
CS 267 Applications of Parallel Computers

Lecture 13:

Graph Partitioning - I

Bob Lucas

from earlier lectures by Jim Demmel and Dave Culler

www.nersc.gov/~dhbailey/cs267

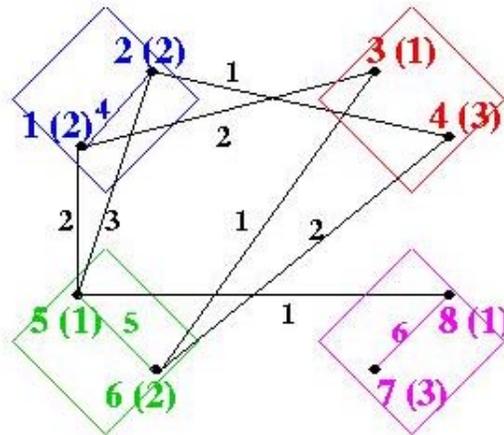
Outline of Graph Partitioning Lectures

- Review definition of Graph Partitioning problem
- Outline of Heuristics
- Partitioning with Nodal Coordinates
- Partitioning without Nodal Coordinates

- Multilevel Acceleration
 - **BIG IDEA**, will appear often in course
- Available Software
 - good sequential and parallel software available
- Comparison of Methods
- Applications

Definition of Graph Partitioning

- Given a graph $G = (N, E, W_N, W_E)$
 - N = nodes (or vertices), E = edges
 - W_N = node weights, W_E = edge weights
- Ex: $N = \{\text{tasks}\}$, $W_N = \{\text{task costs}\}$, edge (j,k) in E means task j sends $W_E(j,k)$ words to task k
- Choose a partition $N = N_1 \cup N_2 \cup \dots \cup N_p$ 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

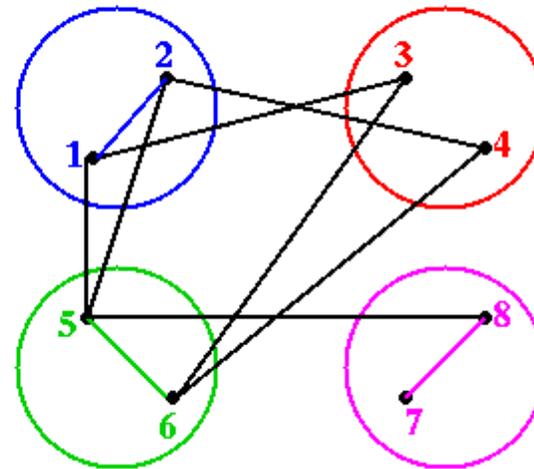
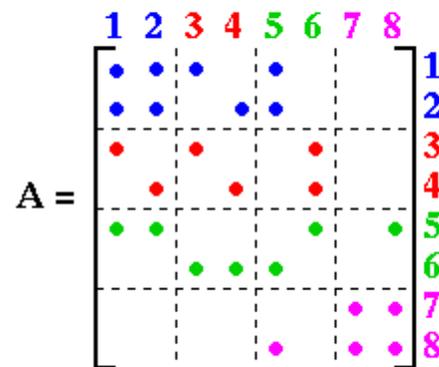


Applications

- **Telephone network design**
 - Original application, algorithm due to Kernighan
- **Load Balancing while Minimizing Communication**
- **Sparse Matrix times Vector Multiplication**
 - Solving PDEs
 - $N = \{1, \dots, n\}$, $(j, k) \in E$ if $A(j, k)$ nonzero,
 - $W_N(j) = \# \text{nonzeros in row } j$, $W_E(j, k) = 1$
- **VLSI Layout**
 - $N = \{\text{units on chip}\}$, $E = \{\text{wires}\}$, $W_E(j, k) = \text{wire length}$
- **Sparse Gaussian Elimination**
 - Used to reorder rows and columns to increase parallelism, decrease “fill-in”
- **Physical Mapping of DNA**
- ***Evaluating Bayesian Networks?***

