Master Course – High-Performance Computing

Parallel Graph-Partitioning on HPC Architectures

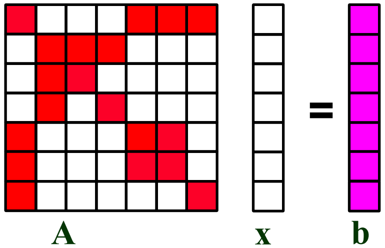
Olaf Schenk
Institute of Computational Science
USI Lugano, Switzerland
November 16, 2021

Content

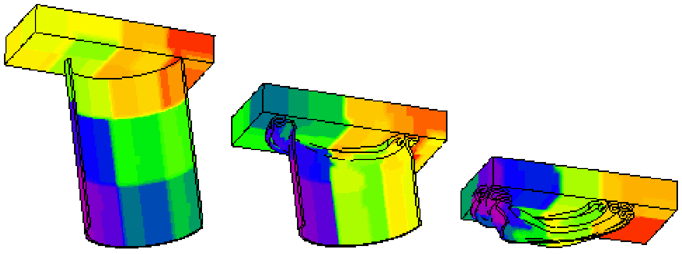
- Motivation for graph partitioning
- Overview of heuristics
- Partitioning with nodal coordinates
  - Ex: In finite element models, node at point in (x, y, z) space
    Recursive Coordinate Bisection
    Inertial Partitioning
- Partitioning without nodal coordinates
  - Ex: In model of WWW, nodes are web pages
    Fiduccia-Matteyes
    Spectral Methods
- Multilevel acceleration
  - **BIG IDEA**, appears often in scientific computing
- Available implementations
- Beyond Graph Partitioning: Hypergraphs

Partitioning and Load Balancing

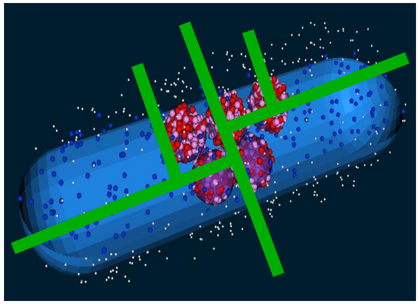
- **Goal:** assign data to processors to
  - minimize parallel application runtime
  - maximize utilization of computing resources
- **Metrics:**
  - minimize processor idle time (balance workloads)
  - keep inter-processor communication costs low
- Impacts performance of a wide range of simulations



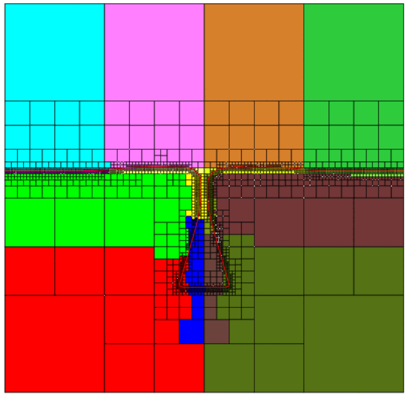
Linear solvers & preconditioners



Contact detection



Particle simulations



Adaptive mesh refinement

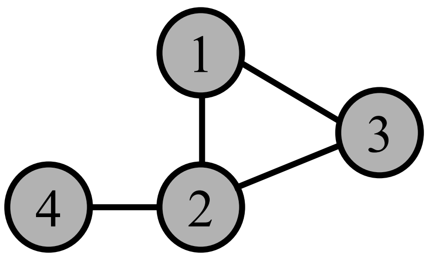
Graph Partitioning

- Work-horse of load-balancing community.
- **Highly successful model for PDE problems.**
- Model problem as a graph:
  - vertices = work associated with data (computation)
  - edges = relationships between data/computation (communication)
- Goal: Evenly distribute vertex weight while minimizing weight of cut edges.

- **Excellent software available**
  - Serial: METIS (U. Minn.), Scotch (U. Bordeaux)
  - Parallel: ParMETIS (U. Minn.), PT-Sotch (U. Bordeaux)

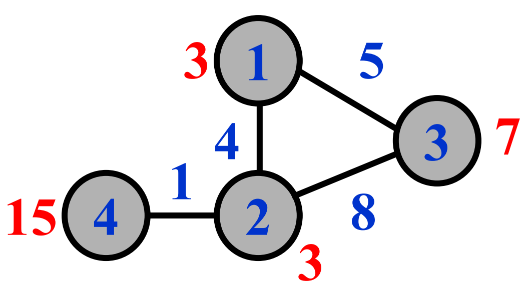
Definition of Graph

- Given a graph $G = (V, E)$ with
  - Vertices $V = \{ v_i \mid i=1,\ldots,n\}$
  - Edges $E = \{ e_{ij} \mid v_i \text{ and } v_j \text{ are connected}\}$



$V = \{1, 2, 3, 4\}$ $E = \{ (1,2), (1,3), (2,3), (2,4)\}$

- A weighted graph $G = (V, E, W_v, W_e)$ has **node weights** and **edge weights**
  - $W_v = \{ w_v(v_i) \mid v_i \epsilon V\}$ („weight of vertices“).
  - $W_e = \{ w_e(e_{ij}) \mid e_{ij} \epsilon E\}$ („weight of edges“).

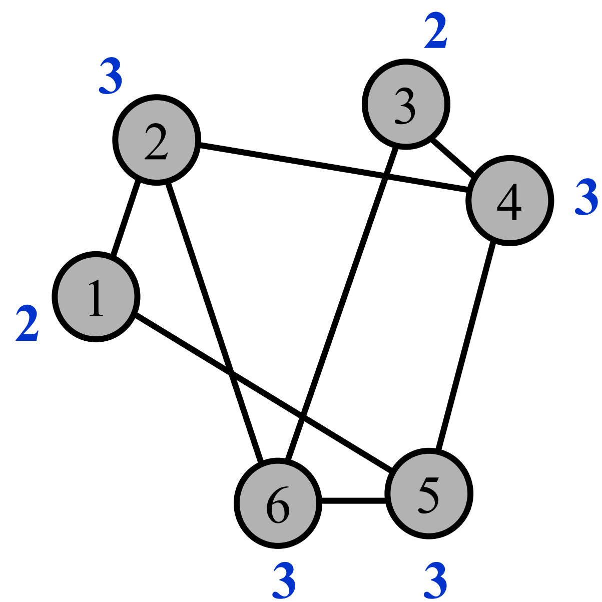


$W_v = \{ 3, 3, 7, 15\}$, $W_e = \{ 4, 5, 8, 1\}$

Examples for Graphs

- Symmetric sparse matrix and Graph G_A

$$A = \begin{array}{c|cccccc|} & 1 & 2 & 3 & 4 & 5 & 6 \\ \hline 1 & 1 & 1 & & & 7 & \\ 2 & 8 & 6 & & 1 & & 1 \\ 3 & & & 3 & 1 & & 1 \\ 4 & & 5 & 1 & 2 & 1 & \\ 5 & 2 & & & 1 & 5 & 1 \\ 6 & & 1 & 5 & & 1 & 1 \\ \hline \end{array}$$



- $G_A = (V, E, W_V, W_E)$; $V = \{1, 2, 3, 4, 5, 6\}$,

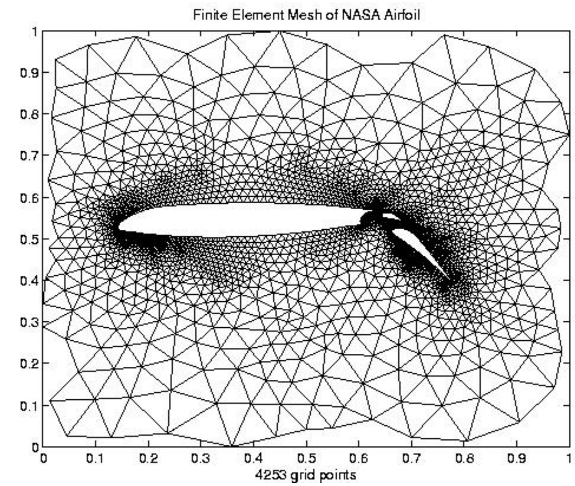
  $E = \{ (1,2), (1,5), (2,4), (2,6), (3,4), (3,6), (4,5), (5,6) \}$

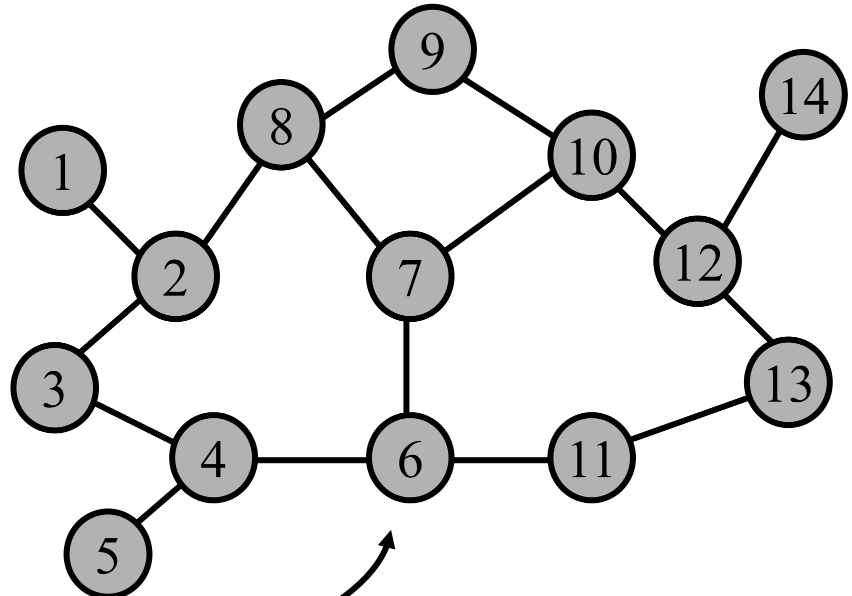
  $W_v = \{2, 3, 2, 3, 3, 3\}$ e.g. numbers of nonzeros in each row

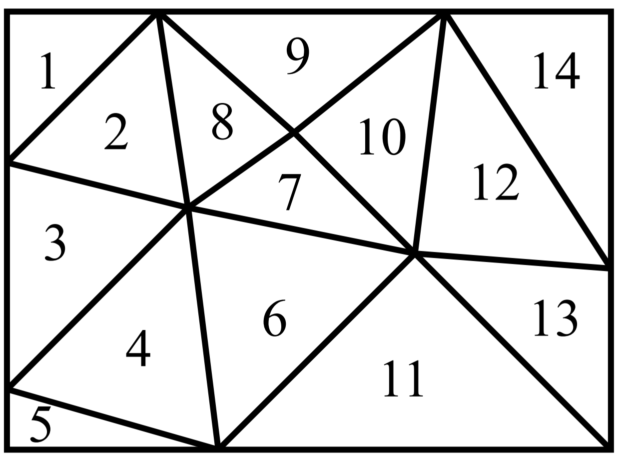
  $W_e = \{1, 1, 1, 1, 1, 1, 1, 1\}$

Examples for Graphs

- Finite-Element Simulations







Finite-Element Mesh

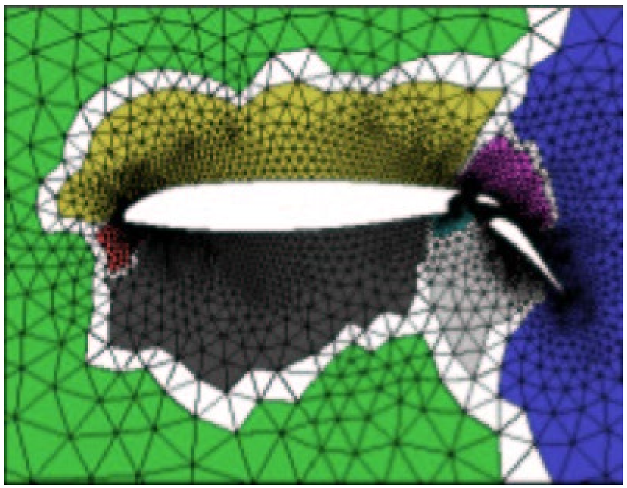
$G_{FE} = (V, E), V = \{1, \ldots, 14\}$

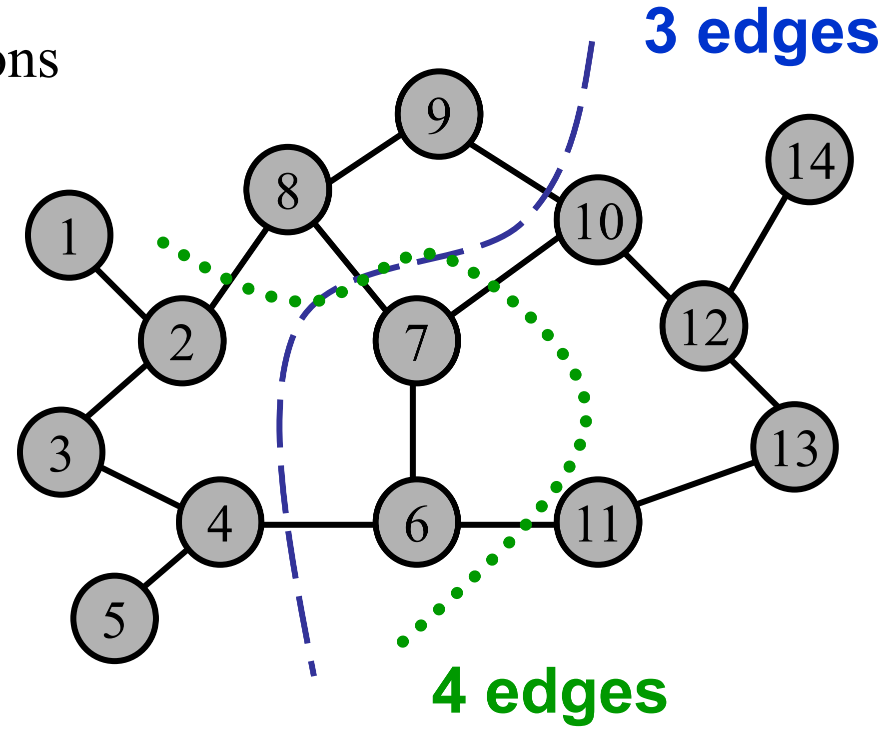
$E = \{(1,2), \ldots, (12,14)\}$

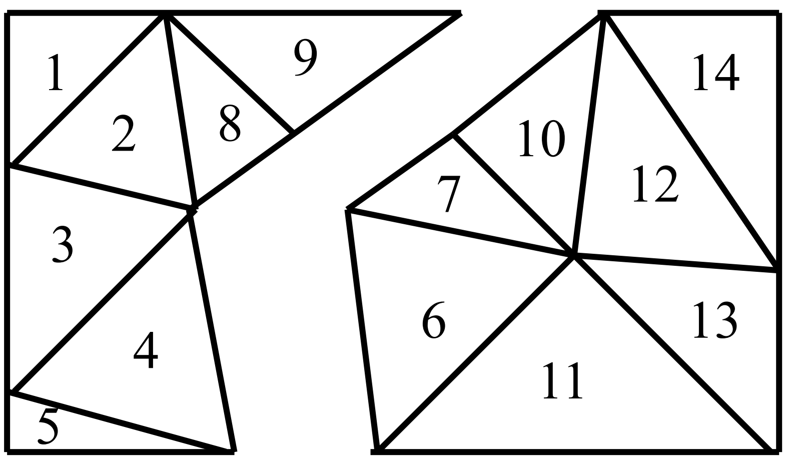
$W_e \equiv 1, \; W_v \equiv 1$

Examples for Graph Partitioning

- Parallel Finite-Element Simulations







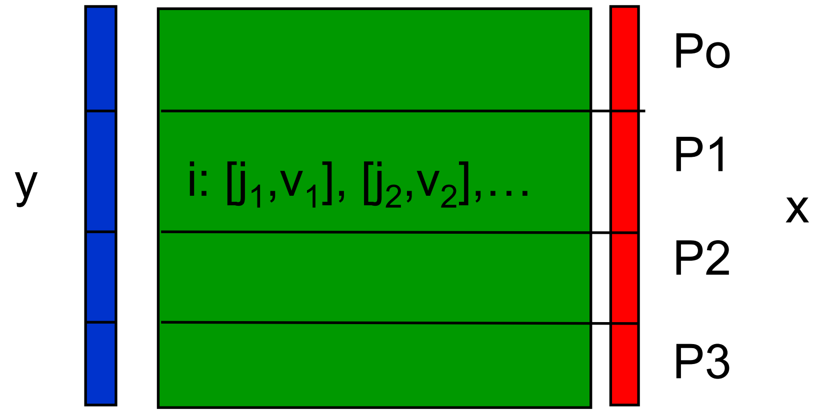
Proc. 1 Proc. 2

A good partitioning G_{FE} results in
- equal #elements/processor („load“ and „storage balancing“).
- Minimal #edges between P1 and P2 (minimal communication volume).

