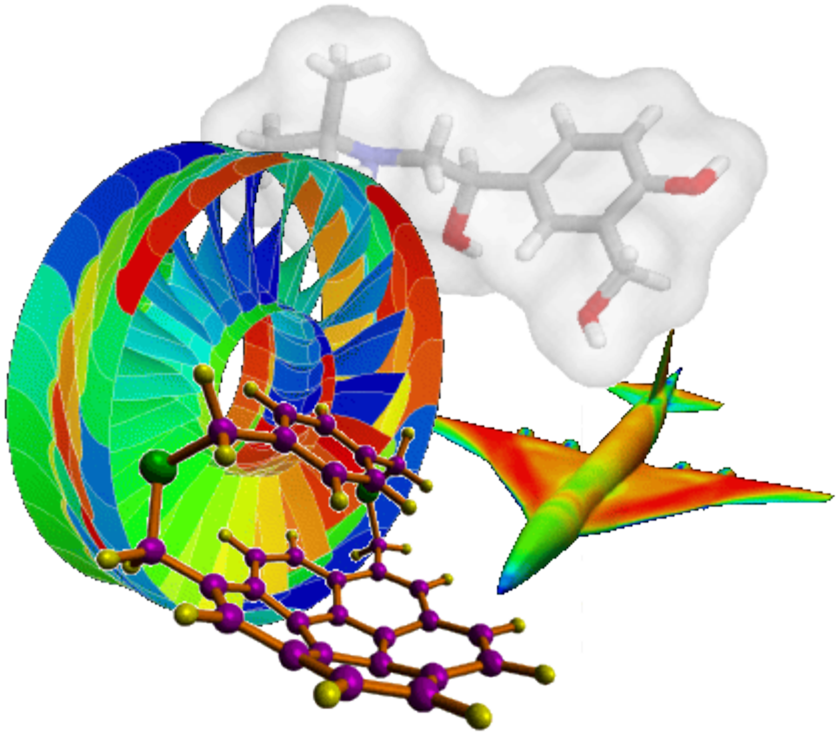
CME342 - Parallel Methods in Numerical Analysis

Graph Partitioning Algorithms



April 21-23, 2014

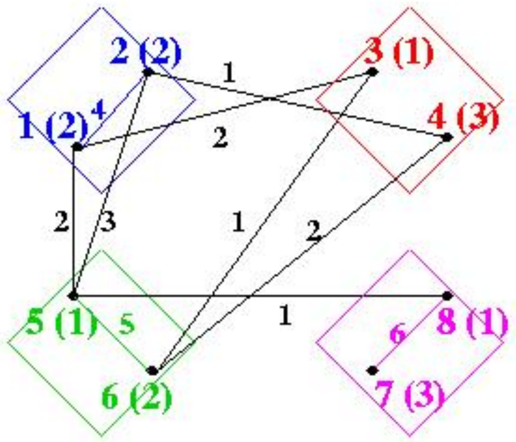
Lectures 7-8

Outline

- Definition of graph partitioning problem
- Sample applications
- N-P complete problem
- Available heuristic algorithms (with and without nodal coordinates)
  - Inertial partitioning
  - Breadth first search
  - Kernighan-Lin
  - Spectral bisection
- Multilevel acceleration (multigrid for graph partitioning problems)
- Metis, ParMetis, and others

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
- N can be thought of as tasks, W_N are the task costs, edge (j,k) in E means task j sends $W_E(j,k)$ words to task k
- Choose a partition $N = N_1 \cup N_2 \cup \ldots \cup N_P$ such that
  - The sum of the node weights in each N_j is distributed evenly (load balance)
  - The sum of all edge weights of edges connecting all different partitions is minimized (decrease parallel overhead)
- In other words, divide work evenly and minimize communication
- Partition into two parts is called graph bisection, which recursively applied can be turned into algorithms for complete graph partitioning