Sparse Matrix Vector Multiplication

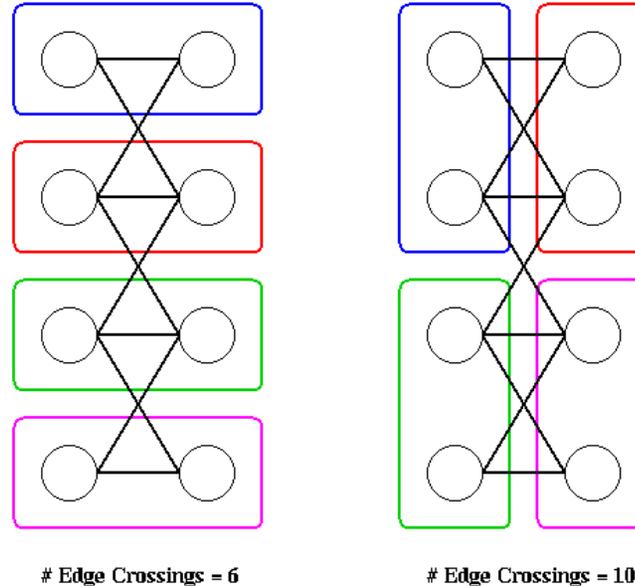
Partitioning a Sparse Symmetric Matrix



Cost of Graph Partitioning

- Many possible partitionings to search:

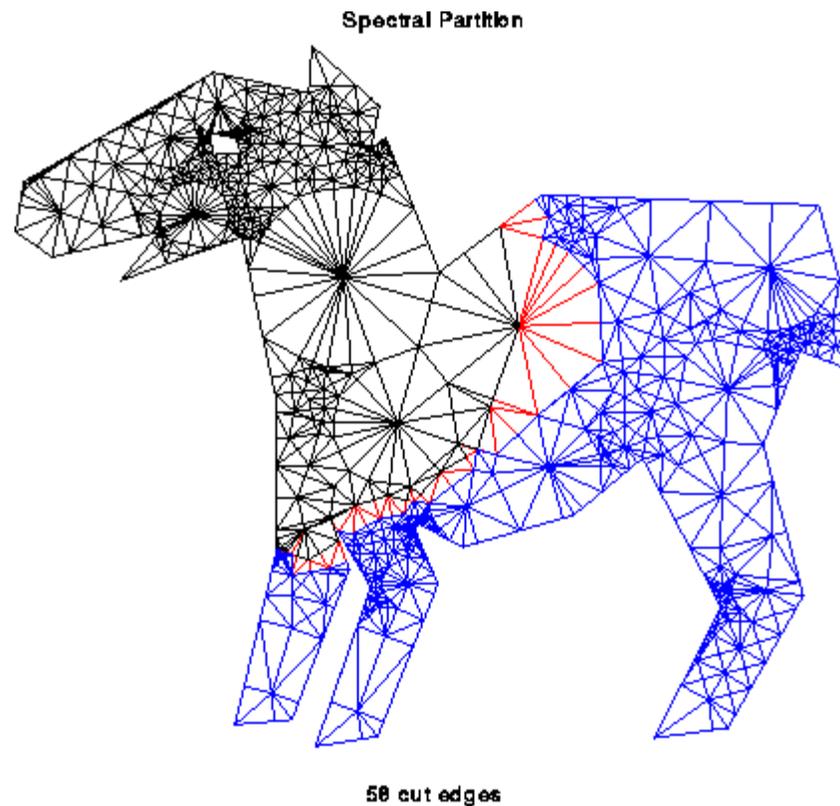
Sample Graph Partitionings



- n choose $n/2 \sim \sqrt{2n/\pi} * 2^n$ bisection possibilities
- Choosing optimal partitioning is NP-complete
 - Only known exact algorithms have cost = exponential(n)
- We need good heuristics