Examples for Graph Partitioning

- **Sparse Matrix-Vector Multiplication:**

  y = A*x



- Which processor stores the y[i], x[i] and A[i, j] ?
- Which processor computes

  y[i] = ∑ (j from 1 to n) A[i, j] * x[j]

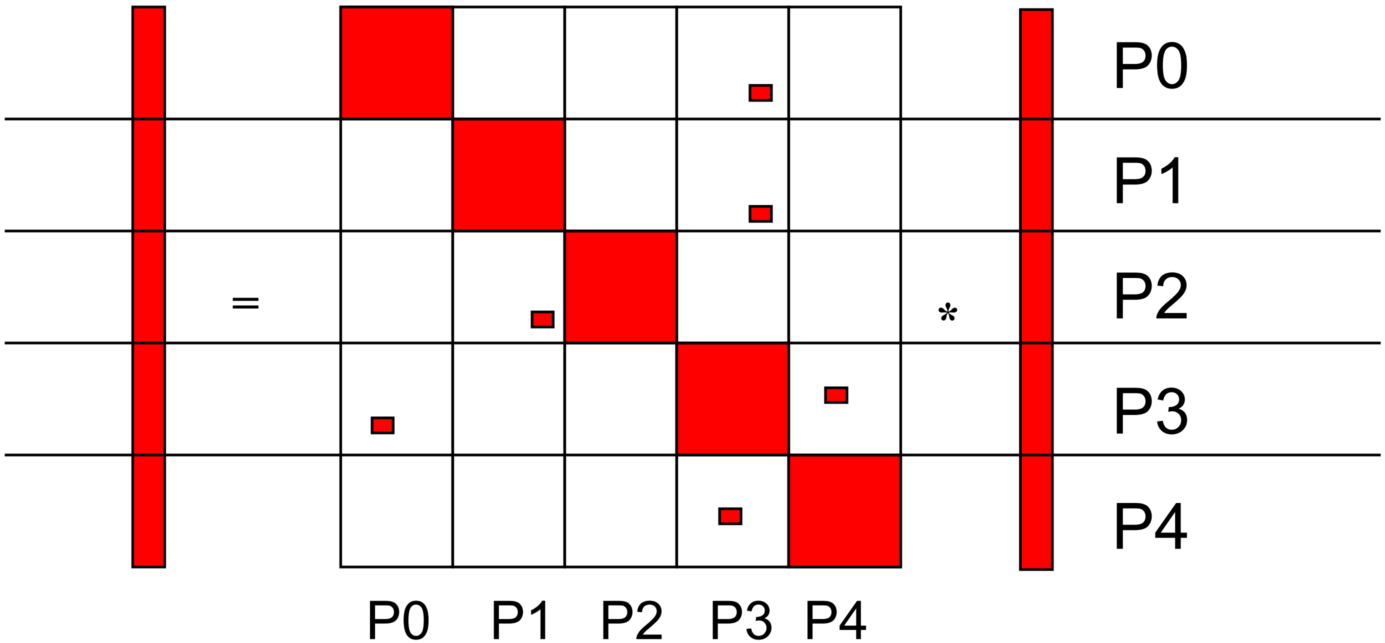
  communication !

  = (row i of A) * x **… sparse scalar product**

- Partitioning
  - Partition index set {1,…, n} = $V_1 + V_2 + \ldots + V_p$.
  - For each i in V_k : Proc. k stores y[i], x[i] and row i of A
  - For each i in V_k : Proc. k computes y[i] = (row i of A) * x
  - "Rule": Proc. k computes own index set y[i]'s.

Examples for Graph Partitioning

- Perfect matrix structure for parallelization: block-diagonal
  - p (# processors) blocks which can be computed locally
  - minimize non-zero elements in the off-diagonal block elements
  - Question: Can we permute rows and columns in such a way that the permuted matrix has a block diagonal structure

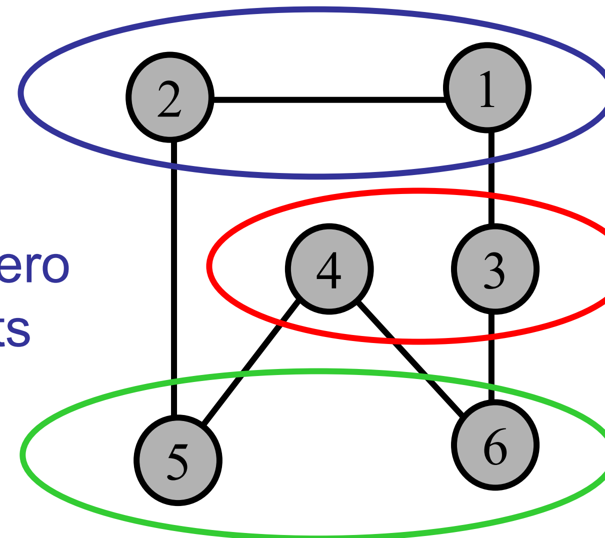


Examples for Graph Partitioning

- Parallel matrix-vector multiplication y = A*x

$$
A = \begin{array}{c|cccccc|}
 & 1 & 2 & 3 & 4 & 5 & 6 \\
\hline
1 & 1 & 1 & 1 & & & \\
2 & 1 & 2 & & & 1 & \\
3 & 1 & & 3 & & & 1 \\
4 & & & & 4 & 1 & 1 \\
5 & & 1 & & 1 & 5 & \\
6 & & & 1 & 1 & & 6 \\
\hline
\end{array}
$$

5 non-zero elements



5 edges

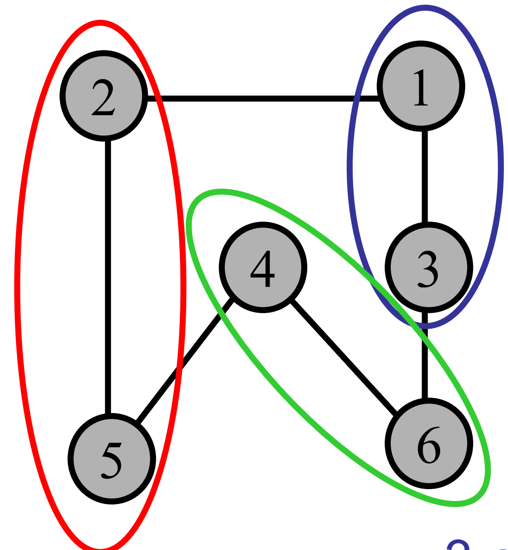
- Good graph partitioning $G_A = (V, E, W_v)$ results in
  - Equal number of vertices on each processor („load“ and „storage balancing“)
  - Minimal number of edges between partitioning (minimal communication)
- Permute rows and columns according to the partitioning

Examples for Graph Partitioning

- Better partitioning y’ = Py = PAP^T P*x

$$A = \begin{array}{c|cccccc|} & 1 & 3 & 2 & 5 & 4 & 6 \\ \hline 1 & 1 & 1 & 1 & & & \\ 3 & 1 & 3 & & & & 1 \\ 2 & 1 & & 2 & 1 & & \\ 5 & & & 1 & 5 & 1 & \\ 4 & & & & 1 & 4 & 1 \\ 6 & & 1 & & & 1 & 6 \\ \hline \end{array}$$

3 nonzero elemens



3 edges

- **Original system y=A*x:**

y[0] = 1*x[0] + 1*x[1] + 1*x[2]

y[1] = 1*x[0] + 2*x[1] + 1*x[4]

y[2] = 1*x[0] + 3*x[2] + 1*x[5]

y[3] = 4*x[3] + 1*x[4] + 1*x[5]

- **Permuted System Py = PAP^T P*x**

y[0] = 1*x[0] + 1*x[2] + 1*x[1]

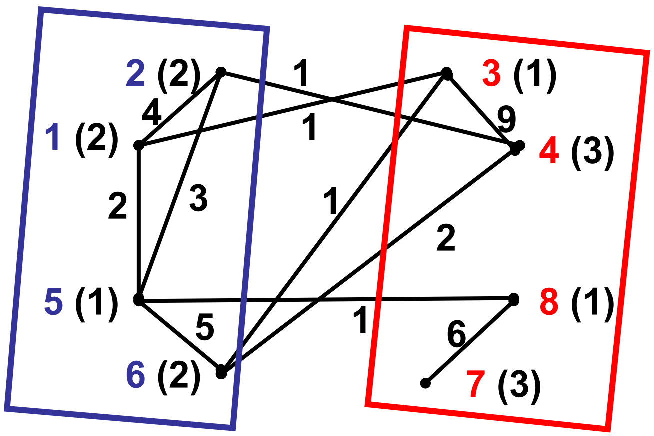
y[2] = 1*x[0] + 3*x[1] + 1*x[5]

y[1] = 1*x[0] + 2*x[2] + 1*x[4]

y[4] = 1*x[1] + 5*x[4] + 1*x[3]

Definition of Graph Partitioning: Bisection

- Given a graph $G = (V, E, W_V, W_E)$
  - V = nodes (or vertices)
  - E = edges



- Choose a partition $V = V_1 \cup V_2$ such that:

The sum of the node in each V_j is "about the same"

$$V = V_1 \cup V_2\ ,\ \ V_1 \cap V_2 = \emptyset, \quad |V_1| = |V_2|$$

The sum of edge connecting pairs V_1 and V_2 is minimized

$$\min |\{e_{IJ} \epsilon E \mid v_i \ \epsilon V_1 \text{ und } v_j \ \epsilon\ V_2\}|$$

Definition of Graph Partitioning: k-Partitioning

- $G = (V, E)$ Graph with weights W_v and W_e

  Find a k-Partition $(V_1, V_2, \ldots, V_k)$ of V with

  (1) $V = \bigcup_{i=1}^{k} V_i$ und $V_i \cap V_j = \emptyset$ (for all $i \neq j$)

  (2) $\sum_{v(i) \in V_j} w_v(v_i)$ of equal size for all $j \in \{1, \ldots, k\}$

  (3) $\sum_{\substack{e_{ij} \in E \text{ und} \\ v_i \in V_p, v_j \in V_q \text{ und } p \neq q}} w_e(e_{ij})$ minimizes over all partitionings of V.

- (1) and (2) is important for „load balance“
- (3) minimizes „edge cut“

Expected # cuts for 64-way partitioning

Question: Expected # edge cuts for 64-way partition of 2D mesh of n nodes?

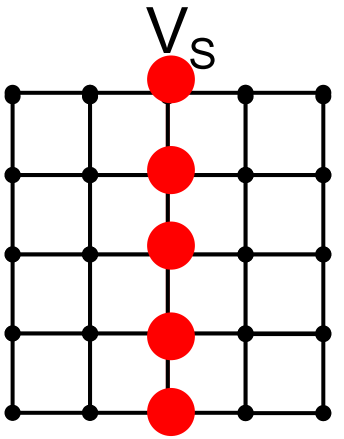
$$n^{1/2} + 2*(n/2)^{1/2} + 4*(n/4)^{1/2} + \ldots + 32*(n/32)^{1/2} \sim \mathbf{17 * n^{1/2}}$$

In the printed version, the solutions can be found in the appendix

Question: Expected # edge cuts for 64-way partition of 3D mesh of n nodes?

$$n^{2/3} + 2*(n/2)^{2/3} + 4*(n/4)^{2/3} + \ldots + 32*(n/32)^{2/3} \sim \mathbf{11.5 * n^{2/3}}$$

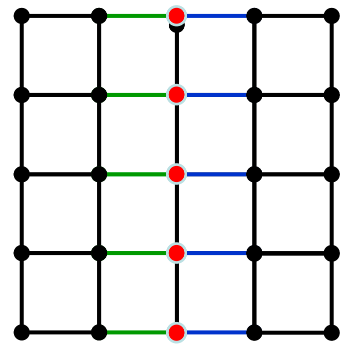
In the printed version, the solutions can be found in the appendix



Heuristics — Edge Separators vs. Vertex Separators

- **Edge Separator**: E_s (subset of E) separates G if removing E_s from E leaves two ~equal-sized, disconnected components of V: V_1 and V_2
- **Vertex Separator:** V_s (subset of V) separates G if removing V_s and all incident edges leaves two ~equal-sized, disconnected components of V: V_1 and V_2

G = (V, E) Vertices V and Edges E
E_s = green edges or blue edges
V_s = red vertices



- Making a V_s from an E_s: pick one endpoint of each edge in E_s

Question: Given E_s - can we define an upper bound or V_s?

$$|\mathbf{V_s}| <= |\mathbf{E_s}|$$

- Making an E_s from a V_s: pick all edges incident on V_s

Question: Given V_s - can be define an upper bound or E_s?

$|E_s| <= d * |V_s|$ where d is the maximum degree of the graph

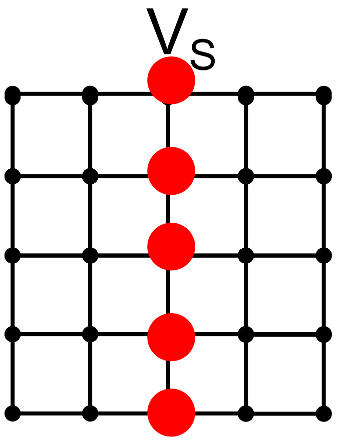
Heuristics — Graph Partitioning with Vertex Separators

Graph $G = (V, E)$ compute partitioning with V_1, V_2, V_S such as

(1) $V = V_1 \cup V_2 \cup V_s$, $V_1 \cap V_2 \cap V_s = \emptyset$,

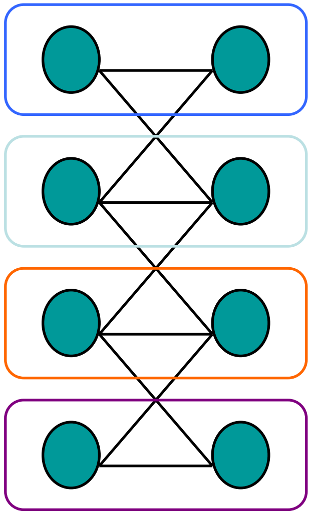
(2) $|V_1| = |V_2|$

(3) $|V_S|$ small (Vertex separator)

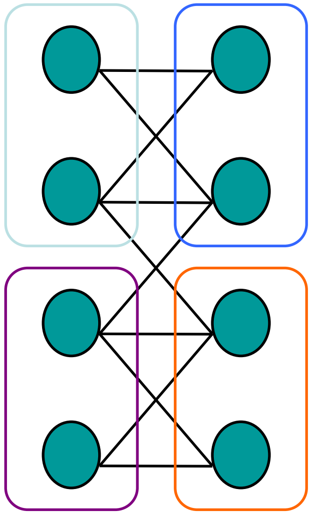


Heuristics — Cost of Graph Partitioning

- Many possible partitionings to search



„edge cut" = 6



„edge cut" = 10

- Choosing optimal partitioning is **NP-complete**
  - Only known exact algorithms have cost = exponential(n)

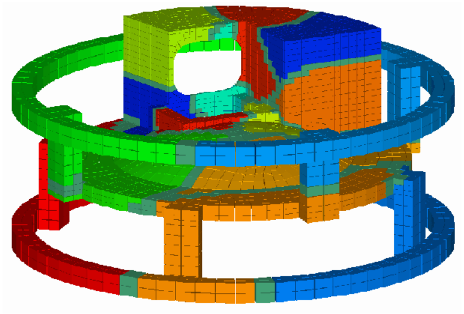
- We need **good** **and** **fast** heuristics

Heuristics — Overview of Bisection Algorithms

- Partitioning <u>with</u> nodal coordinates — e.g. each node has x,y,z coordinates → partition space

  <u>Algorithms</u>: **Recursive Coordinate Bisection**

  Inertial Partitioning



- Partitioning <u>without</u> Nodal Coordinates — e.g. indexing of web documents A(j,k) = # times keyword j appears in URL k

  <u>Algorithms:</u> **Fiduccia-Matteyes**

  Spectral Methods

- Multilevel acceleration
  - Approximate problem by "coarse graph," do so recursively

Nodal Coordinates: How Well Can We Do?

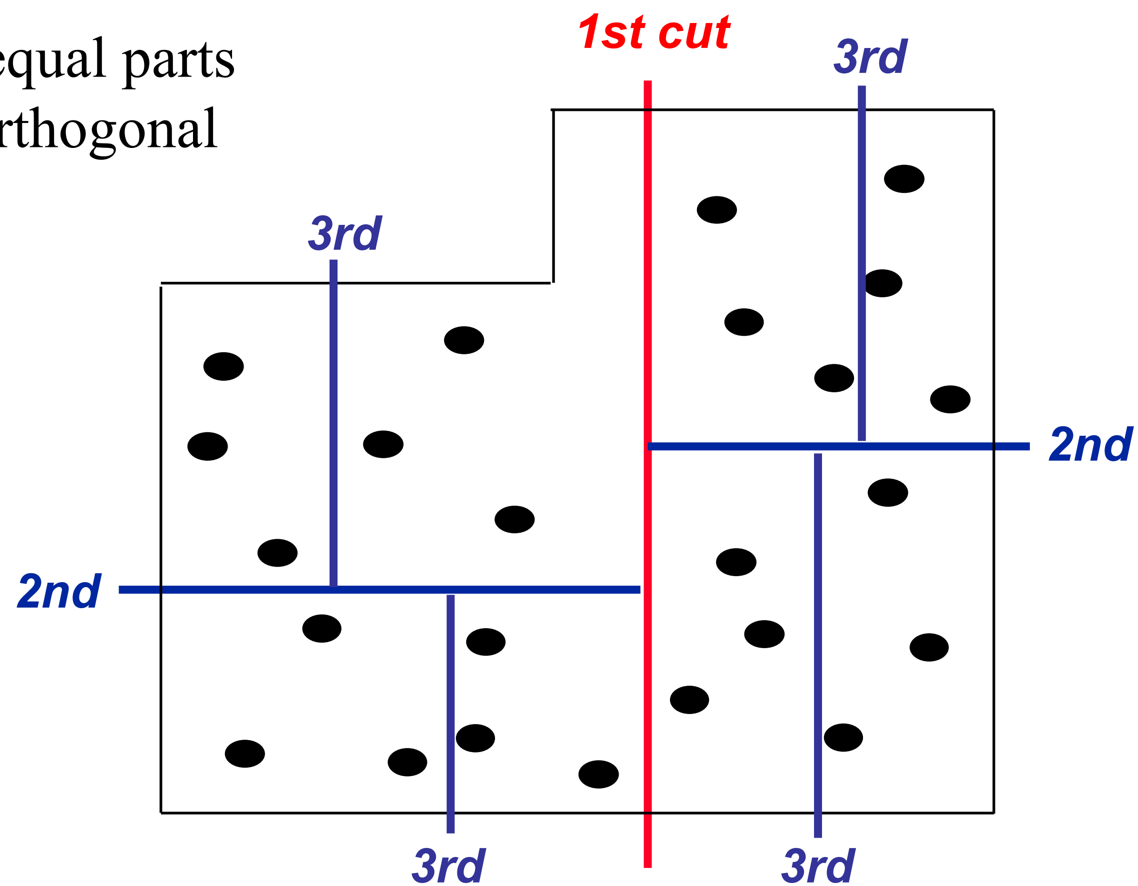
- **A planar graph** can be drawn in plane without edge crossings
- Example: m x m grid of m^2 nodes: there exists a vertex separator N_s with $|N_s| = m = |N|^{1/2}$ (see earlier slide for m=5)
- *Theorem* (Tarjan, Lipton, 1979): If G is planar, there exists N_s such that
  - **$N = N_1 \cup N_s \cup N_2$ is a partition,**
  - **$|N_1| <= 2/3\ |N|$ and $|N_2| <= 2/3\ |N|$**
  - **$|N_s| <= (8 * |N|)^{1/2}$**
- Theorem motivates intuition of following algorithms