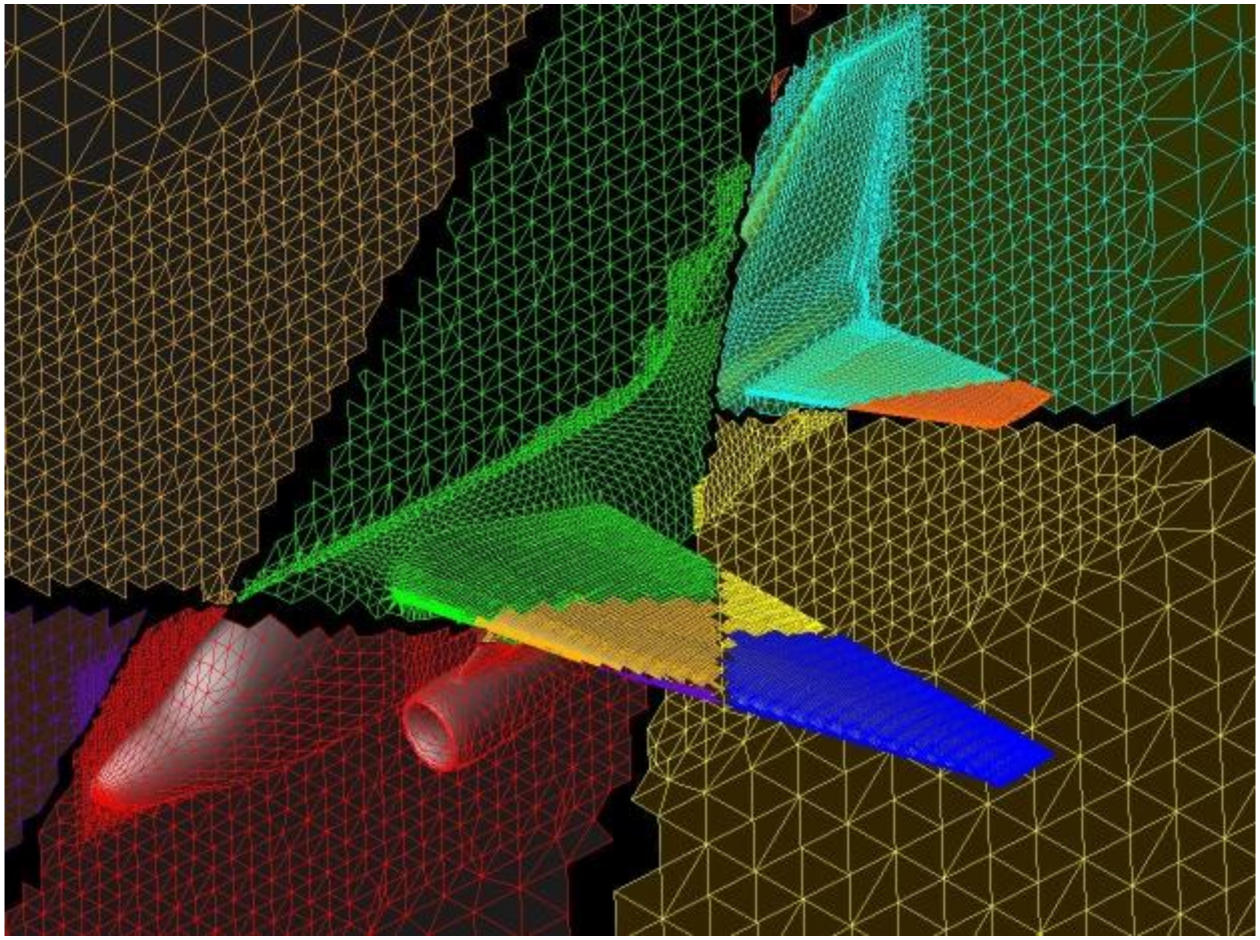


Applications

- Load balancing while minimizing communication
- Structured and unstructured mesh distribution for distributed memory parallel computing (FEM, CFD, RCS, etc.)
- Sparse matrix times vector multiplication
  - Solving PDEs (above)
  - 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
- Telephone network design
  - Original application, algorithm due to Kernighan
- Sparse Gaussian Elimination
  - Used to reorder rows and columns to increase parallelism, decrease "fill-in"

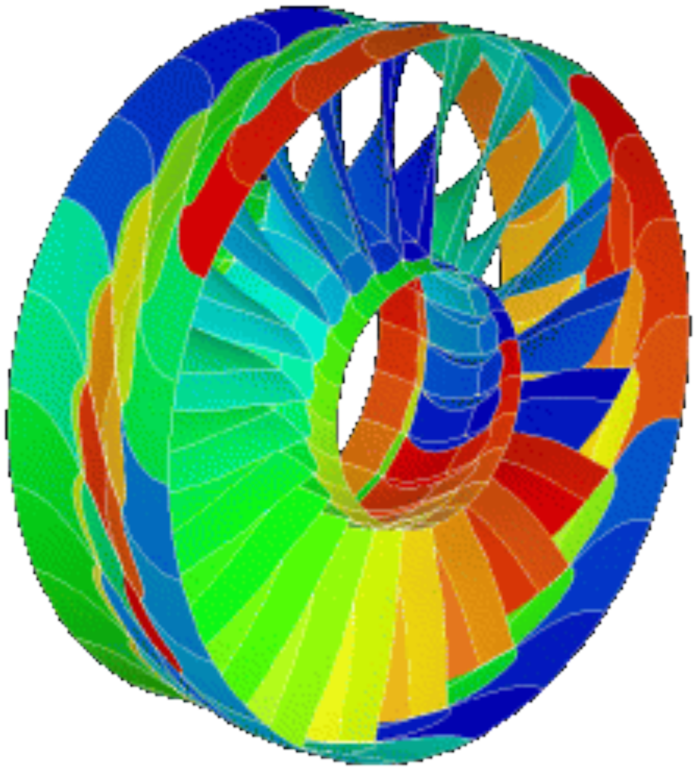
Applications-Unstructured CFD

Partitioning of an undirected nodal graph for parallel computation of the flow over an S3A aircraft using 16 processors of an IBM SP2 system (1995). Colors denote the partition number. Edge separators not shown. Solution via AIRPLANE code.



