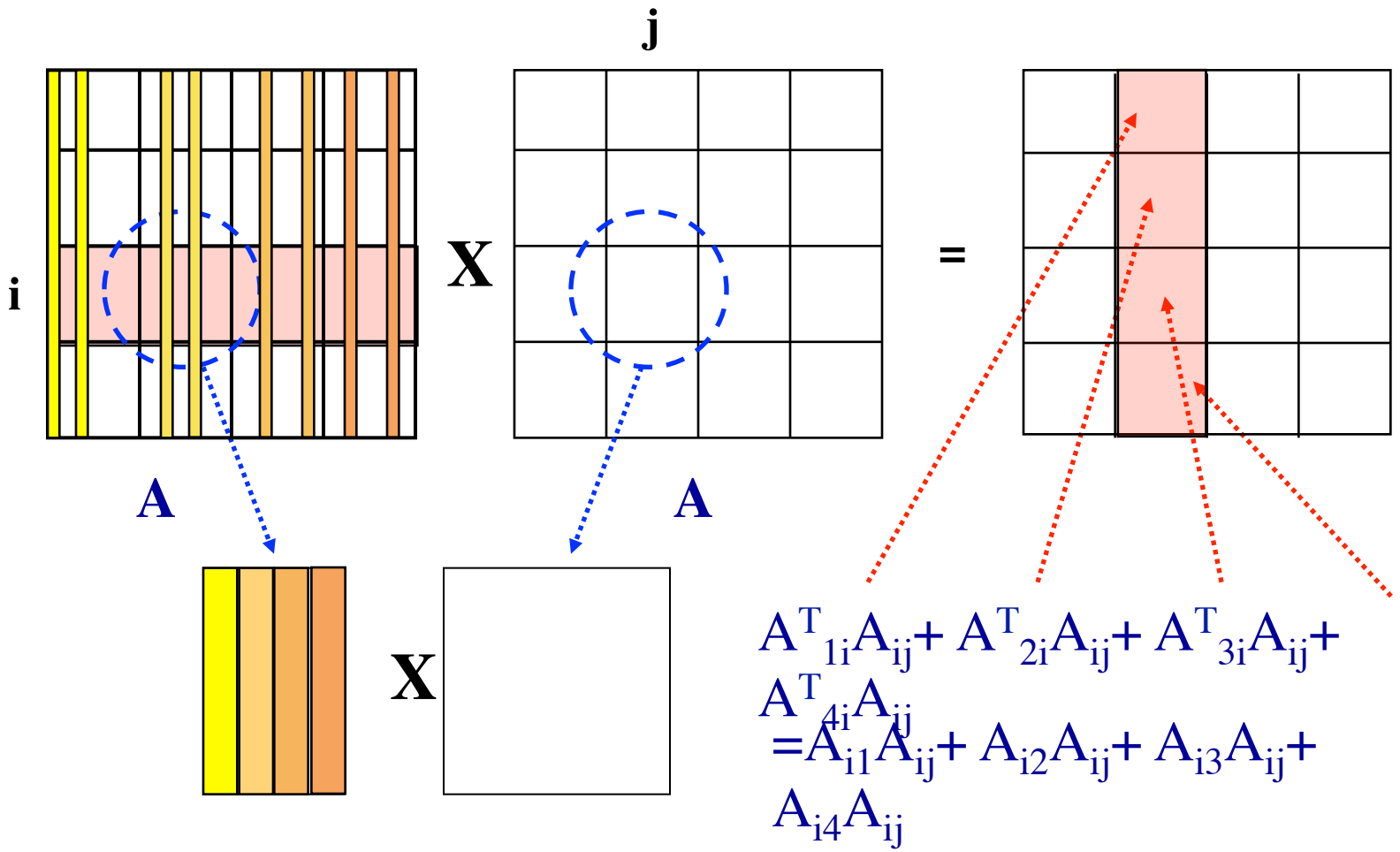
Coarsening via Parallel Matching in 2D Data Layout

- Greedy maximal weight matching
  - Heavy connectivity matching (Çatalyürek)
    Inner-product matching (Bisseling)
  - Match columns (vertices) with greatest inner product ⇒ greatest similarity in connectivity
- Each processor, on each round:
  - Broadcast candidate vertices along processor row
  - Compute (partial) inner products of received candidates with local vertices
  - Accrue inner products in processor column
  - Identify best local matches for received candidates
  - Send best matches to candidates' owners
  - Select best global match for each owned candidate
  - Send "match accepted" messages to processors owning matched vertices