Content

- Motivation for graph partitioning
- Overview of heuristics
- **Partitioning with nodal coordinates**
  - **Ex: In finite element models, node at point in (x, y, z) space**
    Recursive Coordinate Bisection
    Inertial Partitioning
- Partitioning without nodal coordinates
  - Ex: model of WWW, nodes are web pages
    Fiduccia-Matteyes
    Spectral Methods
- Multilevel Acceleration
- Comparison of Methods and Applications

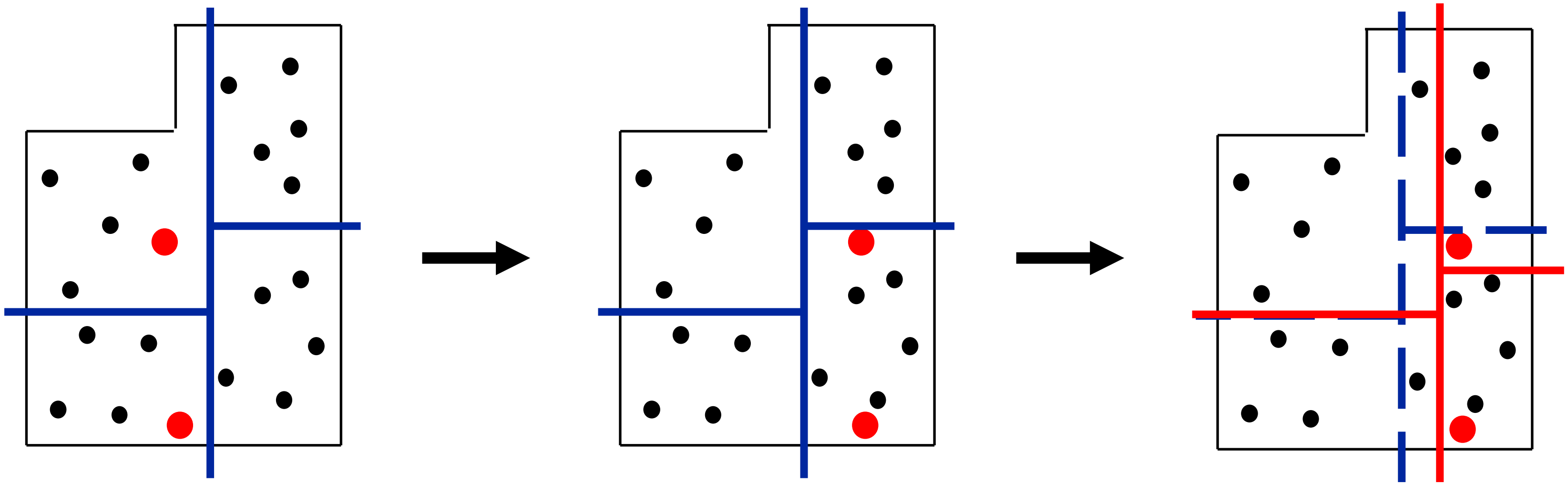
Nodal Coordinates — Recursive Coordinate Bisection (RCB)

- Developed by Berger & Bokhari (1987)
  - Independently discovered by others.
- Idea:
  - Divide work into two equal parts using a cutting plane orthogonal to a coordinate axis.
  - Recursively cut the resulting subdomains.



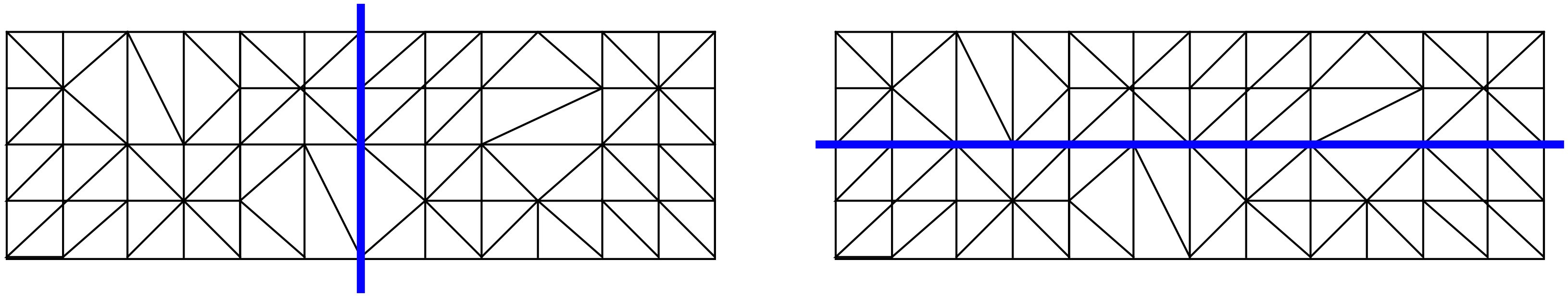
Nodal Coordinates — RCB Advantages

- Conceptually simple; fast and inexpensive.
- Regular subdomains.
  - Can be used for structured or unstructured applications.
- Effective when connectivity info is not available.
- **Incremental, but no control** of communication costs.

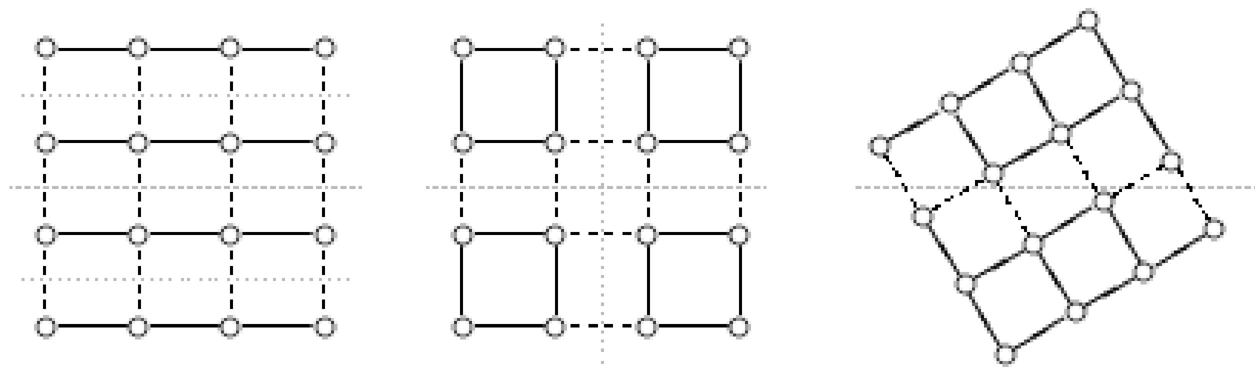


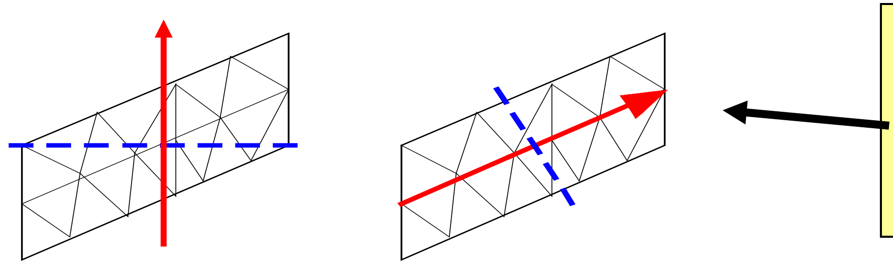
Nodal Coordinates — Coordinate Bisection

- Partition the domain along hyperplanes with node coordinates



- Change the coordinate systems

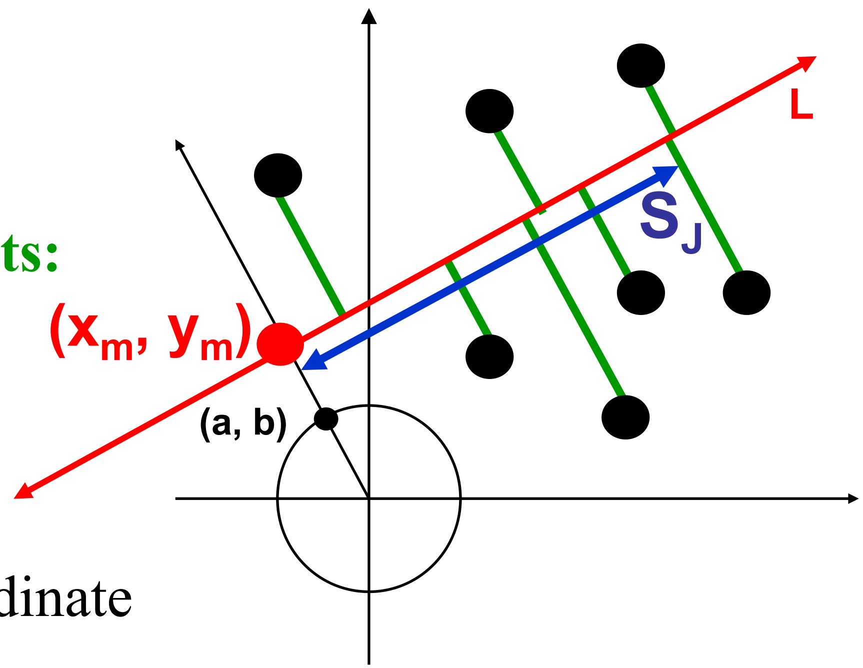




Good choice of coordinate system leads to inertial bisection

Nodal Coordinates — Inertial Partitioning

- Choose a line L, and then choose a line H orthogonal to it, with half the nodes on either side



(1) Center of mass: x_m, y_m

(2) Choose a line L through the points:

L given by $a*(x-x_m)+b*(y-y_m)=0$ with $a^2+b^2=1$; (a, b) is a unit vector orthogonal to L

(3) Project each point to the line

For each $n_j = (x_j, y_j)$ compute coordinate $S_j = -b*(x_j-x_m) + a*(y_j-y_m)$ along L

(4) Compute the median

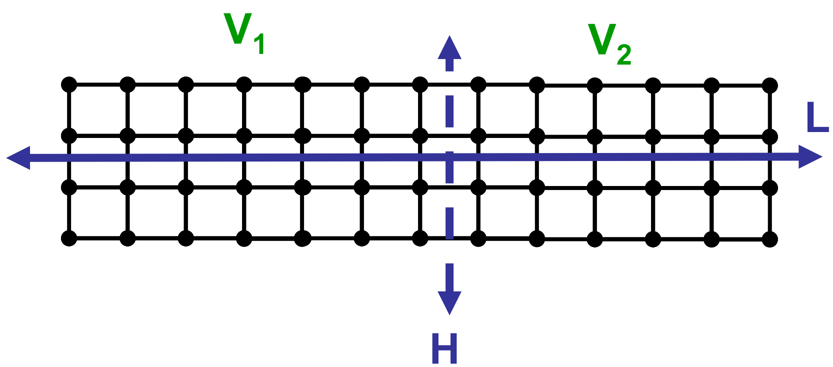
– Let $S_k = \text{median}(S_1,\ldots, S_n)$

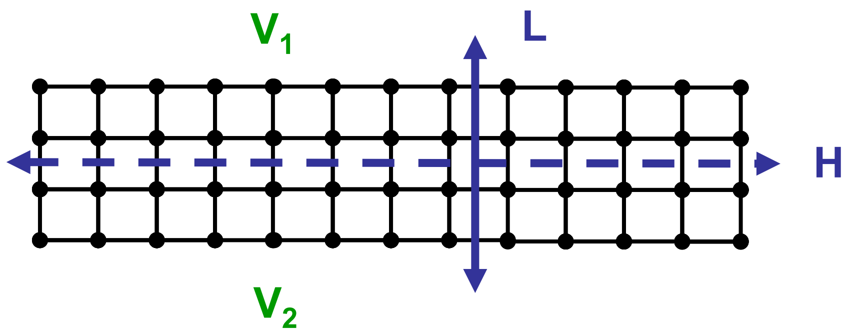
(5) Use median to partition the nodes

– Let nodes with $S_j < S_m$ be in V_1, rest in V_2

Nodal Coordinates — Inertial Partitioning, Choosing L

- Clearly prefer L on left below

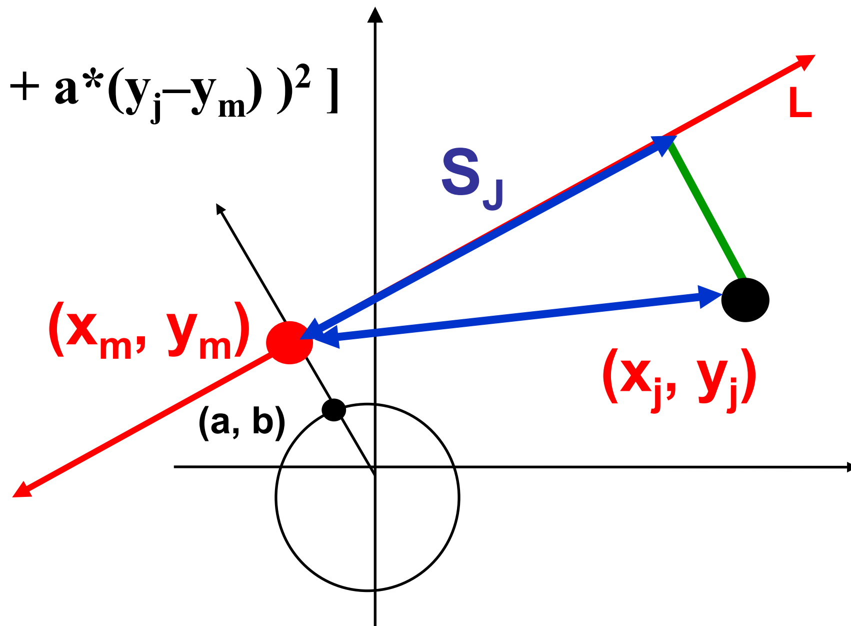




- Mathematically, choose L to be a **total least squares fit of the nodes**
  - Minimize sum of squares of distances to L (green lines on last slide)
  - Equivalent to choosing L as axis of rotation that minimizes the moment of inertia of nodes (unit weights) - source of name

Nodal Coordinates — Inertial Partitioning, Choosing L

- $\sum_j$ (length of j-th green line)^2
  $= \sum_J \; [\, (x_j - x_m)^2 + (y_j - y_m)^2 - (-b*(x_j - x_m) + a*(y_j - y_m))^2 \,]$
  … Pythagorean Theorem



$= (1 - b^2) * \sum_j (x_j - x_m)^2 + 2*a*b* \sum_j (x_j - x_m)*(y_j - y_m) + (1-a^2) \sum_j (y_j - y_m)^2$

$= a^2 * \sum_j (x_j - x_m)^2 + 2*a*b* \sum_j (x_j - x_m)*(y_j - y_m) + b^2 \sum_j (y_j - y_m)^2$

$= a^2 * X_1 + 2*a*b* X_2 + b^2 * X_3$

$= \begin{vmatrix} a & b \end{vmatrix} * \begin{vmatrix} X1 & X2 \\ X2 & X3 \end{vmatrix} * \begin{vmatrix} a \\ b \end{vmatrix}$ = <u>minimum</u> $= \lambda$

$M\,u = \lambda\,u \leftrightarrow$ $\boxed{\text{Minimizing } \lambda \text{ ?}}$

$u^T M\,u = u^T \lambda\,u = \lambda\,u^T u = \lambda$

<u>Minimized</u> by choosing

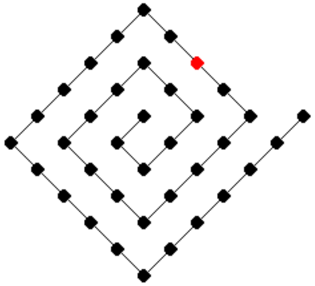
$(x_m, y_m) = (\sum_j x_j, \sum_j y_j) / n$ = center of mass

(a,b) = eigenvector of smallest eigenvalue of 2x2 matrix M = [X1 X2; X2 X3]

Nodal Coordinates — Summary

- Algorithms using nodal coordinates 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 very poorly if graph connection is not spatial

Example (graph that is not spatial connected)



In the printed version, the solutions can be found in the appendix

Content

- Motivation for graph partitioning
- Overview of heuristics
- Partitioning with nodal coordinates
  - Ex: In finite element models, node at point in (x, y, z) space
    Recursive Coordinate Bisection
    Inertial Partitioning
- **Partitioning without nodal coordinates**
  - **Ex: In model of WWW, nodes are web pages**
    Fiduccia-Matteyes
    Spectral Methods
- Multilevel acceleration
  - BIG IDEA, appears often in scientific computing
- Available implementations
- Beyond Graph Partitioning: Hypergraphs

Coordinate-Free: Kernighan/Lin

- Take a initial partition and iteratively improve it
  - Kernighan/Lin (1970), cost = O(|N|3) but easy to understand
  - Fiduccia/Mattheyses (1982), cost = O(|E|), much better, but more complicated
- Given G = (N,E,WE) and a partitioning N = A U B, where |A| = |B|
  - **T = cost(A,B) = S {W(e) where e connects nodes in A and B}**
  - **Find subsets X of A and Y of B with |X| = |Y|**
  - **Consider swapping X and Y if it decreases cost:**
    - newA = (A – X) U Y   and   newB = (B – Y) U X
    - newT = cost(newA , newB) < T = cost(A,B)

- Need to compute newT efficiently for many possible X and Y, choose smallest (best)