Applications- SUmb Load Balancing

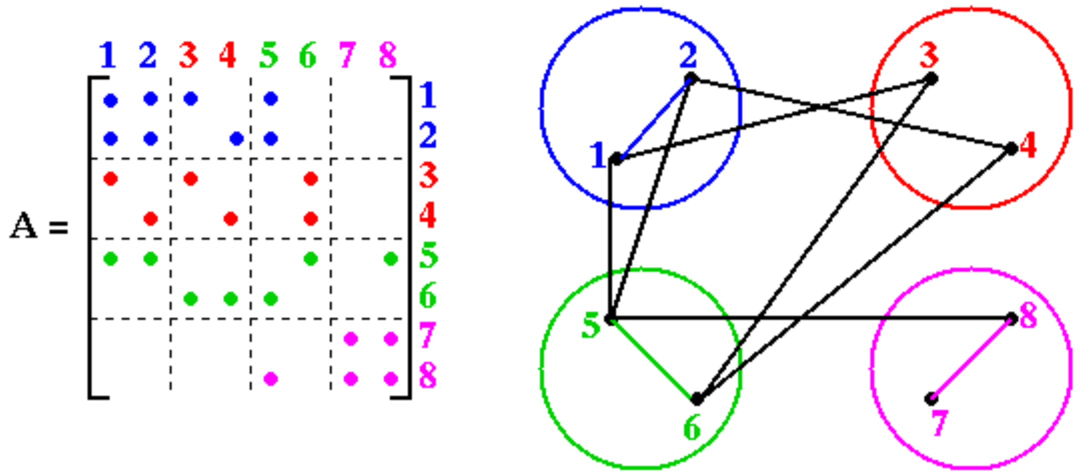
The static load balancing procedure for the multiblock-structured flow solver, SUmb, developed for the ASC project at SU, uses a graph partitioning algorithm where the original graph has nodes corresponding to mesh blocks with weights equal to the total number of cells in the block, and where the edges represent the communication patterns in the mesh; the edge weights are proportional to the surface area of the face that is being communicated



Now, that is where that silly picture comes from!!!!!

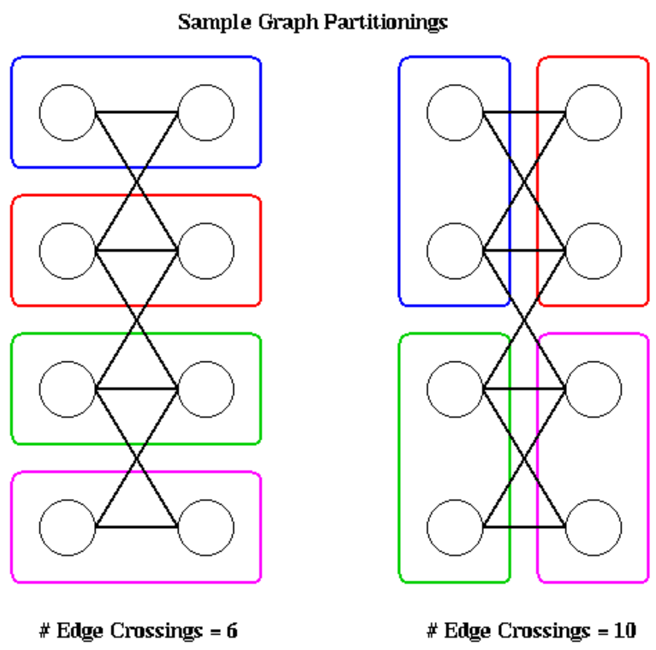
Sparse Matrix Vector Multiplication

Partitioning a Sparse Symmetric Matrix



Cost of Graph Partitioning

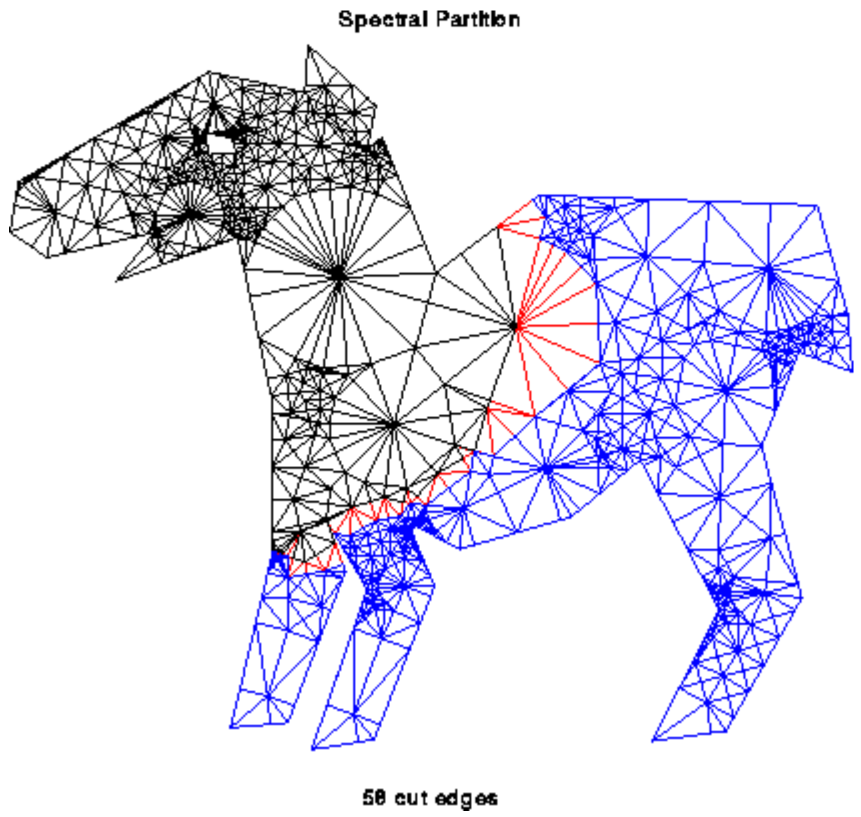
- Many possible partitionings to search:



- n choose n/2 ~ sqrt(2n/pi)*2^n bisection possibilities
- Choosing optimal partitioning is NP-complete
  - Only known exact algorithms have cost that is exponential in the number of nodes in the graph, n
- We need good heuristics-based algorithms!!