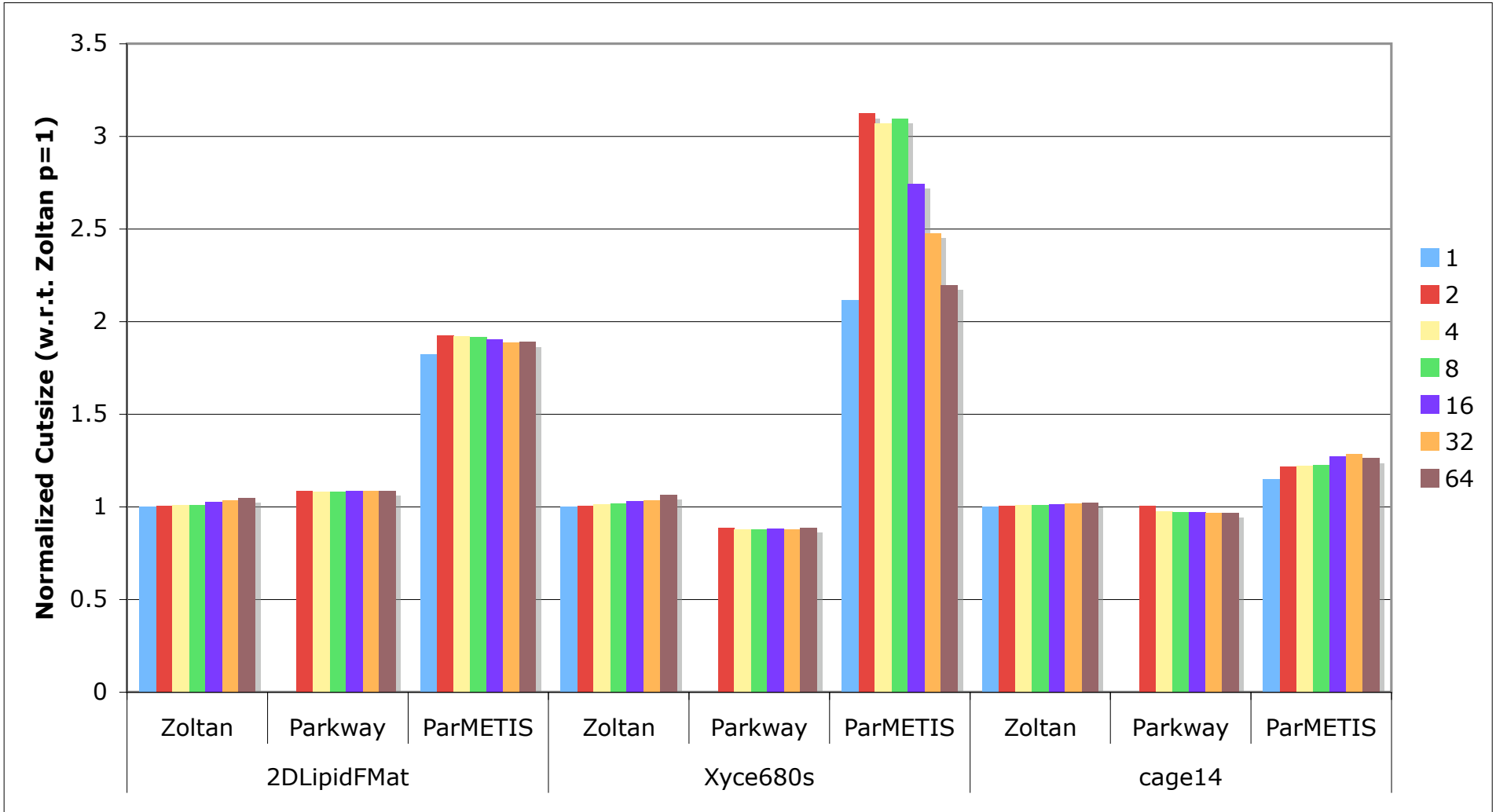
Coarsening (cont.)