Coordinate-Free — Fiduccia-Matteyes

- Take an **initial partition** and iteratively improve it

- Given $G = (V, E, W_E)$ and a partitioning $V = A \cup B$, 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:
    - $newA = A - X \cup Y$ and $newB = B - Y \cup X$
    - $newT = \text{cost}(newA, newB) < \text{cost}(A, B)$

- Need to compute newT efficiently for many possible X and Y, choose smallest

Coordinate-Free — Fiduccia-Matteyes

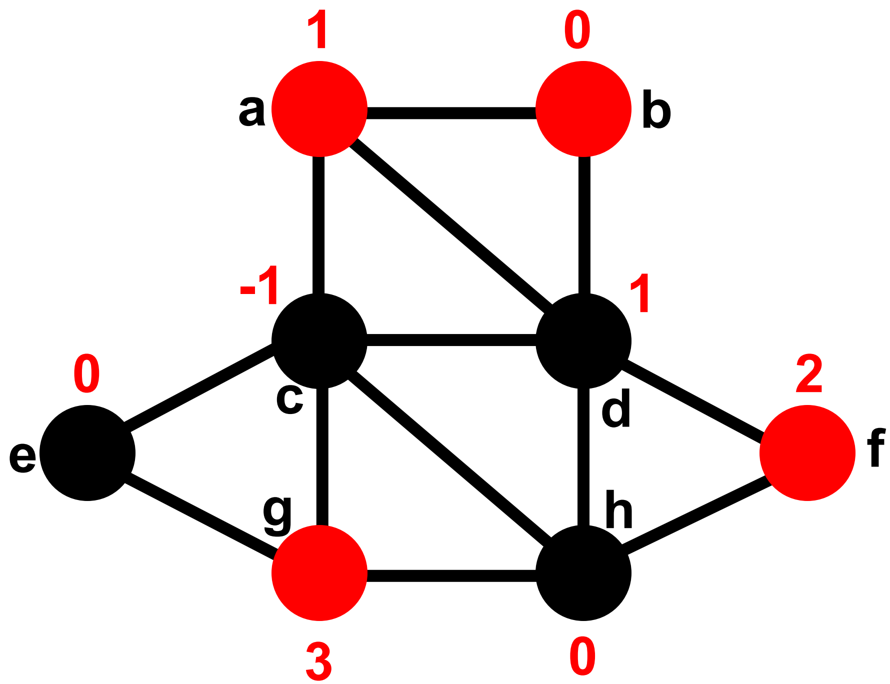
- T = cost(A, B),   newT = cost(newA, newB)
- Need an efficient formula for newT; we will use
  - **E(a) = external cost of a in A = ∑ {W(a, b) for b in B}**
  - **I(a) = internal cost of a in A = ∑ {W(a, a') for other a' in A}**
  - **D(a) = cost of a in A = E(a) - I(a)**
  - **E(b), I(b) and D(b) defined analogously for b in B**

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.



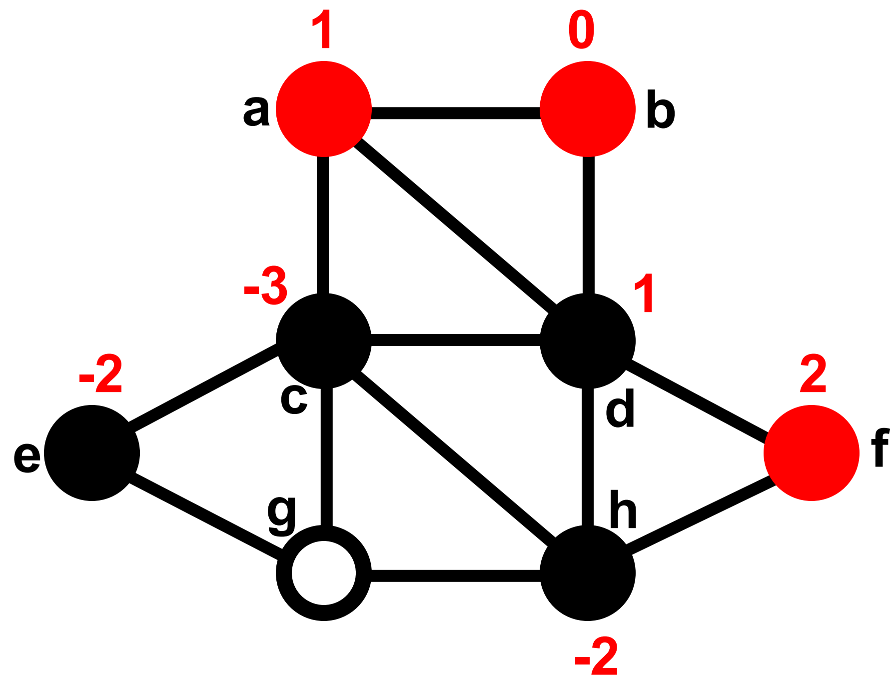
Nodes tentatively moved (and cut size after each pair):

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.



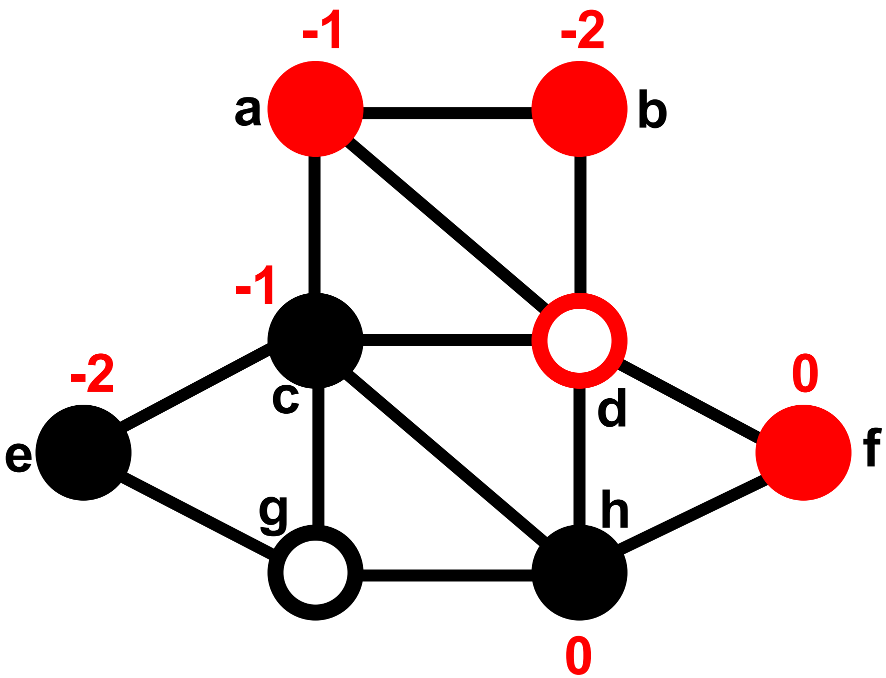
Nodes tentatively moved (and cut size after each pair):

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.

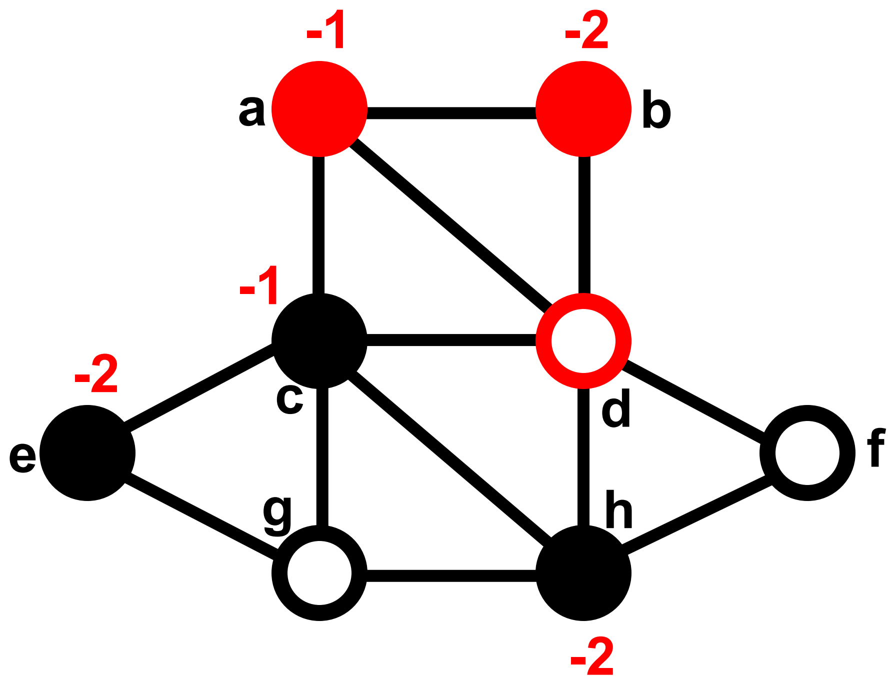


Nodes tentatively moved (and cut size after each pair):

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.



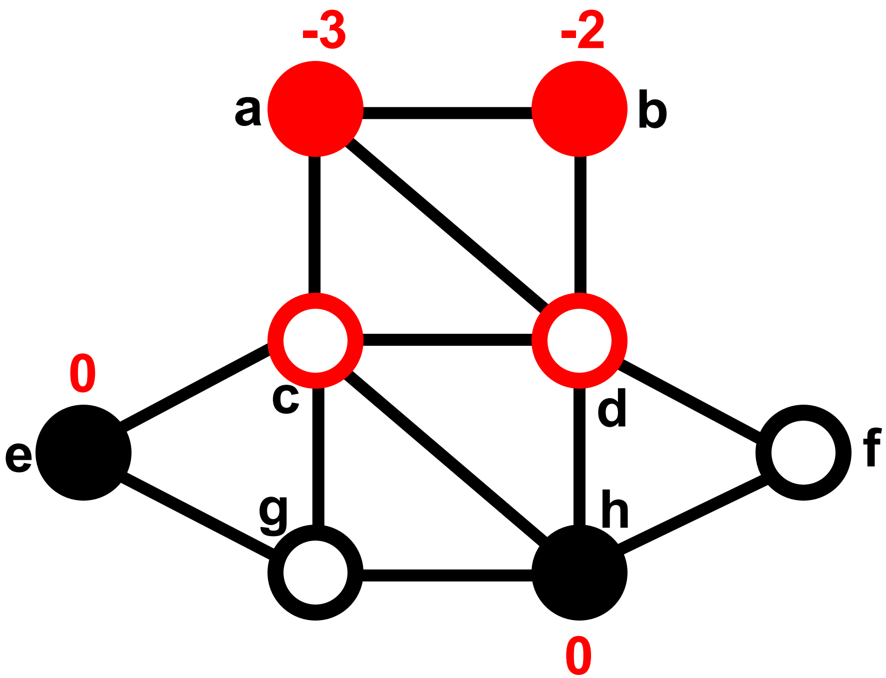
Nodes tentatively moved (and cut size after each pair):

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.

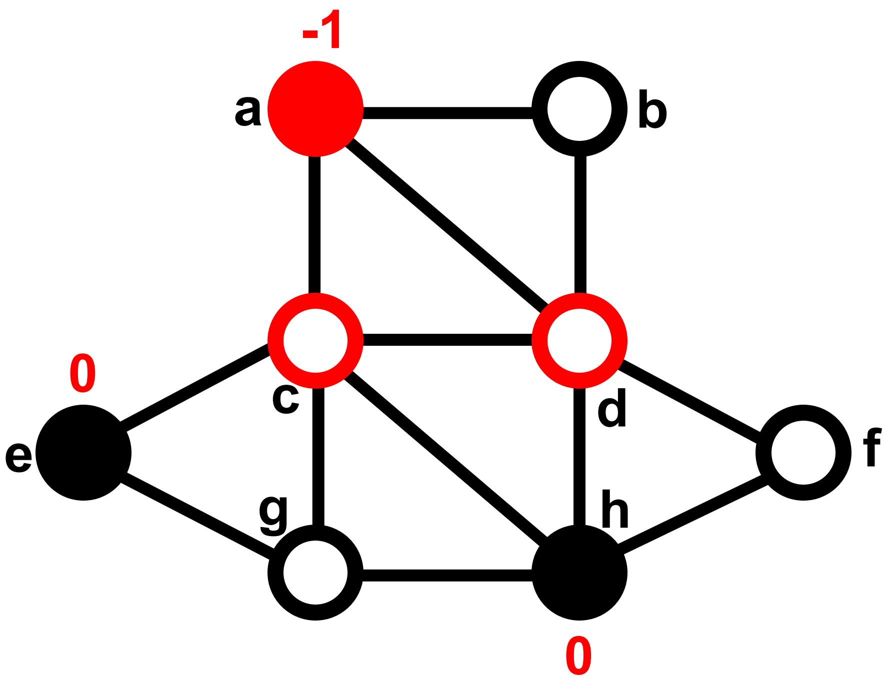


Nodes tentatively moved (and cut size after each pair):

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.

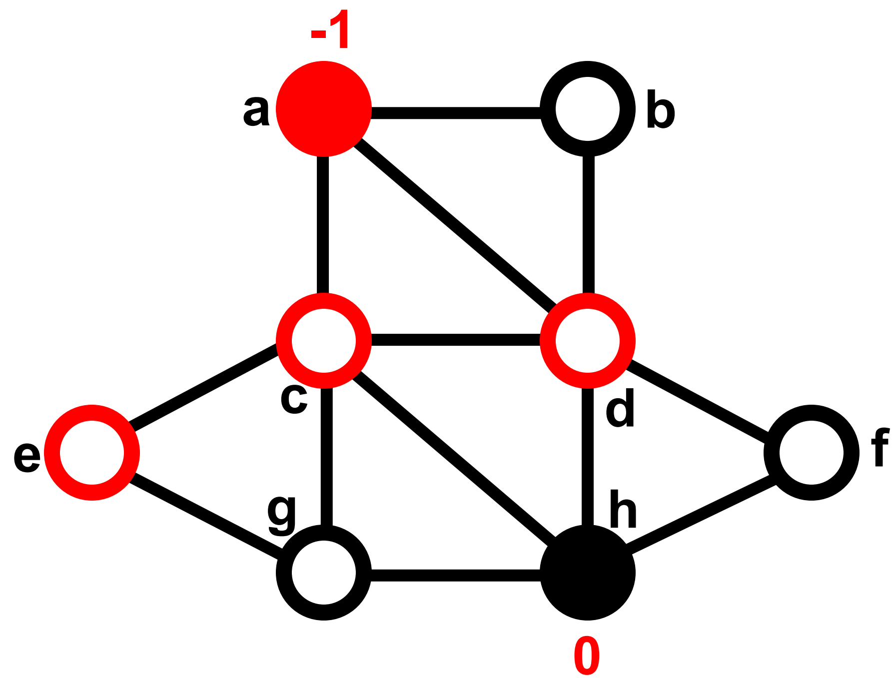


Nodes tentatively moved (and cut size after each pair):

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.



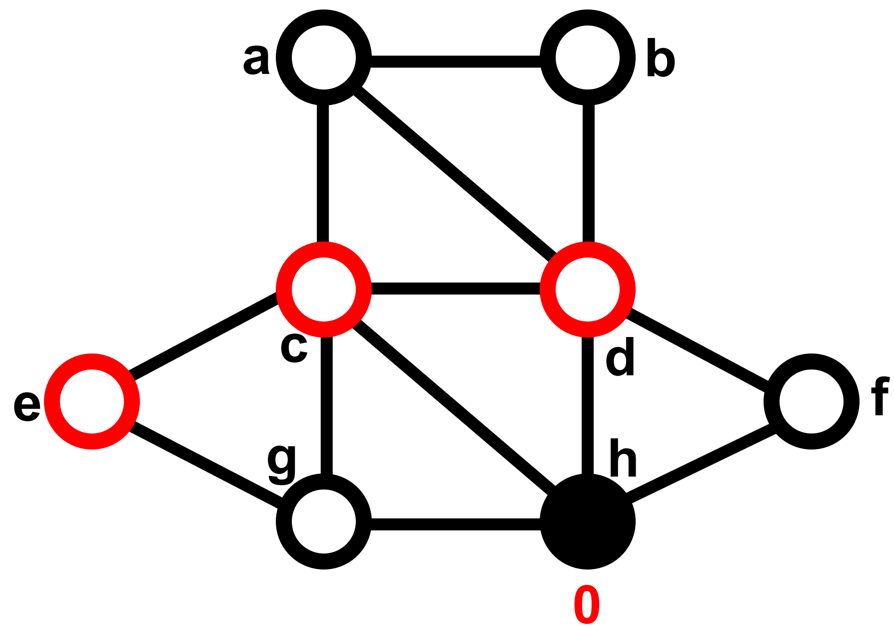
After this tentative swap the cut size is 7.

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.



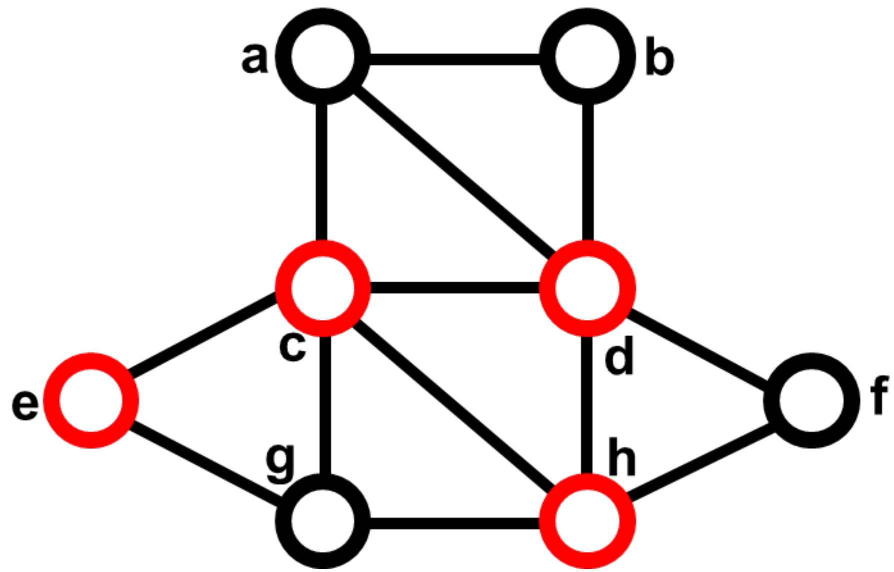
Nodes tentatively moved (and cut size after each pair):

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.