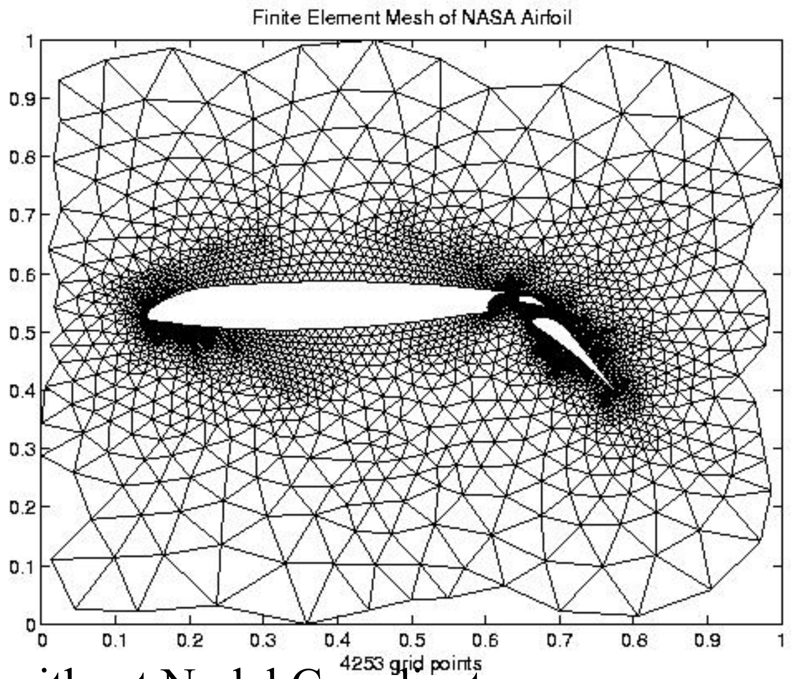
First Heuristic: Repeated Graph Bisection

- 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



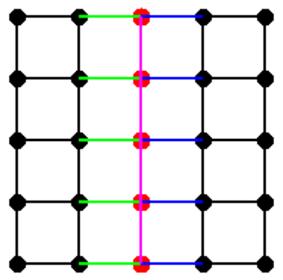
- Partitioning without Nodal Coordinates
  - Sparse matrix of Web: A(j,k) = # times keyword j appears in URL k
- Multilevel acceleration
  - Approximate problem by "coarse graph", do so recursively

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 ~equal-sized, disconnected components of N: N_1 and N_2
- Vertex Separator: N_s (subset of N) separates G if removing N_s and all incident edges leaves two ~equal-sized, disconnected components of N: N_1 and N_2
- Edge cut: Sum of the weights of all edges that form an edge separator

E_S = green edges or blue edges
N_S = red vertices

Edge Separators and Vertex Separators



- 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

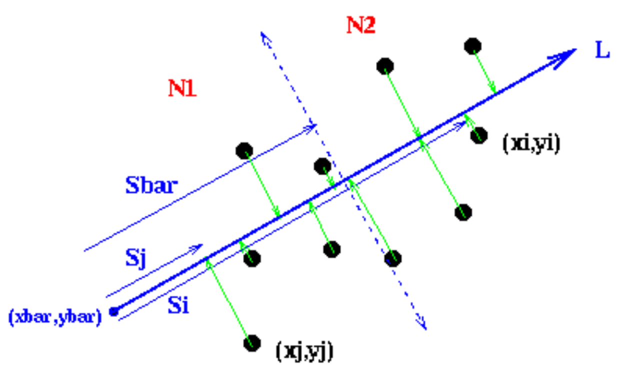
- Planar graph can be drawn in plane without edge crossings
- Ex: m x m grid of m^2 nodes: ∃ vertex separator N_s with $|N_s| = m = \text{sqrt}(|N|)$ (see last slide for m=5)
- *Theorem* (Tarjan, Lipton, 1979): If G is planar, ∃ N_s such that
  - $N = N_1 \cup N_s \cup N_2$ is a partition,
  - $|N_1| <= 2/3\ |N|$ and $|N_2| <= 2/3\ |N|$
  - $|N_s| <= \text{sqrt}(8 * |N|)$
- 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 nj = (xj,yj), compute coordinate
S_j = -b*(x_j-xbar) + a*(y_j-ybar) along L
3) Let Sbar = median(S_1,…,S_n)
4) Let nodes with S_j < Sbar be in N_1, rest in N_2