First Heuristic: Repeated Graph Bisection

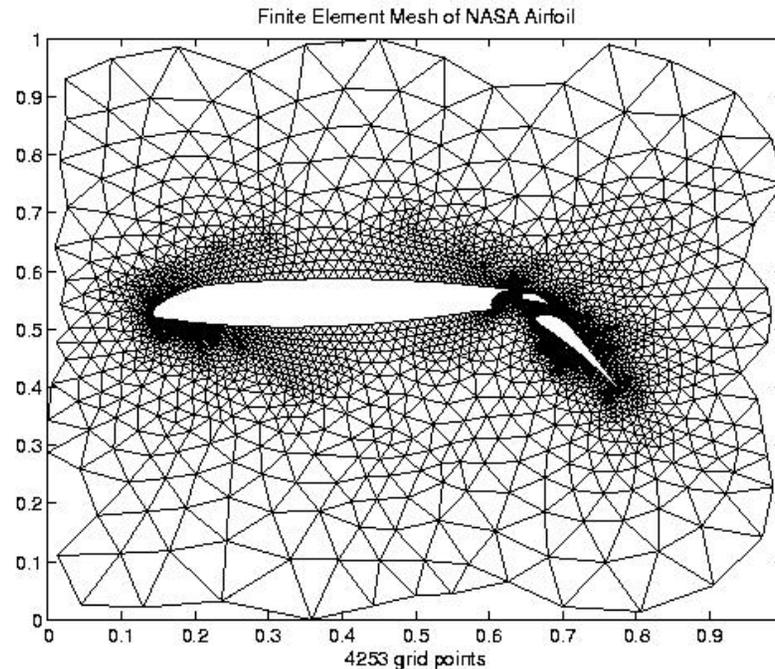
- To partition N into $2k$ parts, bisect graph recursively k times
 - Henceforth discuss mostly graph bisection



Overview of Partitioning Heuristics for Bisection

- **Partitioning with Nodal Coordinates**

- Each node has x,y,z coordinates
- Partition nodes by partitioning space



- **Partitioning without Nodal Coordinates**

- Sparse matrix of Web: $A(j,k) = \#$ times keyword j appears in URL k

- **Multilevel acceleration (BIG IDEA)**

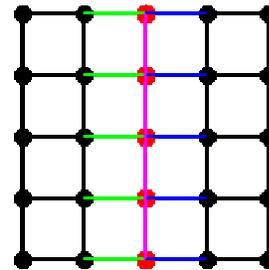
- Approximate problem by “coarse graph”, do so recursively

Edge Separators vs. Vertex Separators of $G(N,E)$

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

Edge Separators and Vertex Separators

E_s = green edges or blue edges
 N_s = red vertices



- Making an N_s from an E_s : pick one endpoint of each edge in E_s
 - How big can $|N_s|$ be, compared to $|E_s|$?
- Making an E_s from an N_s : pick all edges incident on N_s
 - How big can $|E_s|$ be, compared to $|N_s|$?
- We will find Edge or Vertex Separators, as convenient

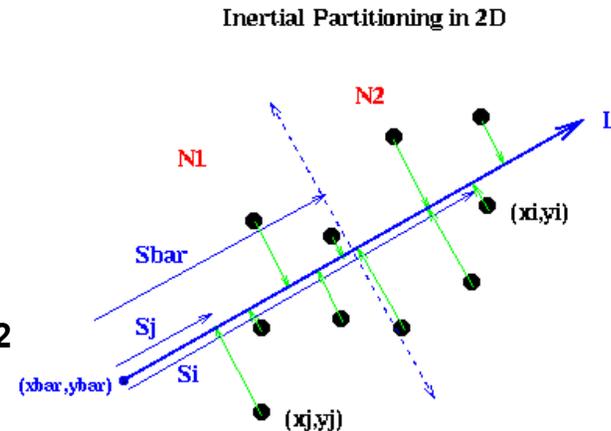
Graphs with Nodal Coordinates - Planar graphs

- 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 $|N_s| = m = \sqrt{|N|}$ (see last slide for $m=5$)
- *Theorem* (Tarjan, Lipton, 1979): If G is planar, $\exists N_s$ such that
 - $N = N_1 \cup N_s \cup N_2$ is a partition,
 - $|N_1| \leq 2/3 |N|$ and $|N_2| \leq 2/3 |N|$
 - $|N_s| \leq \sqrt{8 * |N|}$
- Theorem motivates intuition of following algorithms