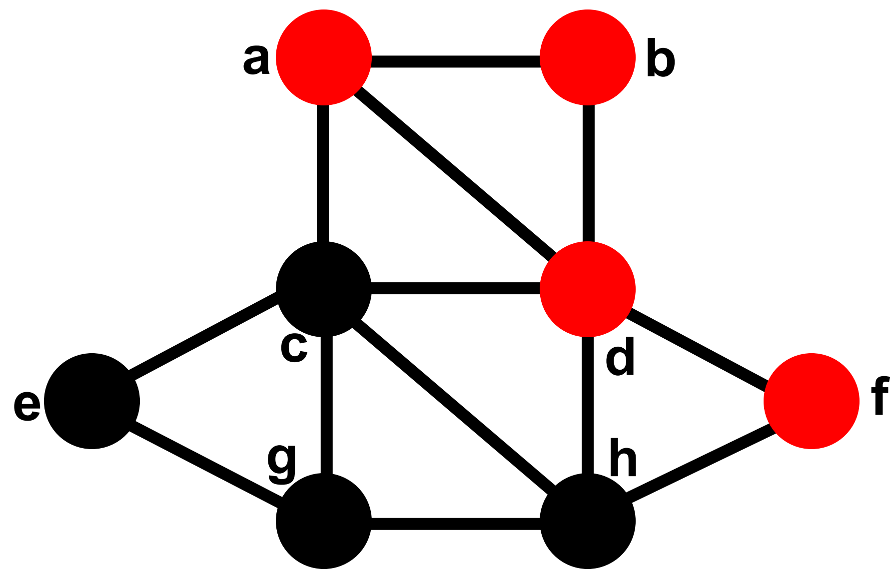


Nodes tentatively moved (and cut size after each pair):

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):

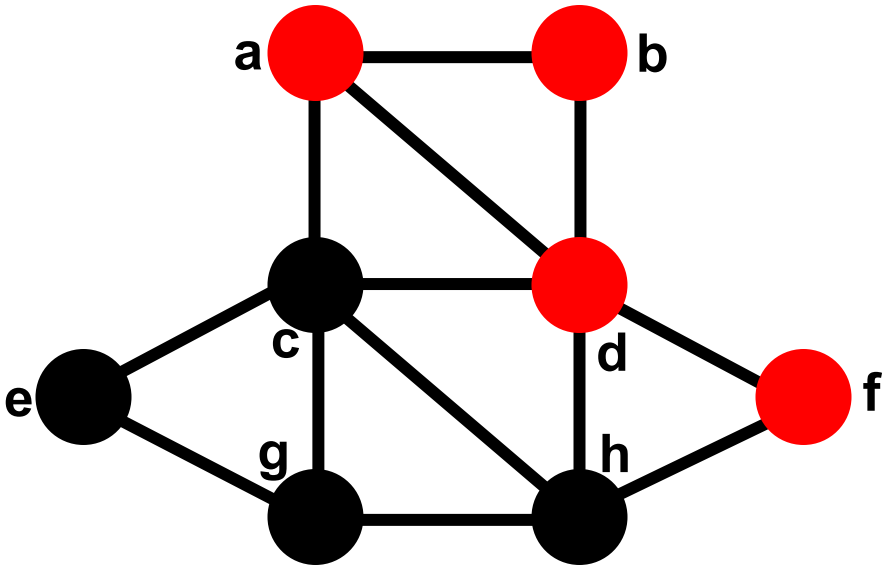
none (8); **g, d** **(4);** f, c (5); b, e (7); a, h (8)

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.



Coordinate-Free — Fiduccia-Matteyes Algorithms (F/M)

- Algorithm:

```
Repeat
    Compute edge-cut T = cost(A, B) for initial A, B                 ...O( |E| )
    Compute D(v) for initial A, B with V = A ∪ B                     ...O( |E| )
    Initialize two queues (Q_A, Q_B) with highest D(v) on top        ...O( |V| log |V| )
    Unmark all nodes in V                                            ...O( |V| )
    While there are nodes v in queue Q_A, Q_B that can be moved      ... |V| iterations
        v = TopNode (Q_A,Q_B) (see next slide)                       ...O( |1| )
        Mark node v and move it in the other subgraph                ...O( |1| )
        Remove node v from selected queue Q_A or Q_B                 ...O( |1| )
        // v has d neighbours
        Update gains of adjacent vertices v                          ...O( |d| )
        Order priority queue Q_A and Q_B                             ...O( |d| )
    Update T = T – D(v)                                              ...O( |1| )
    Endwhile
Until T <= 0                                   Total Complexity      ...|V|*|d| = |E|
```

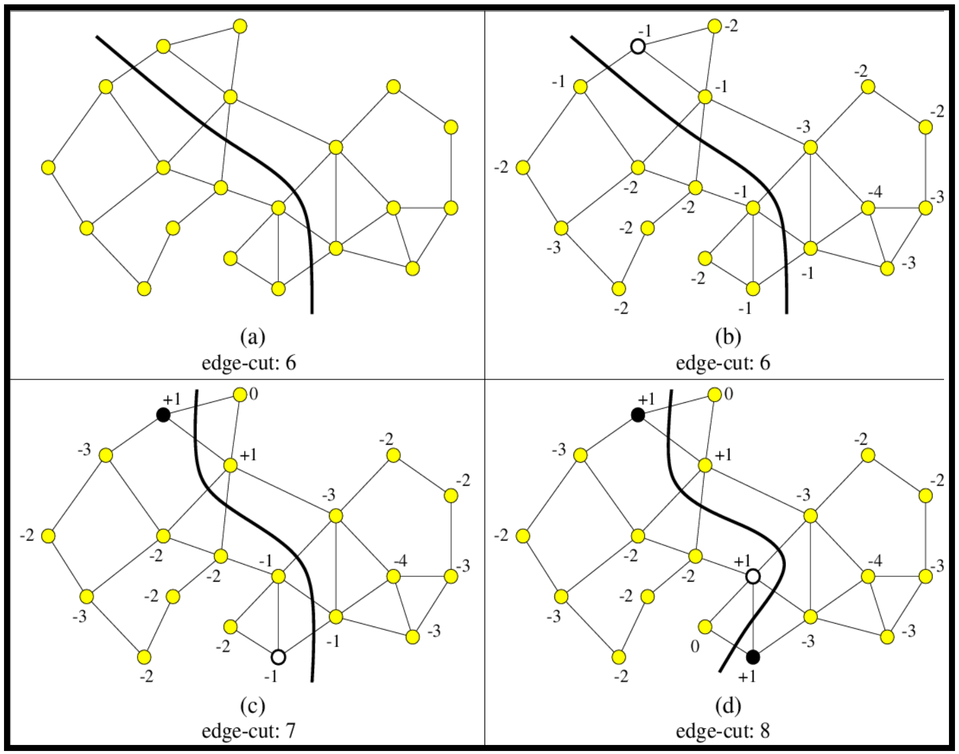
- FM allows a small load imbalance and swaps only one vertex in each iteration

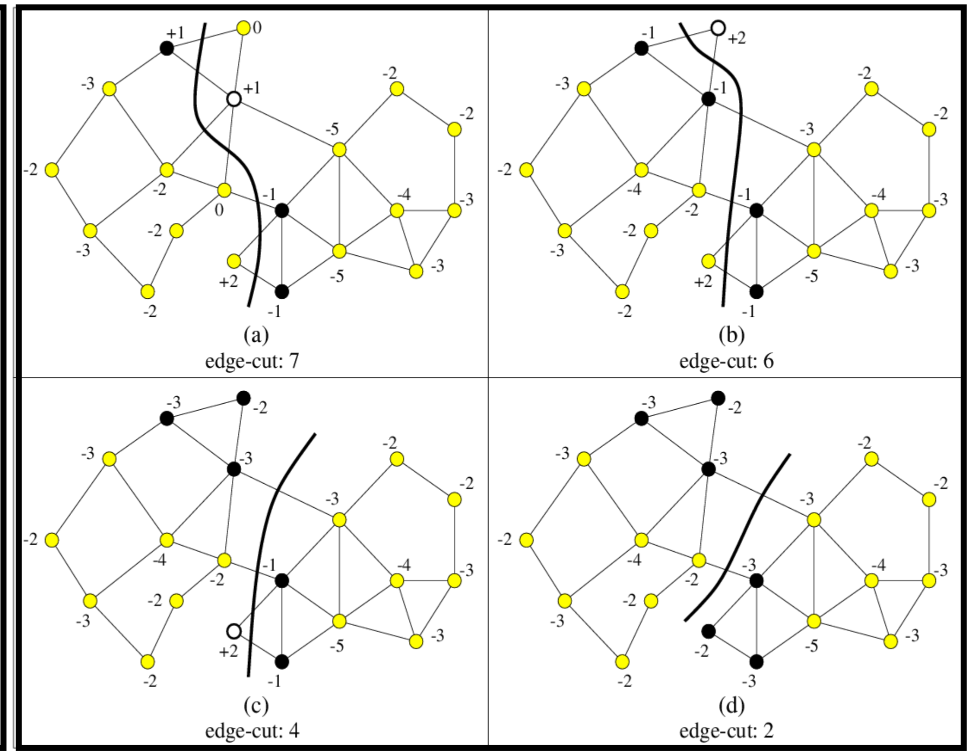
Coordinate-Free — Fiduccia-Matteyes Algorithms (F/M)

```
node v = TopNode (Q_A , Q_B)
    … selection of vertex v from Q_A or Q_B
    a := top node in queue Q_A, b:= top node queue Q_B
    // if one domain is (1+eps) larger than the other domain,
    // select the vertex from the smaller domain
    if (  ( |A| > (1+eps)|B| )  or  ( |B| > (1+eps)|A|)  )
            select the node v from larger domain
    else
    {
            if ( D (a) < D (b) )
                        select node b to move
            if ( D (a) > D (b) )
                        select node a to move
            if ( D (a) = D (b) )
             select node from larger domain
    }
```

Coordinate-Free — Fiduccia-Matteyes Algorithms (F/M)

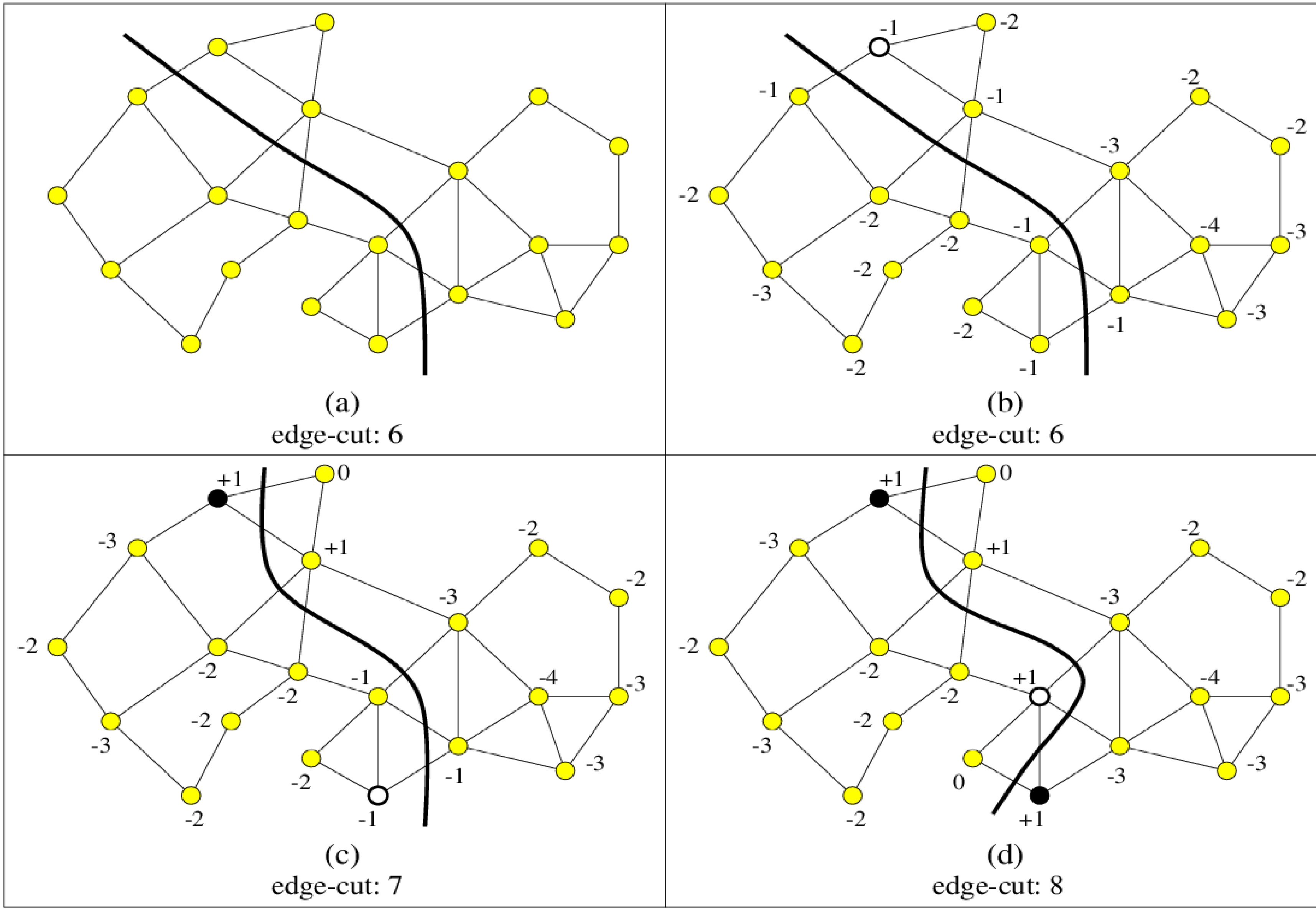
- **Example:**





- All vertices v are marked with D(v) — Reduction of edge-cut.
- Load imbalance: 10% - therefore in iteration (c) the vertex marked with D(v) = +1 can not move into other domain.

Coordinate-Free — Fiduccia-Matteyes Algorithms (F/M)



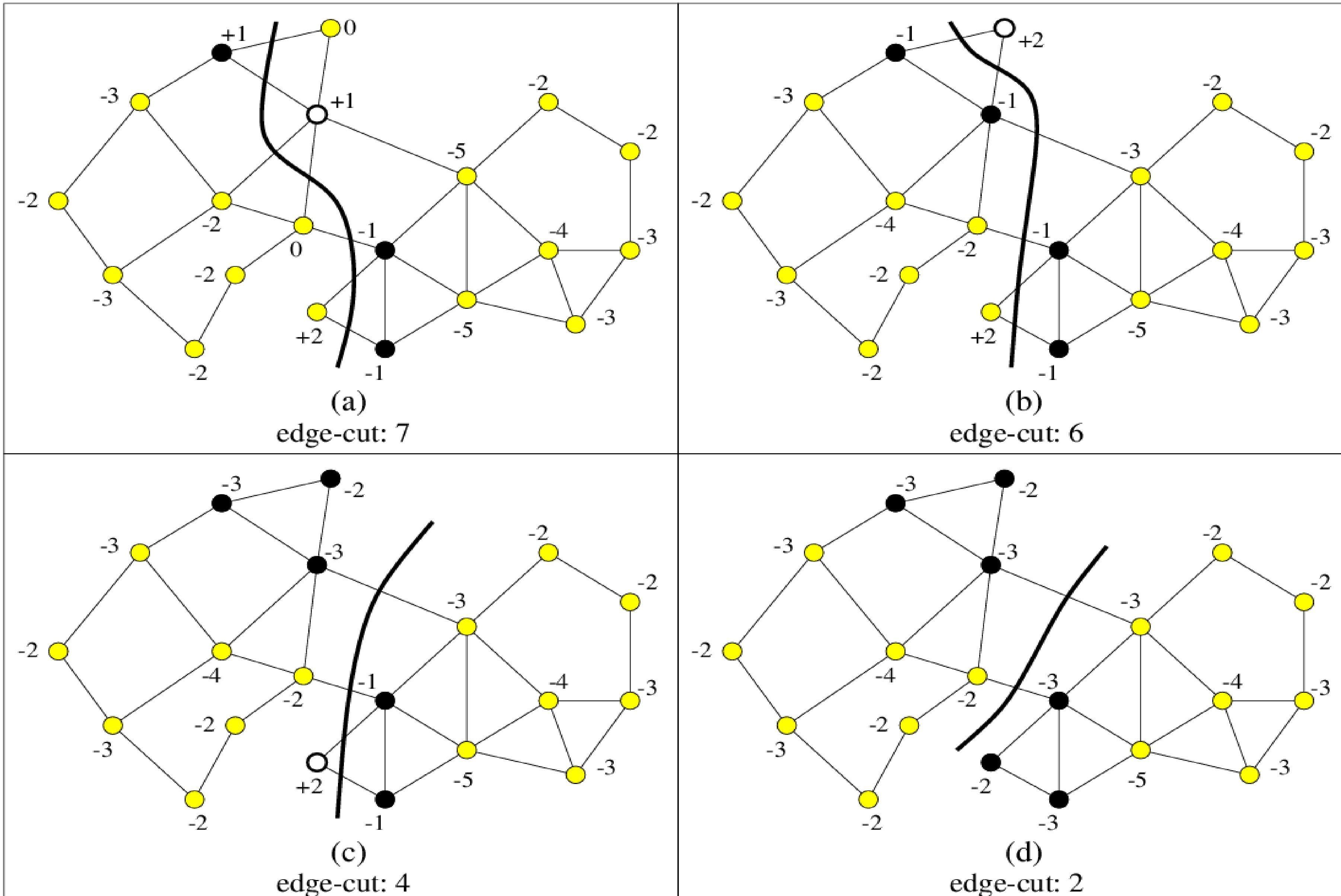
(a) edge-cut: 6

(b) edge-cut: 6

(c) edge-cut: 7

(d) edge-cut: 8

Coordinate-Free — Fiduccia-Matteyes Algorithms (F/M)