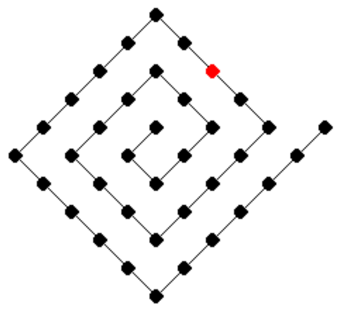
Inertial Partitioning in 2D



- Remains to choose L

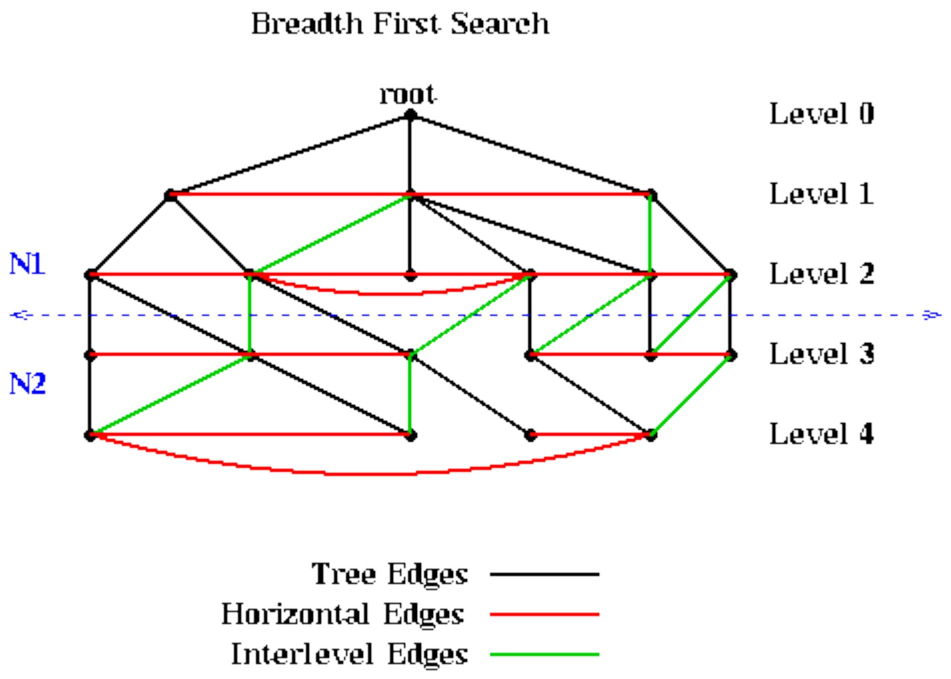
Partitioning with Nodal Coordinates - Summary

- Other algorithms and variations are available (random spheres, etc.)
- 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:

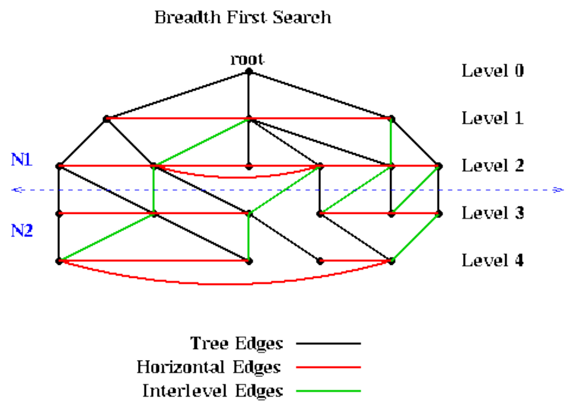


Partitioning without Nodal Coordinates- Breadth First Search (BFS)

- 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



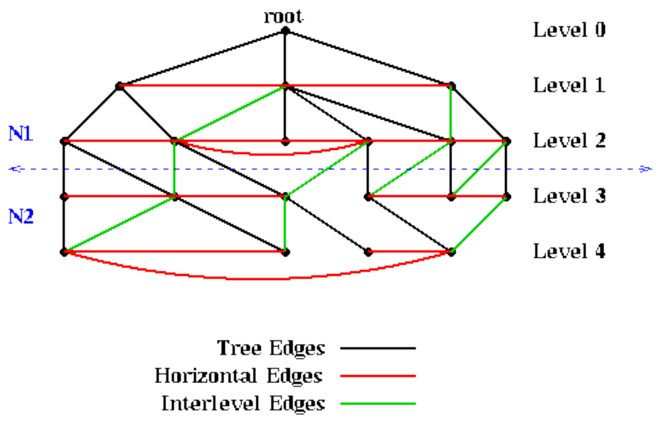
- Queue (First In First Out, or FIFO)
  - Enqueue(x,Q) adds x to back of Q
  - x = Dequeue(Q) removes x from front of Q
- Compute Tree $T(N_T,E_T)$

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

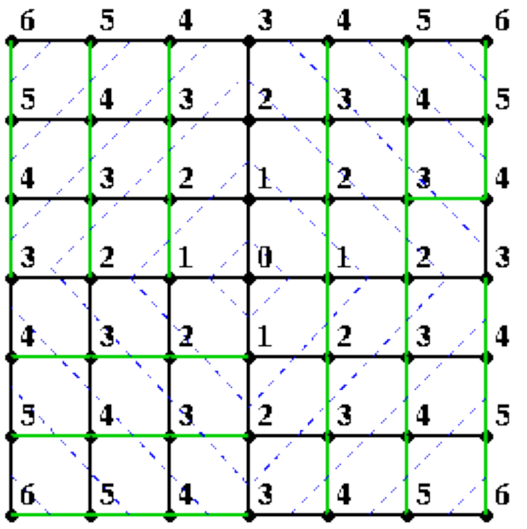
Partitioning via Breadth First Search

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

Breadth First Search



Breadth First Seach 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, better version has cost = $O(|E| \log |E|)$
  - Fiduccia/Mattheyses (1982), cost = $O(|E|)$, much better, but more complicated (it uses the appropriate data structures)
- Given $G = (N,E,W_E)$ and a partitioning $N = A \text{ U } B$, where $|A| = |B|$
  - $T = \text{cost}(A,B)$ = edge cut of A and B partitions
  - Find subsets X of A and Y of B with $|X| = |Y|$
  - Swapping X and Y should decrease cost:
    - $\text{newA} = (A - X) \text{ U } Y$ and $\text{newB} = (B - Y) \text{ U } X$
    - $\text{newT} = \text{cost}(\text{newA}, \text{newB}) < \text{cost}(A,B)$, lower edge cut
- Need to compute newT efficiently for many possible X and Y, choose smallest