The previous loop repeats until all unmatched vertices have been sent as candidates



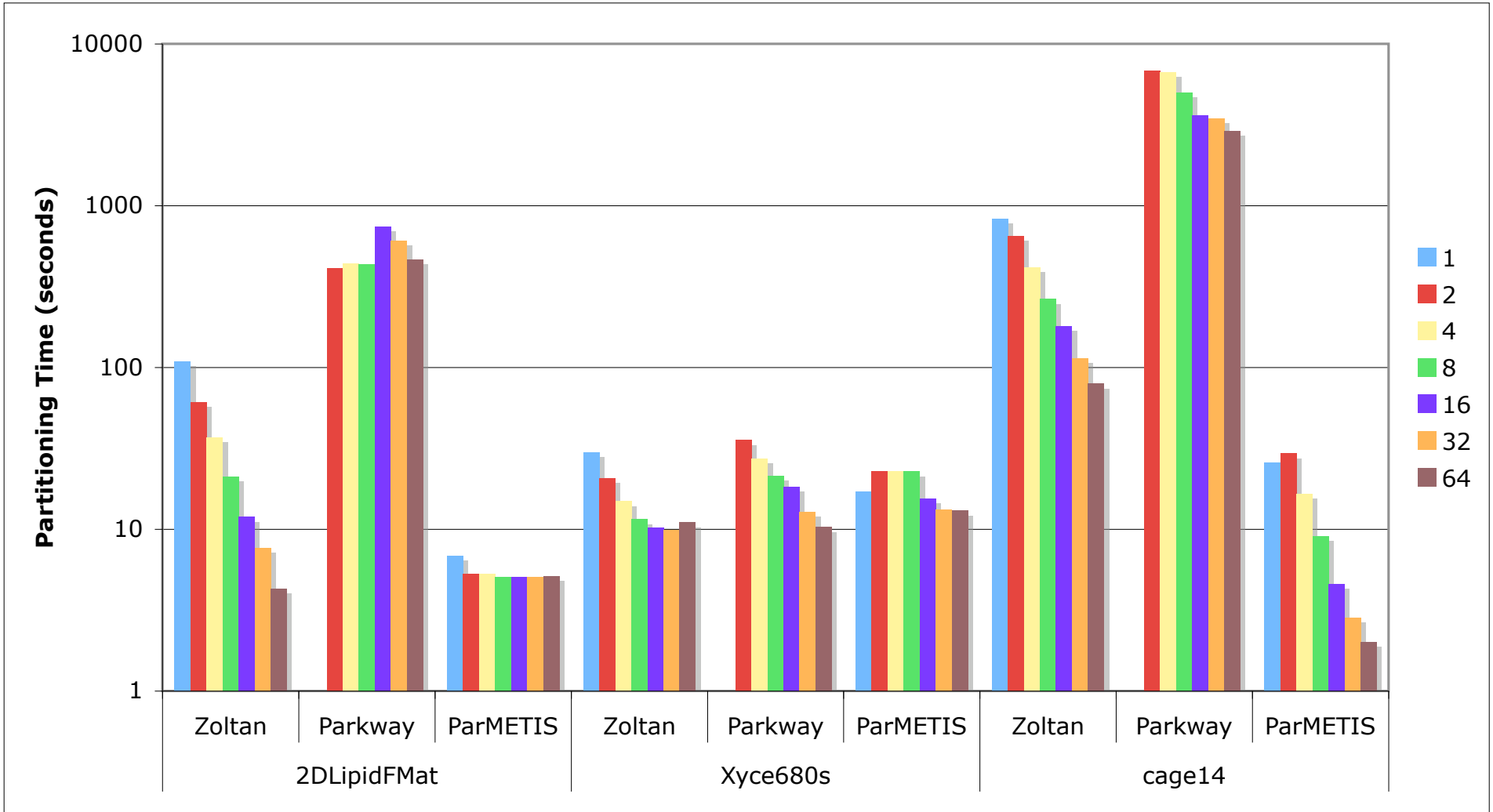
Decomposition Quality

- 64 partitions; p = 1, 2, 4, 8, 16, 32, 64
- *Much better decomposition quality with hypergraphs.*