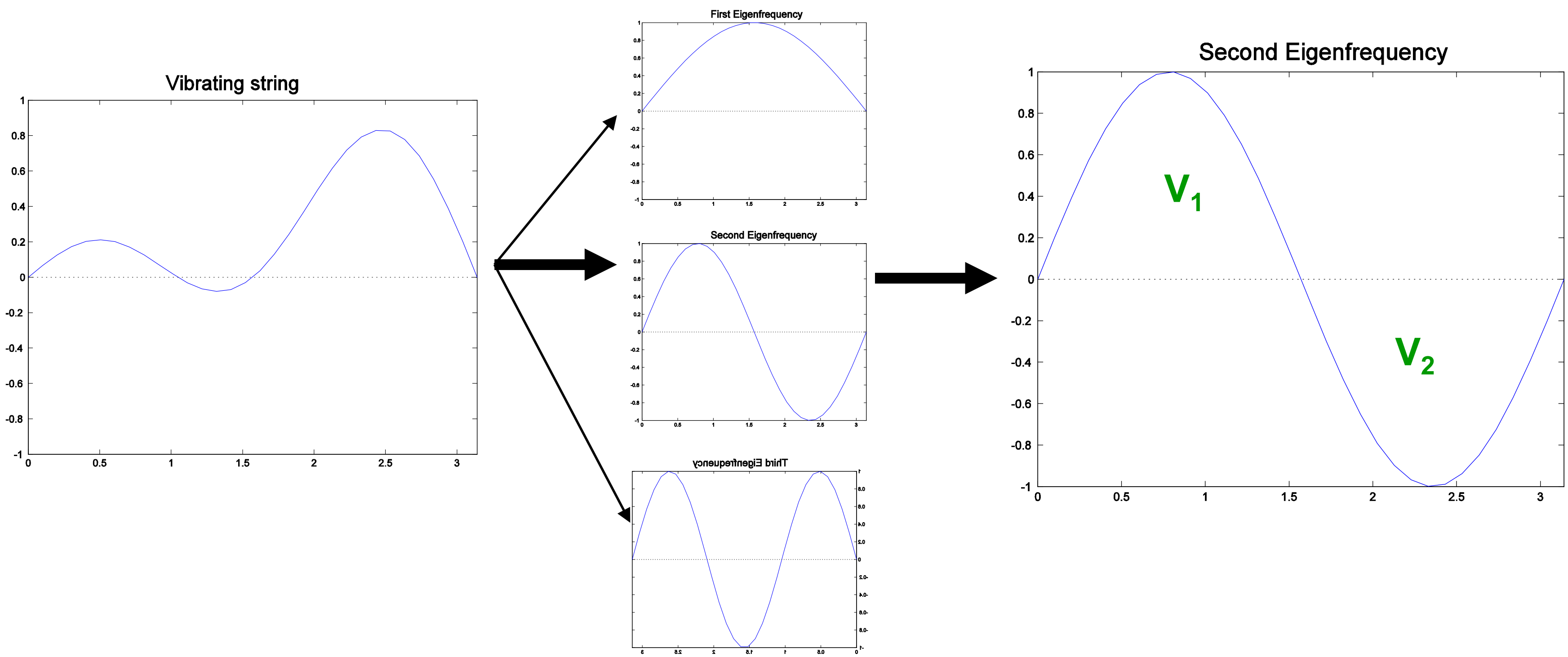


Coordinate-Free — Spectral Methods

- **Spectral methods** as an example for **global partitioning** algorithms
- Heavily use of Eigenvalue/Eigenvector analysis

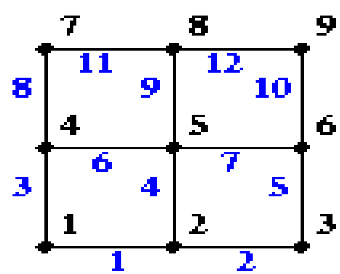
Coordinate-Free — Spectral Methods

- Based on theory of Fiedler (1970s), popularized by Horst Simon (1995)
- First motivation with vibrating string
- Label nodes by whether mode - or + to partition into V_1 and V_2



Coordinate-Free — Spectral Methods, Basic Definitions

- **Definition:** The **Laplacian matrix L(G)** of a graph G(V, E) is a |V| by |V| 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



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

Properties of Laplacian matrices

- **Theorem:** Given a graph 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**.
  - Let $e = [1,\ldots,1]^T$, i.e. the column vector of all ones. Then $L(G)*e=0*e = 0$
  - 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 l_i equal to 0.

- **Definition**: λ_2 (L(G)) is the **algebraic connectivity** of G
  - The magnitude of λ_2 measures connectivity
  - In particular, λ_2 != 0 if and only if G is connected

Relation between Laplace Matrix and Graph Partitioning

- **Theorem (Fiedler, 1975):**
  Let G be connected, L(G) the Laplace matrix, and N_+ and N_- a partitioning with

$$x(i) = +1 \quad \text{if } v_i \text{ in } N_+$$
$$x(i) = -1 \quad \text{if } v_i \text{ in } N_-.$$

Then we have the following property:

#edge-cut between N_+ and N_-

$$= \frac{1}{4} * x^T * L(G) * x$$

Proof: (next slide)

Relation between Laplace Matrix and Graph Partitioning

$$
\begin{aligned}
\boxed{x^T \cdot L(G) \cdot x} &= \sum_j \sum_i L(G)_{(i,j)} \cdot x_i \cdot x_j \\
&= \sum_{i=j} L(G)_{(i,i)} \cdot x_i^2 + \sum_{i \neq j} L(G)_{(i,j)} \cdot x_i \cdot x_j \\
&= \sum_{i=j} L(G)_{(i,i)} \cdot x_i^2 \\
&\quad + \sum_{i \neq j;\ i,\, j \in N^+} L(G)_{(i,j)} \cdot x_i \cdot x_j + \sum_{i \neq j;\ i,\, j \in N^-} L(G)_{(i,j)} \cdot x_i \cdot x_j \\
&\quad + \sum_{i \neq j;\ i \in N^+,\, j \in N^-} L(G)_{(i,j)} L(G)_{(i,j)} \cdot x_i \cdot x_j \\
&= \sum_i degree(i) \\
&\quad + \sum_{i \neq j;\ i,\, j \in N^+} (-1) \cdot (+1) \cdot (+1) + \sum_{i \neq j;\ i,\, j \in N^-} (-1) \cdot (-1) \cdot (-1 \\
&\quad + \sum_{i \neq j;\ i \in N^+,\, j \in N^-} (-1) \cdot (+1) \cdot (-1)
\end{aligned}
$$

Relation between Laplace Matrix and Graph Partitioning

$$
\begin{aligned}
x^T \cdot L(G) \cdot x &= \sum_j \sum_i L(G)_{(i,j)} \cdot x_i \cdot x_j \\
&= \sum_{i=j} L(G)_{(i,i)} \cdot x_i^2 + \sum_{i \neq j} L(G)_{(i,j)} \cdot x_i \cdot x_j \\
&= \sum_{i=j} L(G)_{(i,i)} \cdot x_i^2 \\
&\quad + \sum_{i \neq j;\ i,\ j \in N^+} L(G)_{(i,j)} \cdot x_i \cdot x_j + \sum_{i \neq j;\ i,\ j \in N^-} L(G)_{(i,j)} \cdot x_i \cdot x_j \\
&\quad + \sum_{i \neq j;\ i \in N^+,\ j \in N^-} L(G)_{(i,j)} L(G)_{(i,j)} \cdot x_i \cdot x_j \\
&= \sum_i degree(i) \\
&\quad + \sum_{i \neq j;\ i,\ j \in N^+} (-1) \cdot (+1) \cdot (+1) + \sum_{i \neq j;\ i,\ j \in N^-} (-1) \cdot (-1) \cdot (-1) \\
&\quad + \sum_{i \neq j;\ i \in N^+,\ j \in N^-} (-1) \cdot (+1) \cdot (-1)
\end{aligned}
$$

Relation between Laplace Matrix and Graph Partitioning

$$\boxed{x^T \cdot L(G) \cdot x} = \sum_{i,\,j} L(G)_{(i,j)} \cdot x_i \cdot x_j$$

$$= \sum_i degree(i)$$

$$+ \sum_{i \neq j;\ i,\,j \in N^+} (-1)\cdot(+1)\cdot(+1) + \sum_{i \neq j;\ i,\,j \in N^-} (-1)\cdot(-1)\cdot(-1)$$

$$+ \sum_{i \neq j;\ i \in N^+,\, j \in N^-} (-1)\cdot(+1)\cdot(-1)$$

$$= \quad 2 \cdot \textbf{\#edges in G}$$

$$-2 \cdot (\textbf{\#edges connecting node in } N^+ \textbf{ to nodes in } N^+)$$

$$-2 \cdot (\textbf{\#edges connecting node in } N^- \textbf{ to nodes in } N^-)$$

$$+2 \cdot (\textbf{\#edges connecting node in } N^+ \textbf{ to nodes in } N^-)$$

$$= \boxed{4 \cdot (\textbf{\#edges connecting node in } N^+ \textbf{ to nodes in } N^-)}$$

Relation between Laplace Matrix and Graph Partitioning

- With the theorem we can formulate the **graph bisection** as a discrete optimization problem

1.	$\lvert V_1\rvert = \lvert V_2\rvert$	$\Leftrightarrow \sum_i x(i) = 0$
2.	min #cut edges between V_1 and V_2	$\Leftrightarrow$ min $x^T * L(G) * x$

or

min	$f(x) = ¼\, x^T * L(G) * x$
constraints	$x_I = \{+/- 1\}$, $x^T * x = n$
	$x^T * e = 0$ with $e=[1, 1,\ldots,1]^T$

- The **discrete combinatorial** problem is NP-hard → use a **continuous problem**

min	$f(z) = ¼\, z^T * L(G) * z$
constraints	$z^T * z = n$, z real vector
	$z^T * e = 0$ with $e=[1,1,\ldots,1]^T$

Relation between Laplace Matrix and Graph Partitioning

- Let’ try to solve the continuous graph bisection problem

Relation between Laplace Matrix and Graph Partitioning

- Minimal solution of $z^T * L(G) * z$ is easy to find.
- $L(G)$ is symmetric $\rightarrow$ $L(G)$ has n orthonormal eigenvectors $u_1, \ldots, u_n$ with eigenvalues $0 = \lambda_1 \leq \ldots \leq \lambda_n$ and $u_1 = \mathrm{sqrt}(n) * e$, $e = [1, 1, \ldots, 1]^T$.
- A vector z is a linear combination of eigenvectors u_i:

$$z = \sum \alpha_i u_i = \alpha_1 u_1 + \alpha_2 u_2 + \ldots + \alpha_n u_n.$$

- **First constrained**: $z^T * e = 0$ or $z^T * u_1 = 0$ it is necessary that

$$z^T * u_1 = (\sum \alpha_i u_i)^T * u_1 = \alpha_1 u_1^T * u_1 = \alpha_1 \quad \Rightarrow \boldsymbol{\alpha_1 = 0}$$

- **Second constrained:** $z^T * z = n$ it is necessary that

$$z^T * z = (\sum \alpha_i u_i)^T * (\sum \alpha_j u_j) = \sum\sum \alpha_i \alpha_j u_i^T * u_j = \sum \alpha_i^2 = n$$

- <u>**Minimize $4 * f(z) = z^T * L(G) * z$**</u>

$$z^T * L(G) * z = (\sum \alpha_i u_i) * L * (\sum \alpha_j u_j) = (\sum \alpha_i u_i)^T * (\sum \alpha_j \lambda_j u_j) =$$
$$\sum\sum \alpha_j \alpha_i \lambda_j u_i^T * u_j = \sum \alpha_i^2 \lambda_j \geq \lambda_2 \sum \alpha_i^2 = \boldsymbol{\lambda_2 * n}$$

Relation between Laplace Matrix and Graph Partitioning

- **Minimize $4*f(z)= z^T * L(G) * z$**

$$z^T*L(G)*z \quad = \quad (\sum\alpha_i u_i) *L*(\sum\alpha_j u_j) =(\sum\alpha_i u_i)^T*(\sum\alpha_j \lambda_j u_j) =$$
$$\sum\sum\alpha_j \alpha_i \lambda_j u_i^T*u_j = \sum\alpha_i^2 \lambda_j \quad \geq \quad \lambda_2 \sum\alpha_i^2 = \boldsymbol{\lambda_2 * n}$$

- Minimum is at $z = \text{sqrt}(n)*u_2$.
- <u>**Spectral Bisection Algorithm**</u>**:**
  - Compute eigenvector u_2 corresponding to $\boldsymbol{\lambda}_2 (L (G))$
  - For each vertex v of G
    - if $u_2(v) < 0$ put node v in partition V_1
    - else put vertex v in partition V_2
- The second eigenvector u_2 is called **Fiedler Eigenvector** of the Graph Partitioning problem.

Content

- Motivation for graph partitioning
- Overview of heuristics
- Partitioning with nodal coordinates
  - Ex: In finite element models, node at point in (x, y, z) space
    Recursive Coordinate Bisection
    Inertial Partitioning
- Partitioning without nodal coordinates
  - Ex: In model of WWW, nodes are web pages
    Fiduccia-Matteyes
    Spectral Methods
- Multilevel Acceleration
  - **BIG IDEA**, appears often in scientific computing
- Available Implementations
- Beyond Graph Partitioning: Hypergraphs

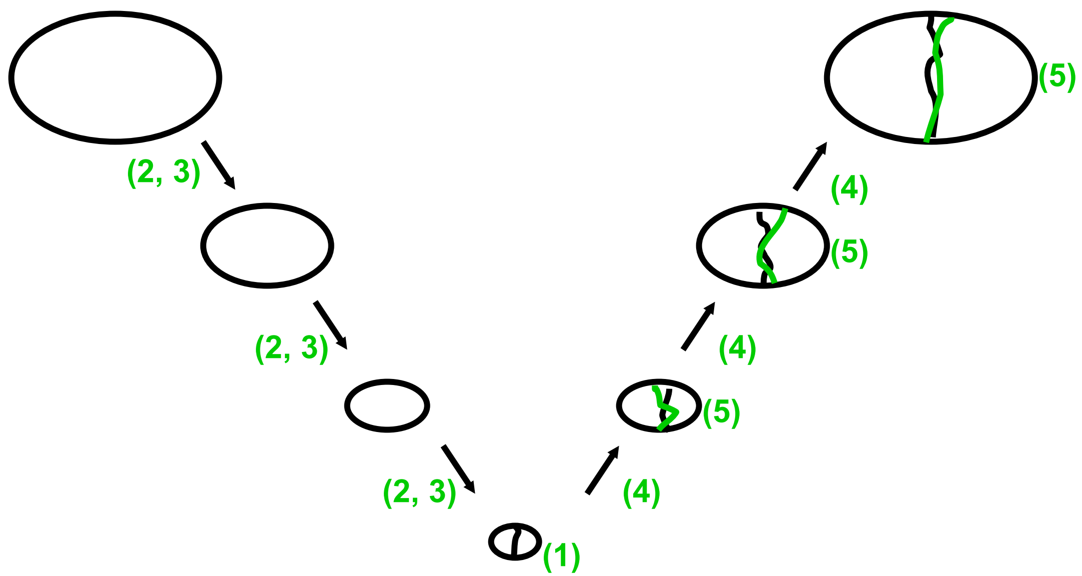
Multilevel Partitioning — Introduction

- If we want to partition G(V, E), but it is too big to do efficiently, what can we do?
  - 1) Replace G(V, E) by a **coarse approximation** G_C (V_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

```
    (V+, V- ) = Multilevel_Partition( V, E )
        // recursive partitioning routine returns V+ and V- where V = V+ U V-
        if |V| is small
(1)         Partition G = (V, E)  directly to get V = V+ U V-
            Return (V+, V- )
        else
(2)         Coarsen G to get an approximation Gc = (Vc, Ec)
(3)         (Vc+ , Vc- ) = Multilevel_Partition( Vc, Ec )
(4)         Expand (Vc+ , Vc- ) to a partition  (V+ , V- ) of V
(5)         Improve the partition ( V+ , V- )
            Return ( V+ , V- )
        endif
```

How do we
Coarsen?
Expand?
Improve?