Graphs with 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 an L^\perp perpendicular to it, with half the nodes on either side

- 1) L given by $a*(x-xbar)+b*(y-ybar)=0$, with $a^2+b^2=1$; (a,b) is unit vector \perp to L
- 2) For each $n_j = (x_j,y_j)$, compute coordinate $S_j = -b*(x_j-xbar) + a*(y_j-ybar)$ along L
- 3) Let $Sbar = \text{median}(S_1, \dots, S_n)$
- 4) Let nodes with $S_j < Sbar$ be in N_1 , rest in N_2

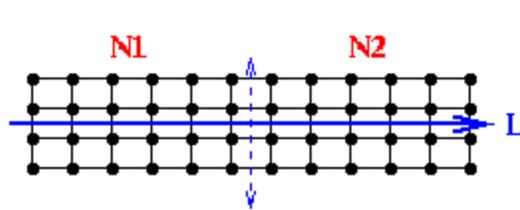


- Remains to choose L

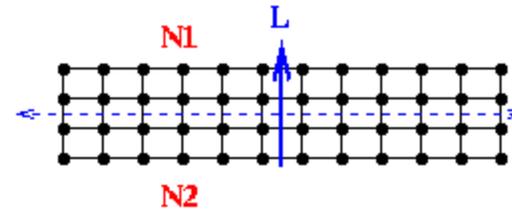
Inertial Partitioning: Choosing L

- Clearly prefer L on left below

Choosing (x,y) for inertial partitioning



Good choice of (x,y)
4 edges cut



Bad Choice of (x,y)
12 edges cut

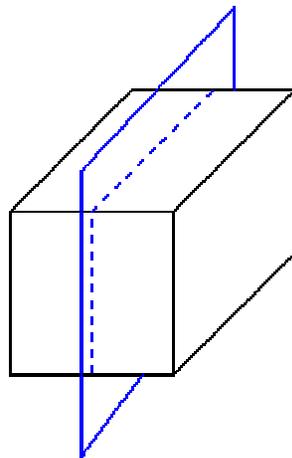
- Mathematically, choose L to be a **total least squares fit of the nodes**

- 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

Graphs with Nodal Coordinates: Random Spheres

- Emulate Lipton/Tarjan in higher dimensions than planar (2D)
- Take an n by n by n mesh of $|N| = n^3$ nodes
 - Edges to 6 nearest neighbors
 - Partition by taking plane parallel to 2 axes
 - Cuts $n^2 = |N|^{2/3} = O(|E|^{2/3})$ edges

Bisecting a 3D Grid



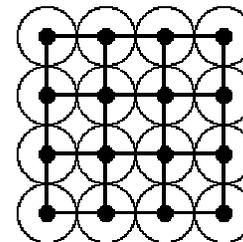
Random Spheres: Well Shaped Graphs

- Need Notion of “well shaped” graphs in 3D, higher D
 - Any graph fits in 3D without edge crossings!
- Approach due to Miller, Teng, Thurston, Vavasis
- **Def: A k -ply neighborhood system in d dimensions** is a set $\{D_1, \dots, D_n\}$ 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 $\alpha \geq 1$ and a k -ply neighborhood system $\{D_1, \dots, D_n\}$: There is a node for each D_j , and an edge from j to k if expanding the radius of the smaller of D_j and D_k by $>\alpha$ causes the two disks to overlap