Kernighan/Lin - Preliminary Definitions

- T = cost(A, B),   newT = cost(newA, newB)
- Need an efficient formula for newT; will use
  - E(a) = external cost of a in A = Σ {W(a,b) for b in B}
  - I(a) = internal cost of a in A = Σ {W(a,a') for other a' in A}
  - D(a) = cost of a in A   = E(a) - I(a)
  - E(b), I(b) and D(b) defined analogously for b in B
- Consider swapping X = {a} and Y = {b}
  - newA = (A - {a}) U {b},   newB = (B - {b}) U {a}
- newT = T - (D(a) + D(b) - 2*w(a,b)) = T - gain(a,b)
  - gain(a,b) measures improvement gotten by swapping a and b
- Update formulas
  - newD(a') = D(a') + 2*w(a',a) - 2*w(a',b)   for a' in A, a' != a
  - newD(b') = D(b') + 2*w(b',b) - 2*w(b',a)   for b' in B, b' != b

Kernighan/Lin Algorithm

```
Compute  T = cost(A,B) for initial A, B                              … cost = $O(|N|^2)$
Repeat
     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)
          …  for k = |N|/2,  ordered by the order in which we marked them
     Pick j maximizing Gain = $\Sigma_{k=1 \text{ to } j}$ gain(k)   … cost = $O(|N|)$
          … Gain is reduction in cost from swapping (a1,b1) through (aj,bj)
     If Gain > 0 then   … it is worth swapping
          Update newA = (A - { a1,…,ak }) U { b1,…,bk }               … cost = $O(|N|)$
          Update newB = (B - { b1,…,bk }) U { a1,…,ak }               … cost = $O(|N|)$
          Update T = T - Gain                                         … cost = $O(1)$
     endif
Until Gain <= 0
```

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

Comments on Kernighan/Lin Algorithm

- Most expensive line show in red
- Some gain(k) may be negative, but if later gains are large, then final Gain may be positive
  - can escape "local minima" where switching no pair helps
- How many times do we Repeat?
  - K/L tested on very small graphs ($|N|<=360$) and got convergence after 2-4 sweeps
  - For random graphs (of theoretical interest) the probability of convergence in one step appears to drop like $2^{-|N|/30}$

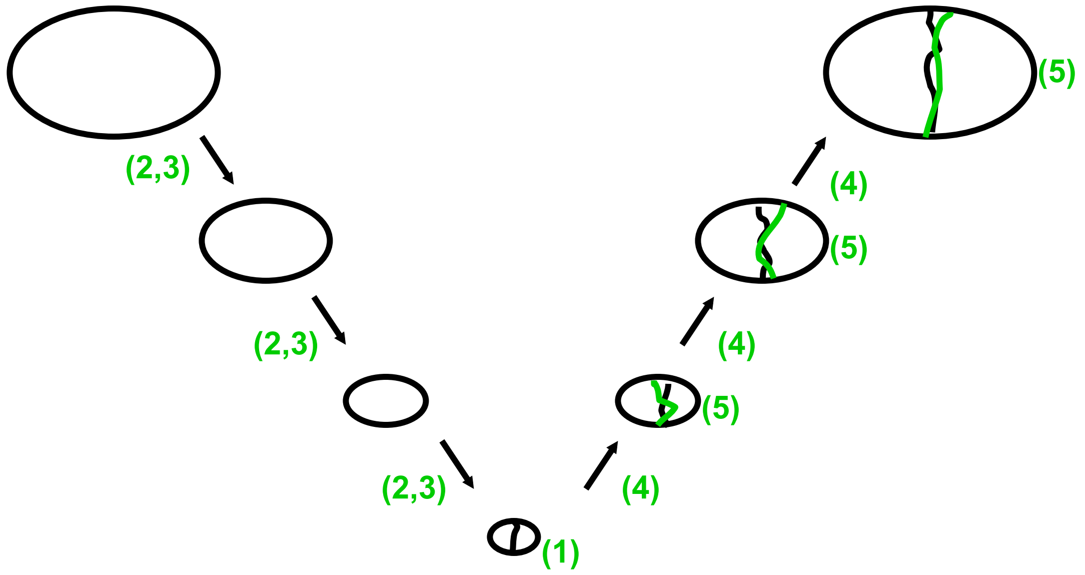
Introduction to Multilevel Partitioning

- 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
- This is identical to the multigrid procedure that is used in the solution of elliptic and hyperbolic PDEs

Multilevel Partitioning - High Level Algorithm

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

"V - cycle:"



How do we
- Coarsen?
- Expand?
- Improve?

Multilevel Kernighan-Lin

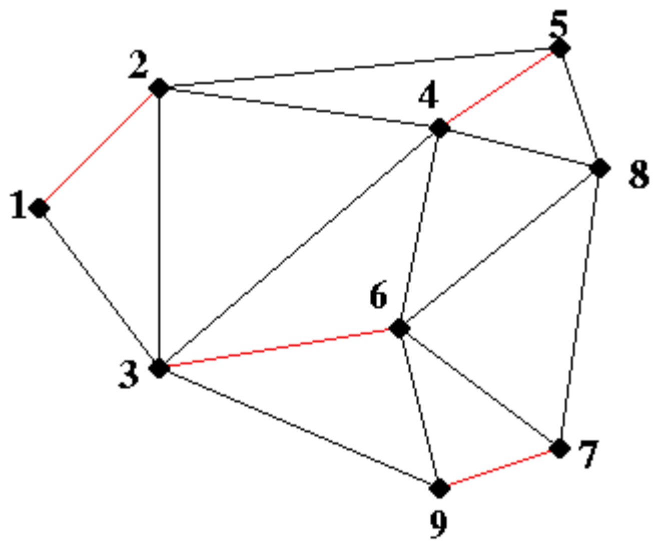
- Coarsen graph and expand partition using maximal matchings
- Improve partition using Kernighan-Lin
- This is the algorithm that is implemented in Metis (see references in web page)