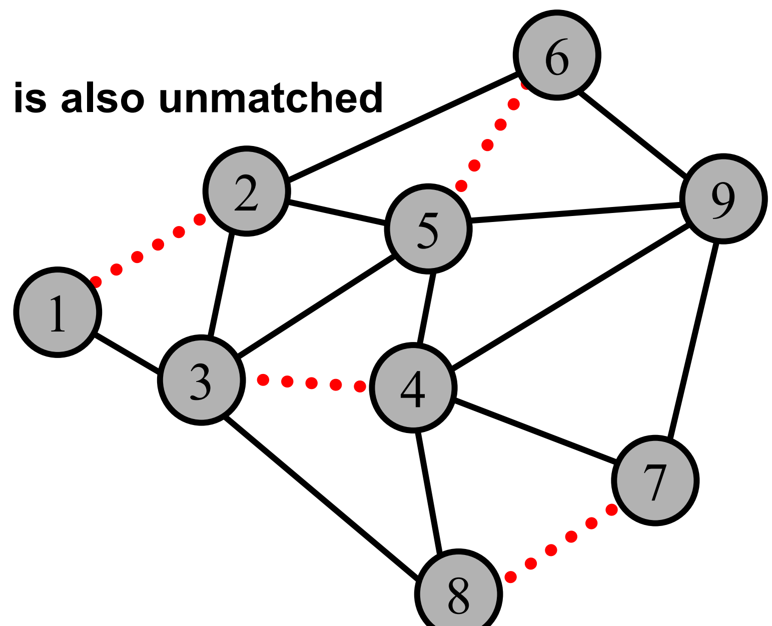
Multilevel Partitioning — Multilevel Fiducca-Matteyes

- **Coarsen** graph and **expand** partition using **maximal matchings**
- **Improve** partition using Fiduccia-Matteyes

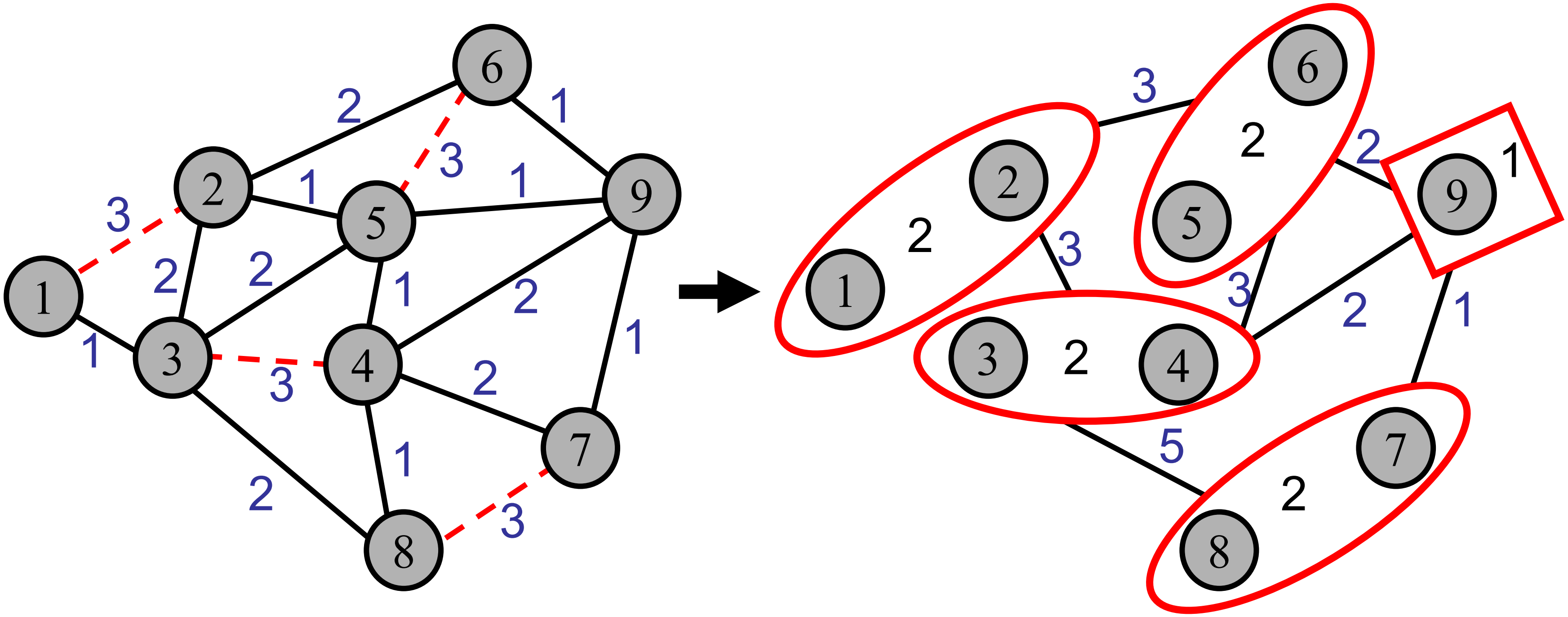
Multilevel Partitioning — Maximal Matching