Ex: n -by- n mesh is a $(1,1)$ overlap graph

Ex: Any planar graph is (α, k) overlap for some α, k

A 2D Mesh is a $(1,1)$ Overlap Graph



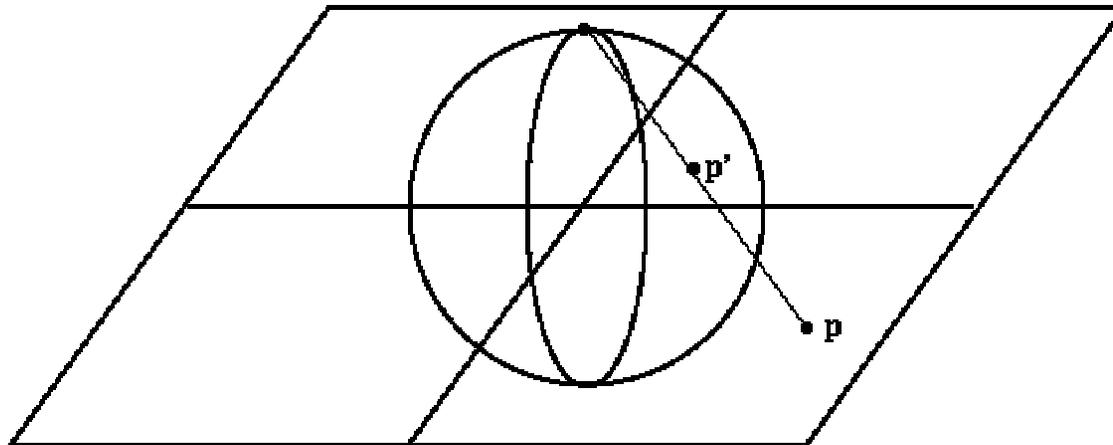
Generalizing Lipton/Tarjan to higher dimensions

- **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 $n \cdot (d+1)/(d+2)$ nodes
 - N_s has at most $O(\alpha \cdot k^{1/d} \cdot n^{(d-1)/d})$ 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

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

Stereographic projection in 2D



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

- **Similar in higher dimensions**

Choosing a Random Sphere

- Do stereographic projection from \mathbb{R}^d to sphere in \mathbb{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 $(0, \dots, 0, r)$ for some r
 - *Dilate* points (unproject, multiply by $\sqrt{(1-r)/(1+r)}$, project)
 - this maps centerpoint to origin $(0, \dots, 0)$
- **Pick a random plane through origin**
 - Intersection of plane and sphere is circle
- **Unproject circle**
 - yields desired circle C in \mathbb{R}^d
- **Create N_s : j belongs to N_s if α^*D_j intersects C**

Example of Random Sphere Algorithm (Gilbert)

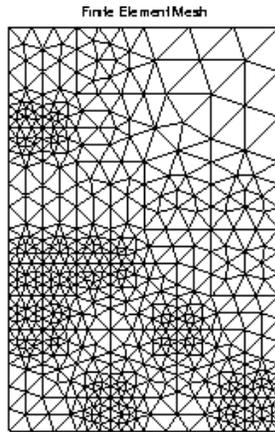


Figure 1: The input mesh.

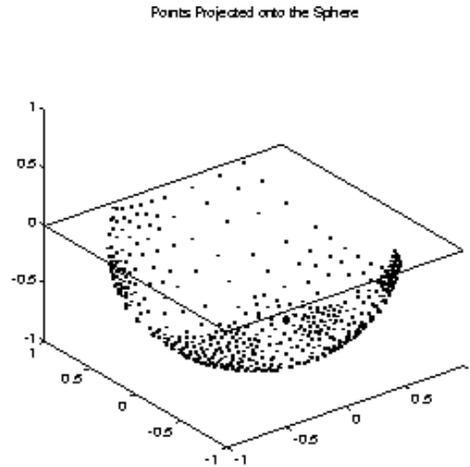


Figure 3: Projected mesh points. The large dot is the center point.