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
          if there is an edge e=(i,j)  where j is also unmatched,
               add e to $E_m$
               mark i and j as matched
          endif
    endif
endfor
```

Maximal Matching - Example



Maximal matching given by red edges:

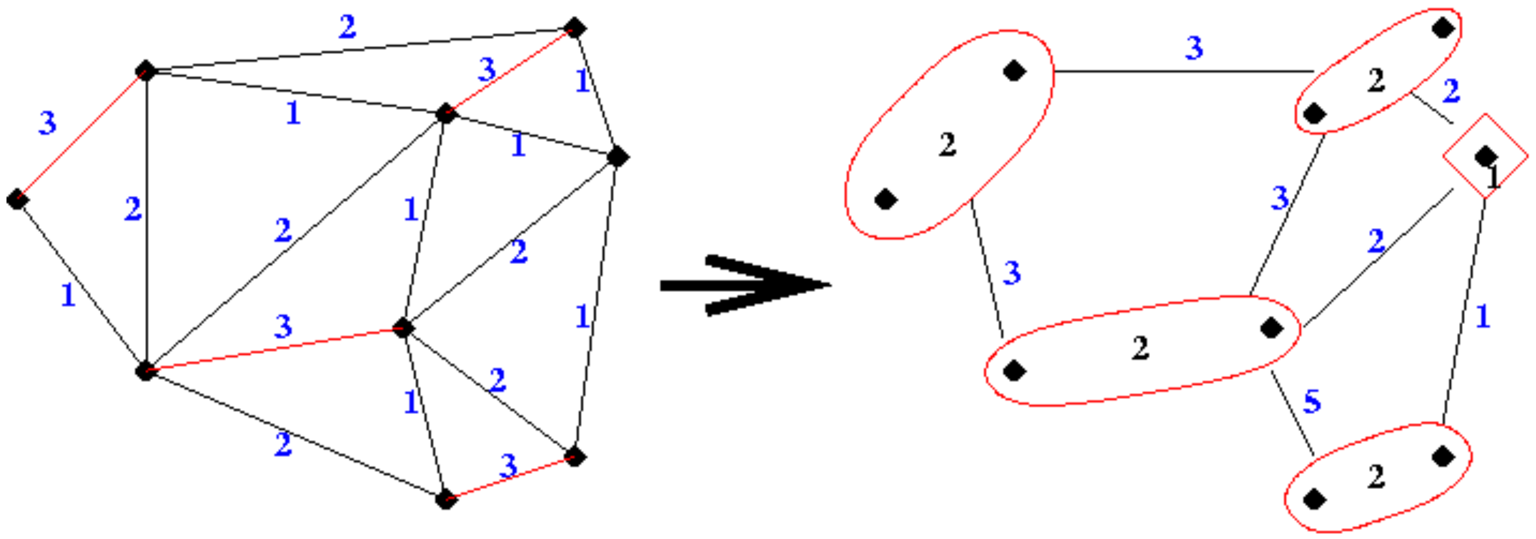
Any additional edge will connect to one of the nodes already present

Coarsening using a maximal matching

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

for all edges e=(j,k) in $E_m$
     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 $E_m$
     Put n in $N_c$      … do not change W(n)
… Now each node r in N is “inside” a unique node n(r) in $N_c$

… Connect two nodes in Nc if nodes inside them are connected in E
for all edges e=(j,k) in $E_m$
     for each other edge e’=(j,r) in E incident on j
          Put edge ee = (n(e),n(r)) in $E_c$
          W(ee) = W(e’)
     for each other edge e’=(r,k) in E incident on k
          Put edge ee = (n(r),n(e)) in $E_c$
          W(ee) = W(e’)

If there are multiple edges connecting two nodes in $N_c$, collapse them,
          adding edge weights
```