- *Definition*: A **matching** of a graph G(V, 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(V, 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 V as unmatched
for vertex i = 1 to |V|      // visit the nodes in any order
   if i has not been matched
       mark vertex i as matched
       if there is an edge e=(i, j) where vertex j is also unmatched
           add e to E_m
           mark vertex j as matched
       endif
   endif
end
```



Multilevel Partitioning — Coarsening



$G = (V, E)$

Matching E_m is red

Edge weights are blue

Vertex weights all 1

$G_c = (V_c, E_c)$

Vertices V_c are red

Edge weights are blue

Vertex weights are black

Multilevel Partitioning — Coarsening with maximal matchings

1) Construct a maximal matching E_m of $G(V, E)$

2) Collapse matched nodes into a single one

for all edges $e = (j, k)$ in E_m

Put vertex $v(e)$ in V_c

$W(v(e)) = W(j) + W(k)$ // update vertex weights

3) Add unmatched vertices

for all vertices v in V not incident on an edge in E_m

Put v in V_c // do not change $W(n)$

// Now each vertex r in V is “inside” a unique node $v(r)$ in V_c

// Compute now the edges and edge weights of the coarse graph

4) Connect two vertices in V_C if vertices inside them are connected in C

for all edges $e = (j, k)$ in E_m

for each other edge $e' = (j, r)$ or (k, r) in E

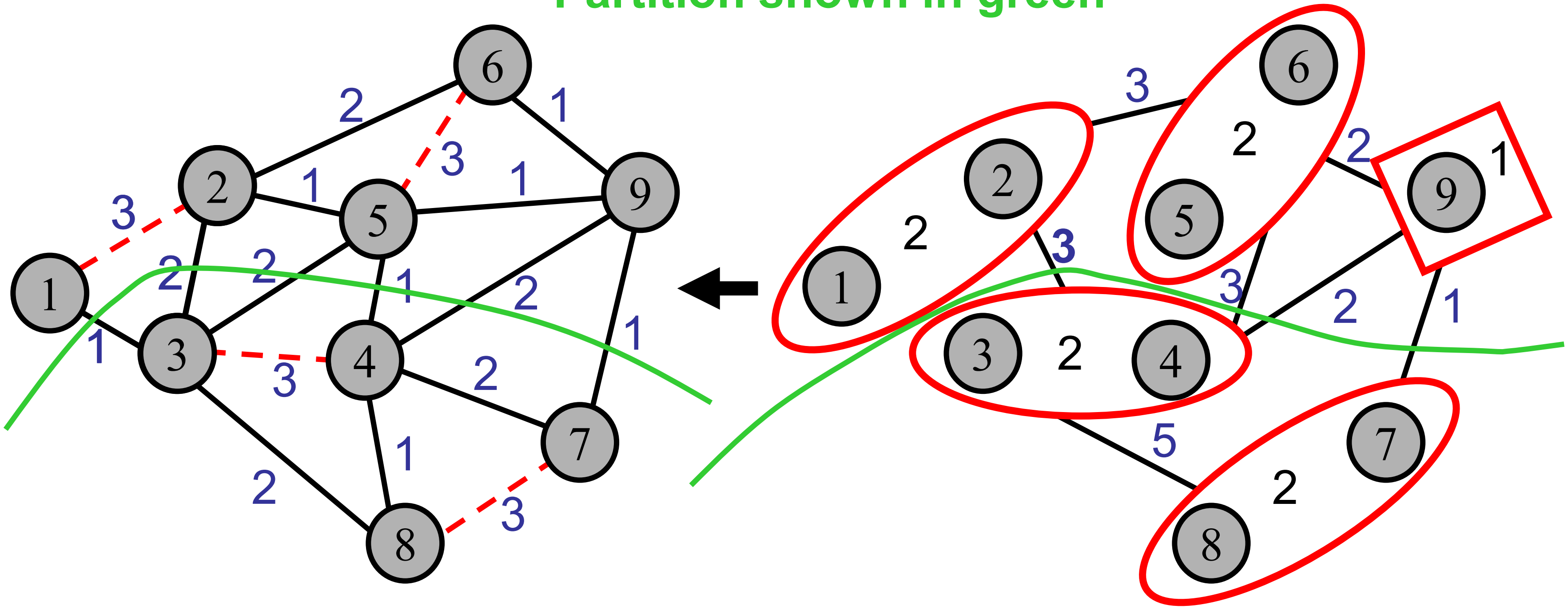
Put edge $ee = (v(e), v(r))$ in E_c

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

If there are multiple edges connecting two vertices in C_c, collapse them, adding edge weights

Multilevel Partitioning — Expanding a partitioning of G_c to G

Partition shown in green



G= (V, E)

Matching E_m is red

Edge weights are blue

Vertex weights all 1

G_c = (V_c, E_c)

Vertices V_c are red

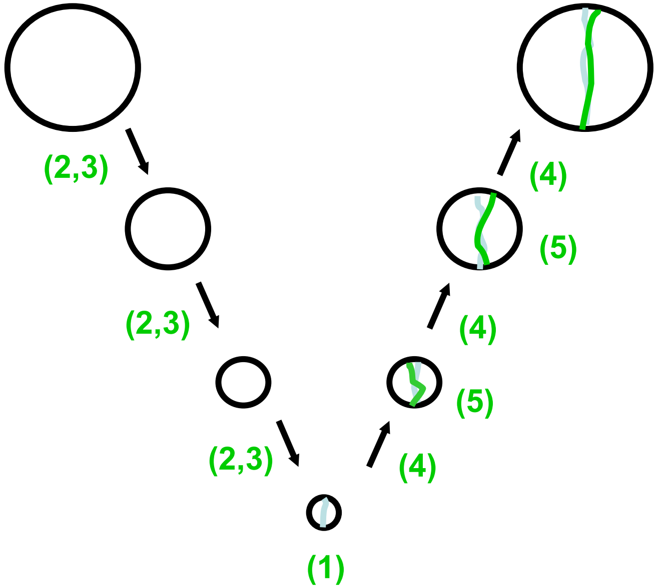
Edge weights are blue

Vertex weights are black

Multilevel Spectral Bisection

f = Fiedler (V, E)
… Recursive computation of Fiedler Vector of Laplacian L(G)
if |V| is small
(1) Calculate $f=u_2$ using eigenvalue/eigenvector algorithms
Return f
else
(2) Coarsen G to get an approximation $G_c = (V_c, E_c)$
(3) $f' = \text{Fiedler}(V_c, E_c)$
(4) Use f' to find an inital guess for $f^{(0)}$
(5) improve f from the initial guess $f^{(0)}$
Return f
endif

How do we
Coarsen?
use initial guess?
improve the initial guess?



Content

- Motivation for graph partitioning
- Overview of heuristics
- Partitioning with nodal coordinates
  - Ex: In finite element models, node at point in (x, y, z) space
    Recursive Coordinate Bisection
    Inertial Partitioning
- Partitioning without nodal coordinates
  - Ex: In model of WWW, nodes are web pages
    Fiduccia-Matteyes
    Spectral Methods
- Multilevel Acceleration
  - BIG IDEA, appears often in scientific computing
- Available Implementations
- Beyond Graph Partitioning: Hypergraphs

Available Implementations

- Multilevel Kernighan/Lin
  - METIS and ParMETIS (glaros.dtc.umn.edu/gkhome/views/metis)
  - SCOTCH and PT-SCOTCH (www.labri.fr/perso/pelegrin/scotch/)
- 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/)

METIS - Family of Graph and Hypergraph Partitioning Software

Karypis Lab

Home | Contact | METIS | CLUTO | BDMPI | Forums

Search

Navigation Menu

- Home
- Research
  - Projects
  - Software
    - METIS
      - METIS
      - ParMETIS
      - hMETIS
    - CLUTO
    - BDMPI
    - PAFI
    - AFGen
    - SUGGEST
    - L2AP
    - L2Knng
    - MGridGen
    - SLIM
    - SPLATT
    - PSPASES
  - Publications
- Education
- Lab Information

Home » Research » Software

Family of Graph and Hypergraph Partitioning Software

METIS - Serial Graph Partitioning and Fill-reducing Matrix Ordering

METIS stable version: 5.1.0, 3/30/2013; MT-METIS version: 0.6.0, 10/30/2016

METIS is a set of serial programs for partitioning graphs, partitioning finite element meshes, and producing fill reducing orderings for sparse matrices. The algorithms implemented in METIS are based on the multilevel recursive-bisection, multilevel *k*-way, and multi-constraint partitioning schemes developed in our lab.

» Read more

ParMETIS - Parallel Graph Partitioning and Fill-reducing Matrix Ordering

Current stable version: 4.0.3, 3/30/2013

ParMETIS is an MPI-based parallel library that implements a variety of algorithms for partitioning unstructured graphs, meshes, and for computing fill-reducing orderings of sparse matrices. ParMETIS extends the functionality provided by METIS and includes routines that are especially suited for parallel AMR computations and large scale numerical simulations. The algorithms implemented in ParMETIS are based on the parallel multilevel *k*-way graph-partitioning, adaptive repartitioning, and parallel multi-constrained partitioning schemes developed in our lab.

» Read more

hMETIS - Hypergraph & Circuit Partitioning

Current version: 1.5.3, 11/22/98 [Alpha version: 2.0pre1, 5/24/07]

hMETIS is a set of programs for partitioning hypergraphs such as those corresponding to VLSI circuits. The algorithms implemented by hMETIS are based on the multilevel hypergraph partitioning schemes developed in our lab.

» Read more

http://glaros.dtc.umn.edu/gkhome/views/metis

KaHyPar - Karlsruhe Hypergraph Partitioning

KaHyPar - Karlsruhe Hypergraph Partitioning

A multilevel hypergraph partitioning framework providing direct k-way and recursive bisection based partitioning algorithms that compute solutions of very high quality.

View the Project on GitHub

This project is maintained by SebastianSchlag

Hosted on GitHub Pages — Theme by orderedlist

KaHyPar - Karlsruhe Hypergraph Partitioning

License	Linux & macOS Build	Windows Build	Code Coverage	Coverity Scan	SonarQube	Fossa
License GPL v3	build passing	build passing	codecov 80%	coverity passed	Qualit	license scan passing

What is a Hypergraph? What is Hypergraph Partitioning?

Hypergraphs are a generalization of graphs, where each (hyper)edge (also called net) can connect more than two vertices. The *k*-way hypergraph partitioning problem is the generalization of the well-known graph partitioning problem: partition the vertex set into *k* disjoint blocks of bounded size (at most $1 + \varepsilon$ times the average block size), while minimizing an objective function defined on the nets.

The two most prominent objective functions are the cut-net and the connectivity (or $\lambda - 1$) metrics. Cut-net is a straightforward generalization of the edge-cut objective in graph partitioning (i.e., minimizing the sum of the

http://kahypar.org

Number of edges cut for a 64-way partition, by METIS

For Multilevel Kernighan/Lin, as implemented in METIS (see KK95a)

Graph	# of Nodes	# of Edges	# Edges cut for 64-way partition	Expected # cuts for 2D mesh	Expected # cuts for 3D mesh	Description
144	144649	1074393	88806	6427	31805	3D FE Mesh
4ELT	15606	45878	2965	2111	7208	2D FE Mesh
ADD32	4960	9462	675	1190	3357	32 bit adder
AUTO	448695	3314611	194436	11320	67647	3D FE Mesh
BBMAT	38744	993481	55753	3326	13215	2D Stiffness M.
FINAN512	74752	261120	11388	4620	20481	Lin. Prog.
LHR10	10672	209093	58784	1746	5595	Chem. Eng.
MAP1	267241	334931	1388	8736	47887	Highway Net.
MEMPLUS	17758	54196	17894	2252	7856	Memory circuit
SHYY161	76480	152002	4365	4674	20796	Navier-Stokes
TORSO	201142	1479989	117997	7579	39623	3D FE Mesh

Expected # cuts for 64-way partition of 2D mesh of n nodes

$$n^{1/2} + 2*(n/2)^{1/2} + 4*(n/4)^{1/2} + \ldots + 32*(n/32)^{1/2} \sim 17 * n^{1/2}$$

Expected # cuts for 64-way partition of 3D mesh of n nodes =

$$n^{2/3} + 2*(n/2)^{2/3} + 4*(n/4)^{2/3} + \ldots + 32*(n/32)^{2/3} \sim 11.5 * n^{2/3}$$

Speed of 256-way partitioning (from KK95a)

Partitioning time in seconds

Graph	# of Nodes	# of Edges	Multilevel Spectral Bisection	Multilevel Kernighan/Lin	Description
144	144649	1074393	607.3	48.1	3D FE Mesh
4ELT	15606	45878	25.0	3.1	2D FE Mesh
ADD32	4960	9462	18.7	1.6	32 bit adder
AUTO	448695	3314611	2214.2	179.2	3D FE Mesh
BBMAT	38744	993481	474.2	25.5	2D Stiffness M.
FINAN512	74752	261120	311.0	18.0	Lin. Prog.
LHR10	10672	209093	142.6	8.1	Chem. Eng.
MAP1	267241	334931	850.2	44.8	Highway Net.
MEMPLUS	17758	54196	117.9	4.3	Memory circuit
SHYY161	76480	152002	130.0	10.1	Navier-Stokes
TORSO	201142	1479989	1053.4	63.9	3D FE Mesh

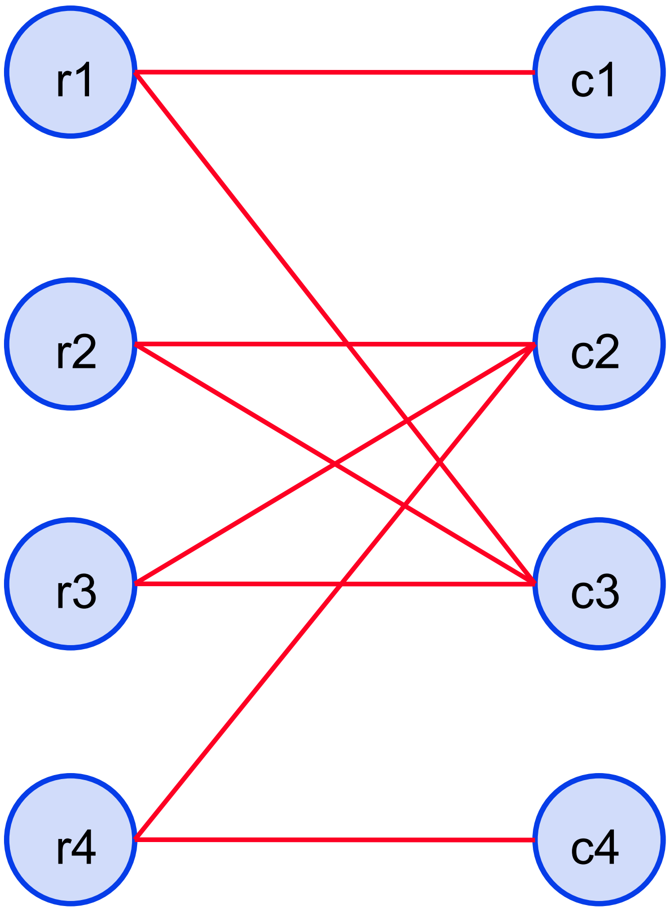
Kernighan/Lin much faster than Spectral Bisection!

Content

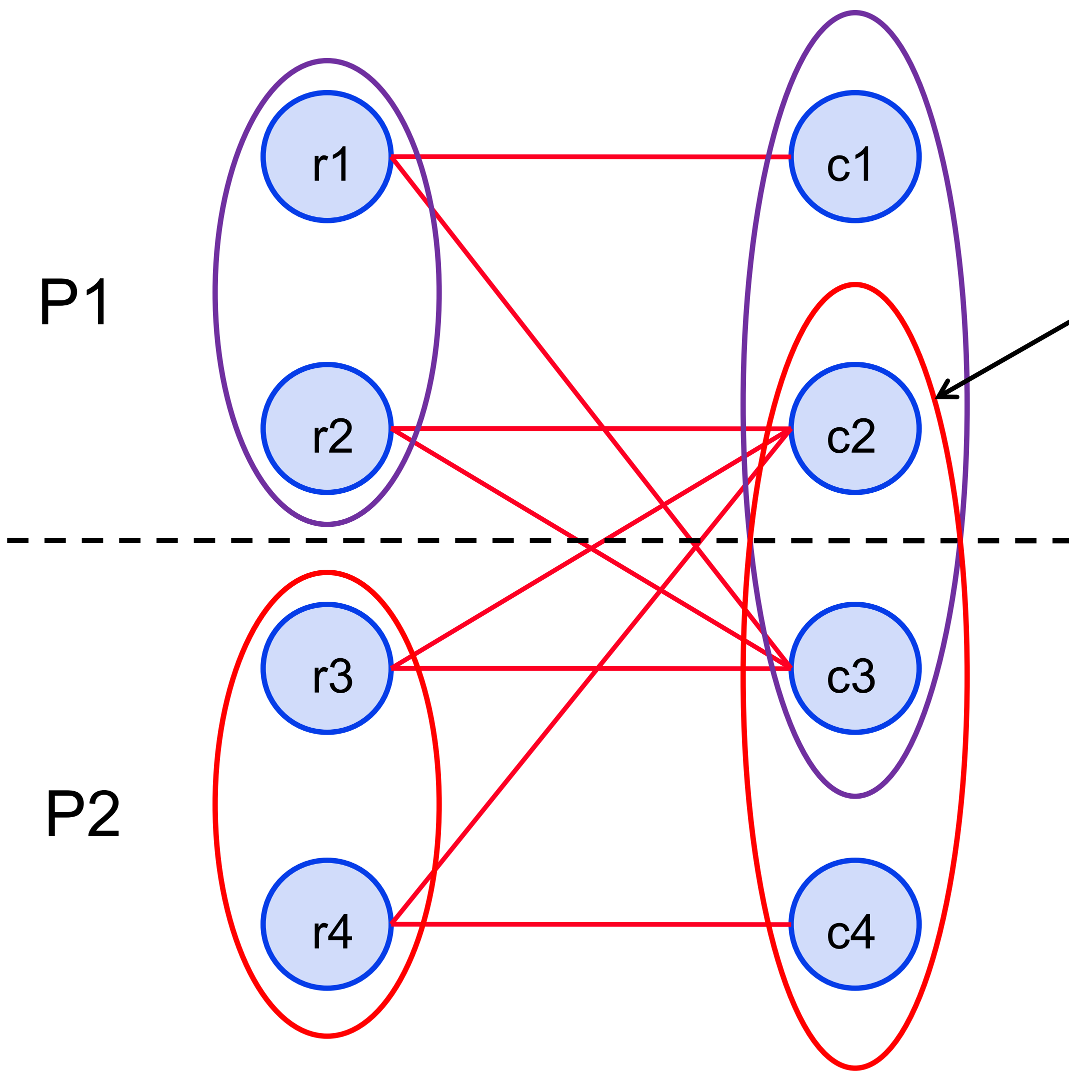
- Motivation for graph partitioning
- Overview of heuristics
- Partitioning <u>with</u> nodal coordinates
  - Ex: In finite element models, node at point in (x, y, z) space
    Recursive Coordinate Bisection
    Inertial Partitioning
- Partitioning <u>without</u> nodal coordinates
  - Ex: In model of WWW, nodes are web pages
    Fiduccia-Matteyes
    Spectral Methods
- Multilevel Acceleration
  - BIG IDEA, appears often in scientific computing
- Available Implementations
- Beyond Graph Partitioning: Hypergraphs

Beyond simple graph partitioning: Representing a sparse matrix as a hypergraph

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



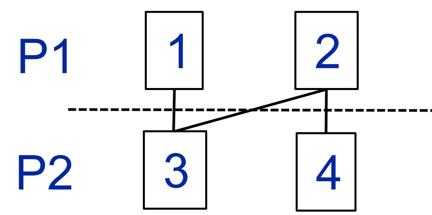
Beyond simple graph partitioning: Representing a sparse matrix as a hypergraph



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

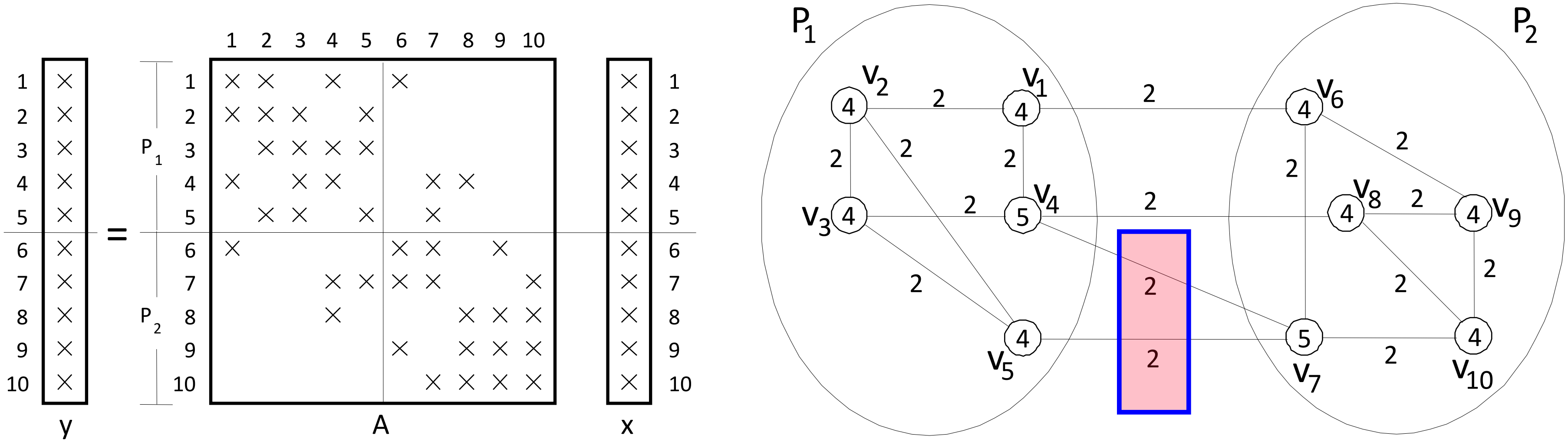
$$\begin{bmatrix} \times & 0 & \times & 0 \\ 0 & \times & \times & 0 \\ 0 & \times & \times & 0 \\ 0 & \times & 0 & \times \end{bmatrix}$$

But graph cut is 3!



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

A sparse matrix in the graph model



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

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

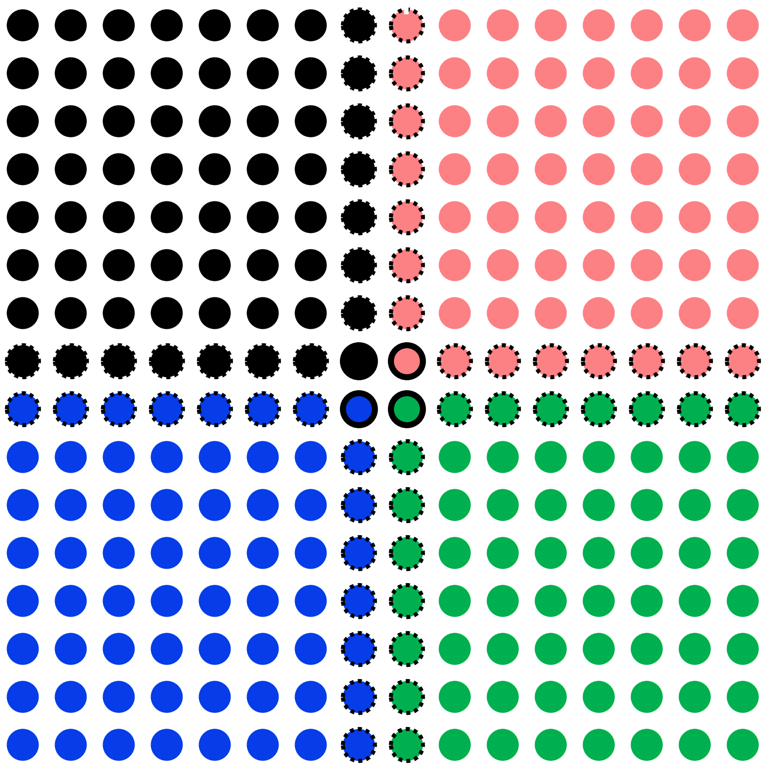
P_1 performs: $y(4) \leftarrow y(4) + A(4,7)\, x(7)$ and

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

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

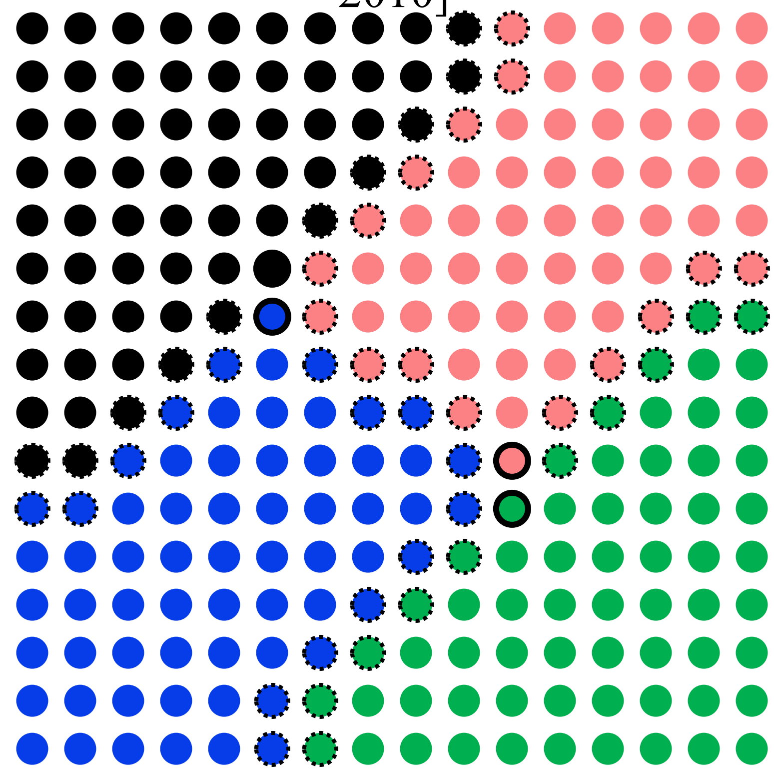
Two Different 2D Mesh Partitioning Strategies

Graph:
Cartesian Partitioning



Total SpMV communication volume = 64

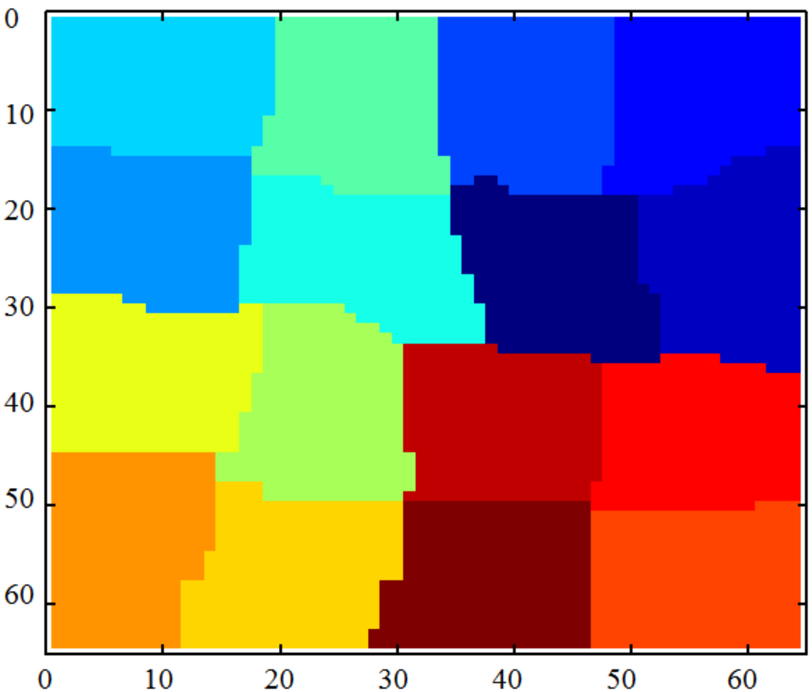
Hypergraph:
MeshPart Algorithm [Ucar, Catalyurek, 2010]



Total SpMV communication volume = 58

Experimental Results: Hypergraph vs. Graph Partitioning

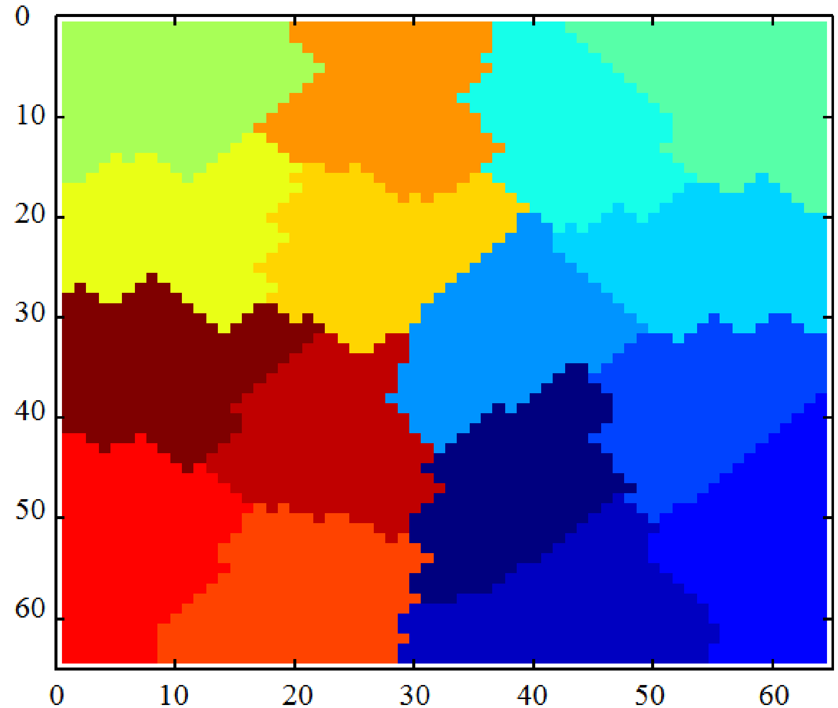
64x64 Mesh (5-pt stencil), 16 processors



Graph Partitioning (Metis)

Total Comm. Vol = 777

Max Vol per Proc = 69



Hypergraph Partitioning (PaToH)

Total Comm. Vol = 719

Max Vol per Proc = 59

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

Coordinate-Free — Summary

- Several techniques for partitioning **without coordinates**
  - Fiduccia-Matteyes – good corrector given reasonable partition
  - Spectral Method – good partitions, but slow
- **Multilevel methods**
  - Used to speed up problems that are too large/slow
  - Can be used with FM and Spectral methods and others
- **Speed/quality**
  - For load balancing of grids, multi-level FM probably best
  - For other partitioning problems (www, circuit, …) spectral may be better
  - Good software available: (METIS, KaHyPar)

Coordinate-Free – Local/Global Partitioning Algorithms

- Fiduccia-Matteyes are **local** partitioning algorithms:
  - both need an initial partitioning.
  - improve partitioning based on local partitioning decisions.

- Multi-level methods and spectral methods are **global** partitioning methods based on information that take all edges into account.

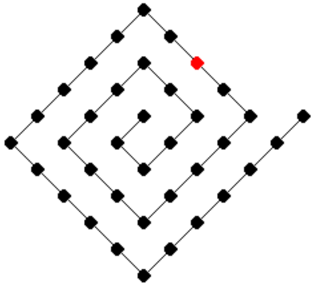
The combination of both methods typically form a very efficient partitioning method.

Appendix

Example 1:

- $|V_s| <= |E_s|$
- $|E_s| <= d * |V_s|$ where d is the maximum degree of the graph

Example (graph that is not spatial connected)



Expected # edge cuts for 64-way partition of 2D mesh of n nodes?

$n^{1/2} + 2*(n/2)^{1/2} + 4*(n/4)^{1/2} + \ldots + 32*(n/32)^{1/2}$ **$\sim 17 * n^{1/2}$**

Expected # edge cuts for 64-way partition of 3D mesh of n nodes?

$n^{2/3} + 2*(n/2)^{2/3} + 4*(n/4)^{2/3} + \ldots + 32*(n/32)^{2/3}$ **$\sim 11.5 * n^{2/3}$**

Inertial Partitioning: $M\, u = \lambda\, u \leftrightarrow u^T M\, u = u^T \lambda\, u = \lambda\, u^T u = \lambda$