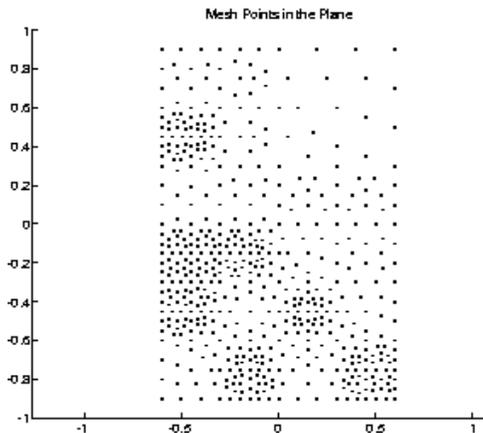
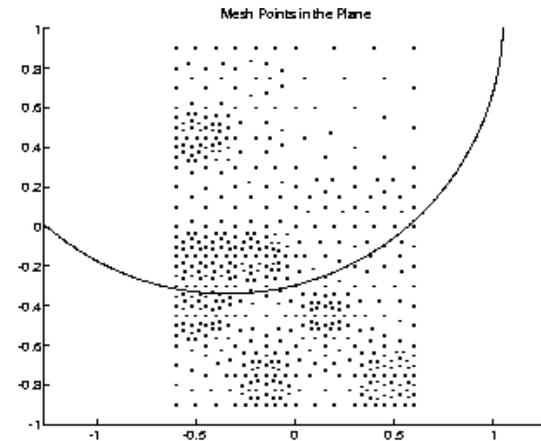
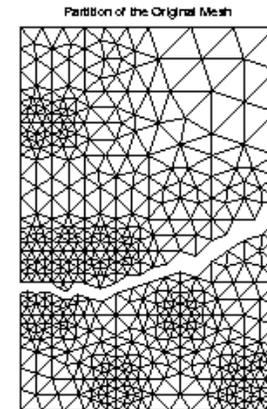
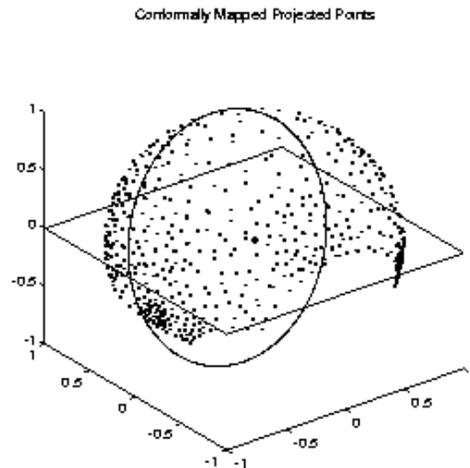
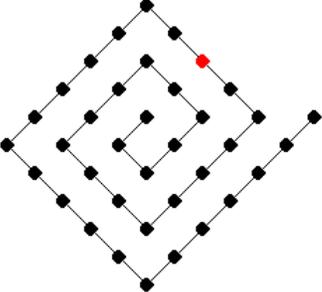


Figure 2: The mesh points.