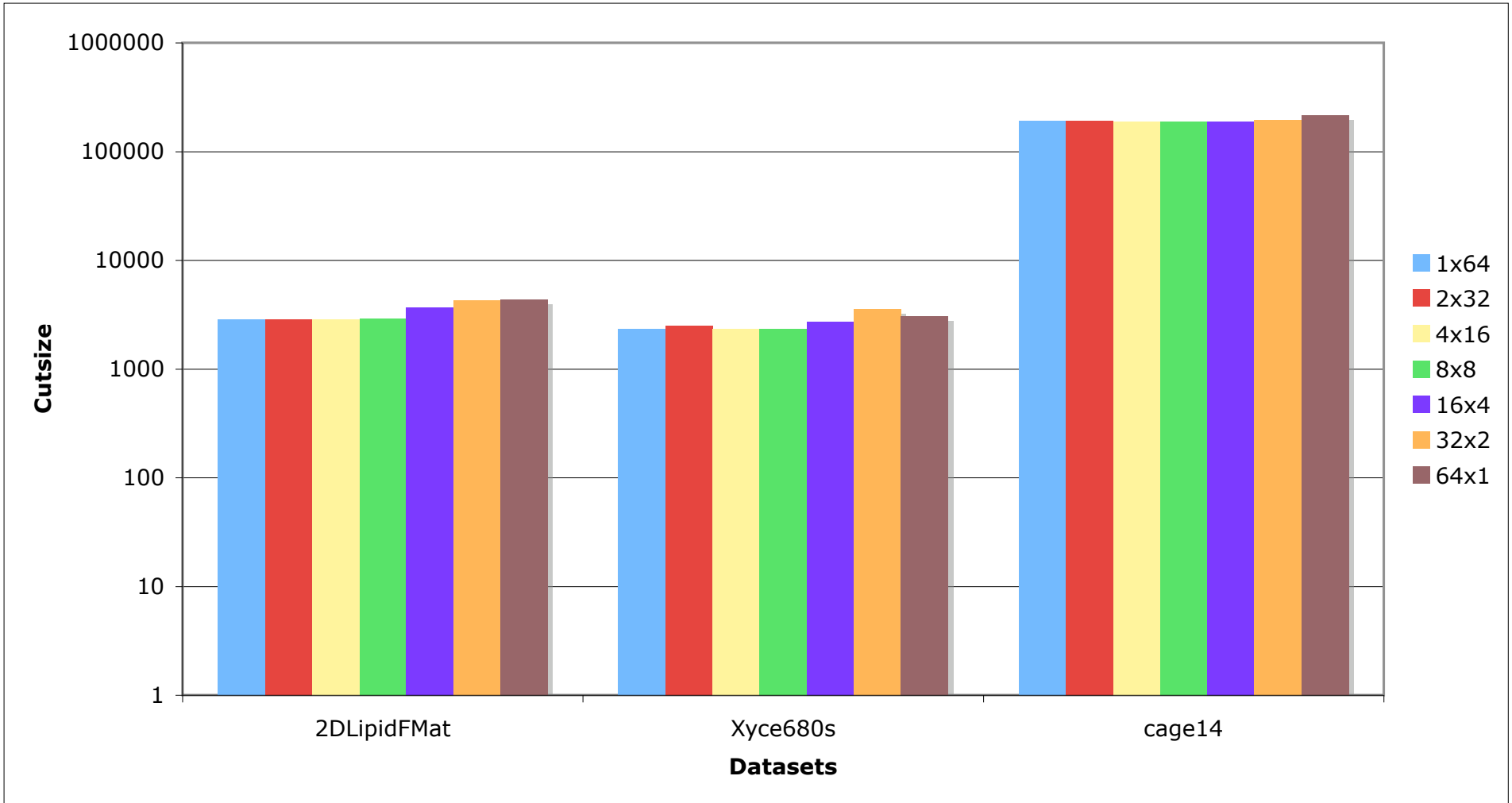
Parallel Performance

- 64 partitions; p = 1, 2, 4, 8, 16, 32, 64
- *Execution time can be higher with hypergraphs, but not always.*
- *Zoltan PHG scales as well as or better than graph partitioner.*



Zoltan 2D Distribution: Decomposition Quality

- Processor configurations: 1x64, 2x32, 4x16, 8x8, 16x4, 32x2, 64x1.
- *Configuration has little affect on decomposition quality.*



Zoltan 2D Distribution: Parallel Performance

- Processor configurations 1x64, 2x32, 4x16, 8x8, 16x4, 32x2, 64x1.
- *Configurations 2x32, 4x16, 8x8 are best.*