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

Example of Coarsening

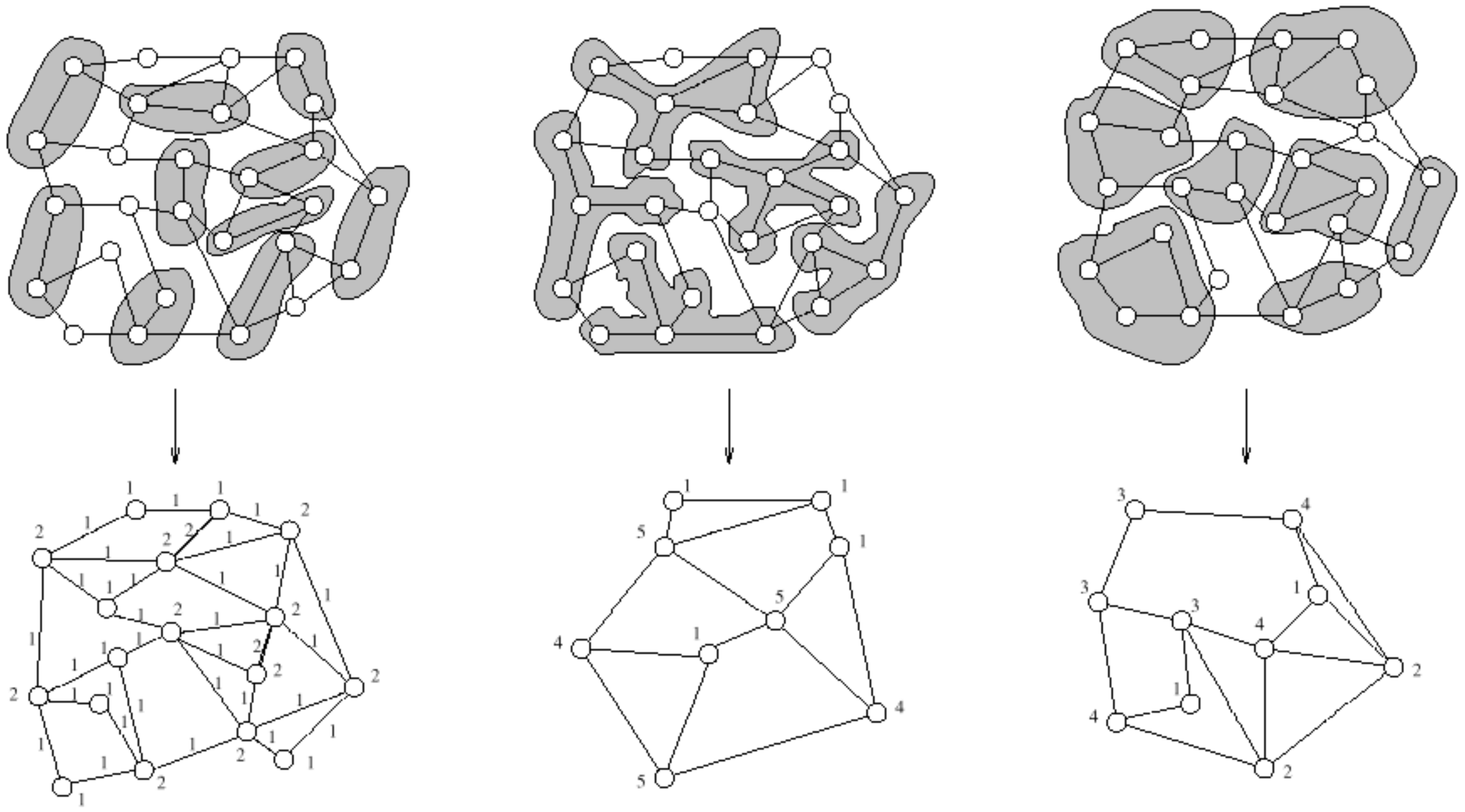
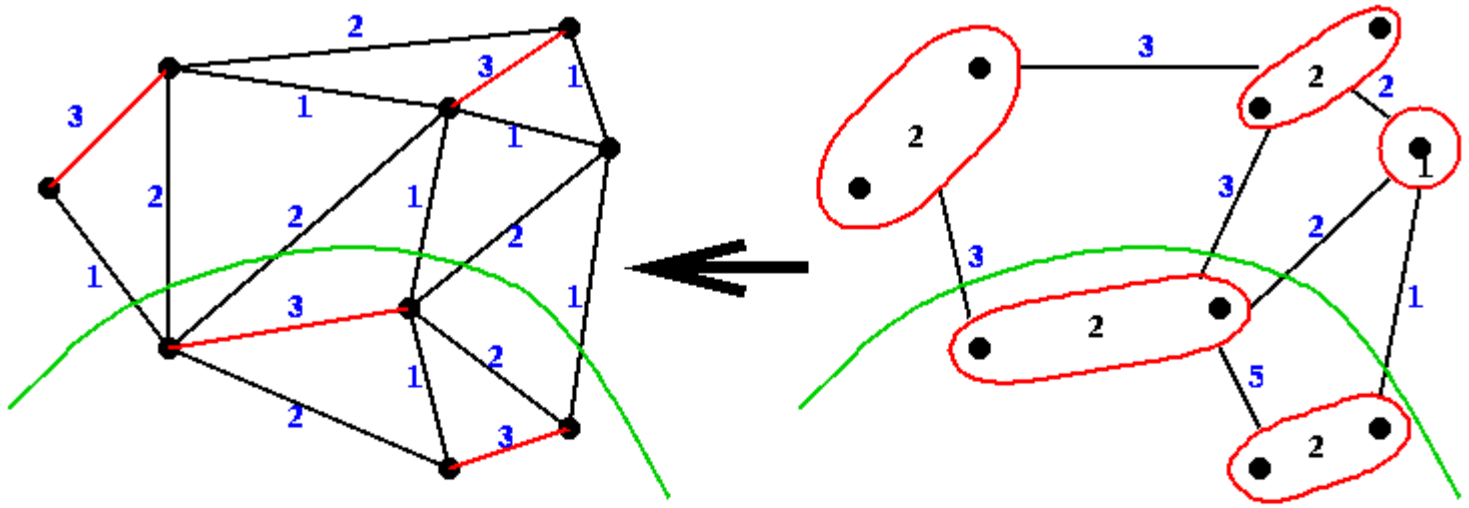


Figure 2: Different ways to coarsen a graph.

Expanding a partition of G_c to a partition of G

Converting a coarse partition to a fine partition



Partition shown in green

Available Implementations

- Multilevel Kernighan/Lin
  - METIS (www.cs.umn.edu/~metis)
  - ParMETIS - parallel version
- Multilevel Spectral Bisection
  - S. Barnard and H. Simon, "A fast multilevel implementation of recursive spectral bisection …", Proc. 6th SIAM Conf. On Parallel Processing, 1993
  - Chaco (www.cs.sandia.gov/CRF/papers_chaco.html)
- Hybrids possible
  - Ex: Using Kernighan/Lin to improve a partition from spectral bisection

Available Implementations

- Multilevel Kernighan/Lin
  - Demonstrated in experience to be the most efficient algorithm available.
- Multilevel Spectral Bisection
  - Gives good partitions but cost is higher than multilevel K/L
- Hybrids possible
  - For example: Using Kernighan/Lin to improve a partition from spectral bisection