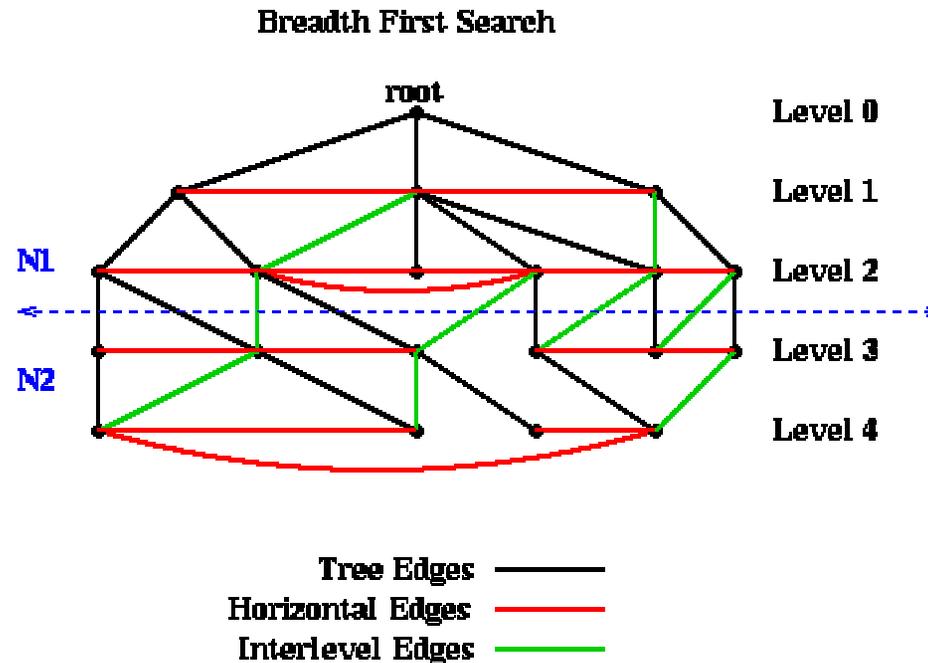
42 cut edges

Partitioning with 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
- Can be used as good starting guess for subsequent partitioners, which do examine edges
- Can do poorly if graph less connected:
 - www.cs.berkeley.edu/~demmel/cs267/lecture18/lecture18.html
 - www.parc.xerox.com/spl/members/gilbert (tech reports and SW)
 - www-sal.cs.uiuc.edu/~steng
- Details at

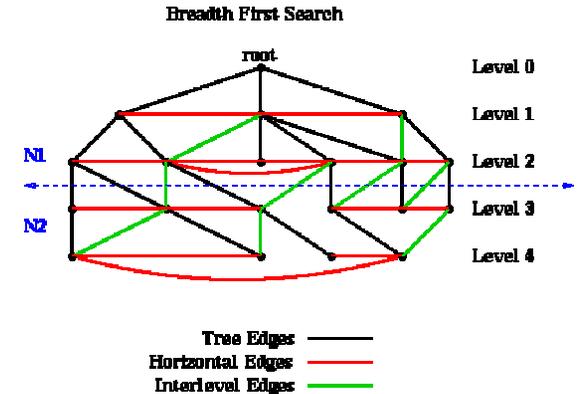
Partitioning without Nodal Coordinates- Breadth First Search (BFS)

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



Breadth First Search

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



$N_T = \{(r,0)\}$, $E_T = \text{empty set}$
 Enqueue((r,0),Q)
 Mark r

While Q not empty

(n,level) = Dequeue(Q)

For all unmarked children c of n

$N_T = N_T \cup (c, \text{level}+1)$

$E_T = E_T \cup (n,c)$

Enqueue((c,level+1),Q)

Mark c

Endfor

Endwhile

... Initially $T = \text{root } r$, which is at level 0

... Put root on initially empty Queue Q

... Mark root as having been processed

... While nodes remain to be processed

... Get a node to process

... Add child c to N_T

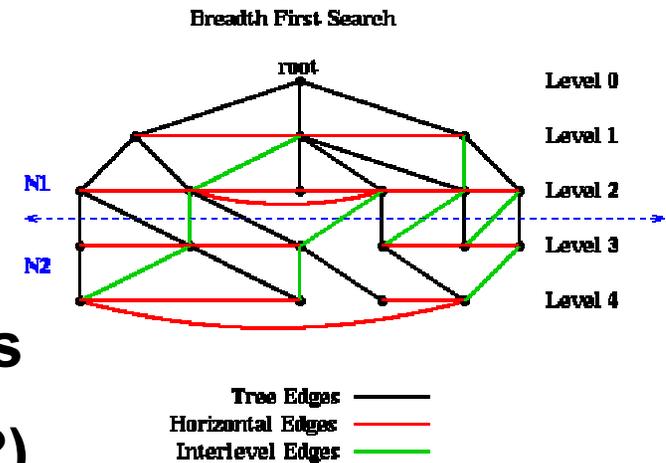
... Add edge (n,c) to E_T

... Add child c to Q for processing

... Mark c as processed

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

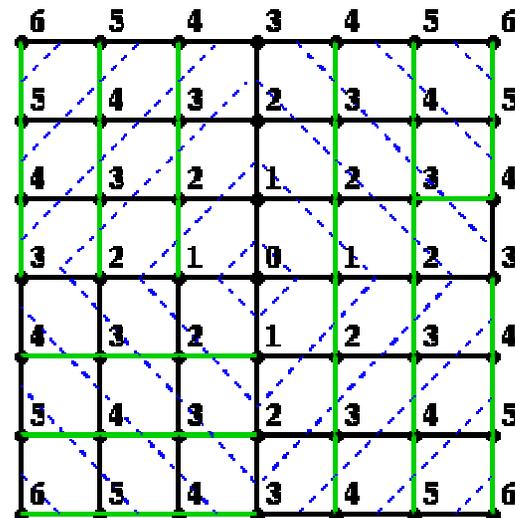


- **No edges connect nodes in levels differing by more than 1 (why?)**

- **BFS partitioning 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 $|N_2|$

Breadth First Search on a 7 by 7 Grid
Starting at the center node
Nodes labeled by level



Partitioning without nodal coordinates - Kernighan/Lin

- **Take a initial partition and iteratively improve it**
 - Kernighan/Lin (1970), cost = $O(|N|^3)$ but easy to understand
 - What else did Kernighan invent?
 - 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 $|A| = |B|$**
 - $T = \text{cost}(A, B) = \sum \{W(e) \text{ where } e \text{ connects nodes in } A \text{ and } B\}$
 - Find subsets X of A and Y of B with $|X| = |Y|$
 - Swapping X and Y should decrease cost:
 - $\text{newA} = A - X \cup Y$ and $\text{newB} = B - Y \cup X$
 - $\text{newT} = \text{cost}(\text{newA}, \text{newB}) < \text{cost}(A, B)$
- **Need to compute newT efficiently for many possible X and Y , choose smallest**

Kernighan/Lin - Preliminary Definitions

- $T = \text{cost}(A, B)$, $\text{new}T = \text{cost}(\text{new}A, \text{new}B)$
- **Need an efficient formula for newT; will use**
 - $E(a) = \text{external cost of } a \text{ in } A = \sum \{W(a,b) \text{ for } b \text{ in } B\}$
 - $I(a) = \text{internal cost of } a \text{ in } A = \sum \{W(a,a') \text{ for other } a' \text{ in } A\}$
 - $D(a) = \text{cost of } a \text{ in } A = E(a) - I(a)$
 - $E(b)$, $I(b)$ and $D(b)$ defined analogously for b in B
- **Consider swapping $X = \{a\}$ and $Y = \{b\}$**
 - $\text{new}A = A - \{a\} \cup \{b\}$, $\text{new}B = B - \{b\} \cup \{a\}$
- $\text{new}T = T - (D(a) + D(b) - 2*w(a,b)) = T - \text{gain}(a,b)$
 - $\text{gain}(a,b)$ measures improvement gotten by swapping a and b
- **Update formulas**
 - $\text{new}D(a') = D(a') + 2*w(a',a) - 2*w(a',b)$ for a' in A , $a' \neq a$
 - $\text{new}D(b') = D(b') + 2*w(b',b) - 2*w(b',a)$ for b' in B , $b' \neq b$

Kernighan/Lin Algorithm

Compute $T = \text{cost}(A,B)$ for initial A, B ... cost = $O(|N|^2)$
Repeat
 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 $\text{gain}(a,b)$... 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
 ... $(a_1,b_1), \dots, (a_k,b_k)$ and gains $\text{gain}(1), \dots, \text{gain}(k)$
 ... for $k = |N|/2$, ordered by the order in which we marked them
 Pick j maximizing $\text{Gain} = \sum_{k=1}^j \text{gain}(k)$... cost = $O(|N|)$
 ... Gain is reduction in cost from swapping (a_1,b_1) through (a_j,b_j)
 If $\text{Gain} > 0$ then ... it is worth swapping
 Update $\text{newA} = A - \{ a_1, \dots, a_k \} \cup \{ b_1, \dots, b_k \}$... cost = $O(|N|)$
 Update $\text{newB} = B - \{ b_1, \dots, b_k \} \cup \{ a_1, \dots, a_k \}$... cost = $O(|N|)$
 Update $T = T - \text{Gain}$... cost = $O(1)$
 endif
Until $\text{Gain} \leq 0$

- One pass greedily computes $|N|/2$ possible X and Y to swap, picks best