MCS/MINF/MAI Master Course – High-Performance Computing

Parallel Graph-Partitioning on HPC Architectures

Olaf Schenk
Institute of Computing
USI Lugano, Switzerland
November 27, 2024

Content

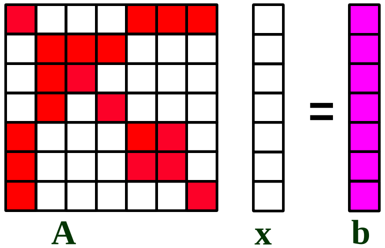
- Motivation for graph partitioning
- Overview of heuristics
- Partitioning with nodal coordinates
  - Ex: In finite element models, node at point in (x, y, z) space
    Recursive Coordinate Bisection
    Inertial Partitioning
- Partitioning without nodal coordinates
  - Ex: In model of WWW, nodes are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

    Fiduccia-Matteyes

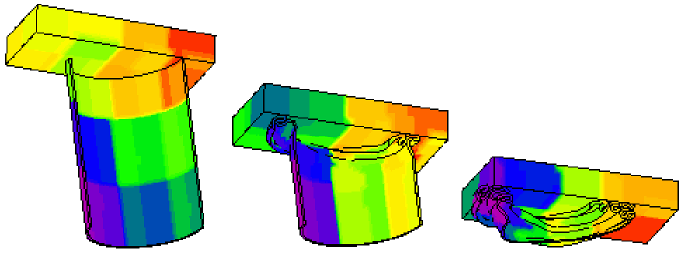
    Spectral Methods
- Multilevel acceleration
  - **BIG IDEA**, appears often in scientific computing
- Available implementations
- Beyond Graph Partitioning: Hypergraphs

Partitioning and Load Balancing

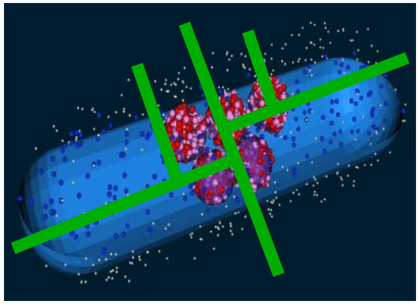
- **Goal:** assign data to processors to
  - minimize parallel application runtime
  - maximize utilization of computing resources
- **Metrics:**
  - minimize processor idle time (balance workloads)
  - keep inter-processor communication costs low
- Impacts performance of a wide range of simulations



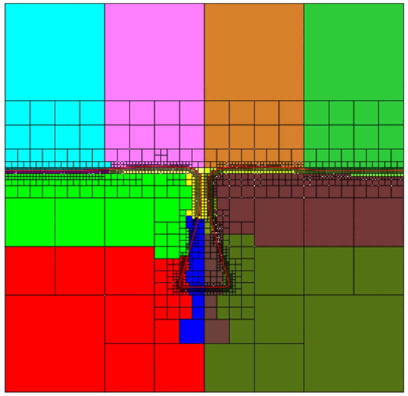
Linear solvers & preconditioners



Contact detection



Particle simulations

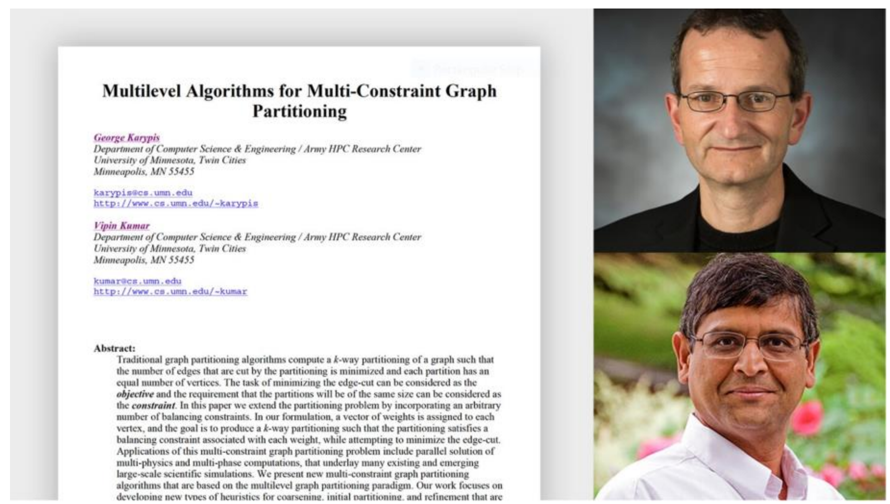


Adaptive mesh refinement

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)
- <u>Goal</u>: Evenly distribute vertex weight while minimizing weight of cut edges.

- **Excellent software available**
  - METIS (U. Minnesota)



Multilevel Algorithms for Multi-Constraint Graph Partitioning

George Karypis
Department of Computer Science & Engineering / Army HPC Research Center
University of Minnesota, Twin Cities
Minneapolis, MN 55455

karypis@cs.umn.edu
http://www.cs.umn.edu/~karypis

Vipin Kumar
Department of Computer Science & Engineering / Army HPC Research Center
University of Minnesota, Twin Cities
Minneapolis, MN 55455

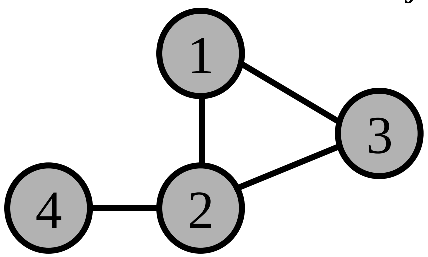
kumar@cs.umn.edu
http://www.cs.umn.edu/~kumar

Abstract:
Traditional graph partitioning algorithms compute a k-way partitioning of a graph such that the number of edges that are cut by the partitioning is minimized and each partition has an equal number of vertices. The task of minimizing the edge-cut can be considered as the ***objective*** and the requirement that the partitions will be of the same size can be considered as the ***constraint***. In this paper we extend the partitioning problem by incorporating an arbitrary number of balancing constraints. In our formulation, a vector of weights is assigned to each vertex, and the goal is to produce a k-way partitioning such that the partitioning satisfies a balancing constraint associated with each weight, while attempting to minimize the edge-cut. Applications of this multi-constraint graph partitioning problem include parallel solution of multi-physics and multi-phase computations, that underlay many existing and emerging large-scale scientific simulations. We present new multi-constraint graph partitioning algorithms that are based on the multilevel graph partitioning paradigm. Our work focuses on developing new types of heuristics for coarsening, initial partitioning, and refinement that are

George Karypis (top), an Amazon senior principal scientist, and Vipin Kumar, a University of Minnesota professor, have been awarded the SC21 Test of Time Award for a 1998 paper they coauthored which presented algorithms that have subsequently been applied in diverse application domains, from electronic design automation tools for field programmable gate arrays, to determining state-level electoral districts in the United States.

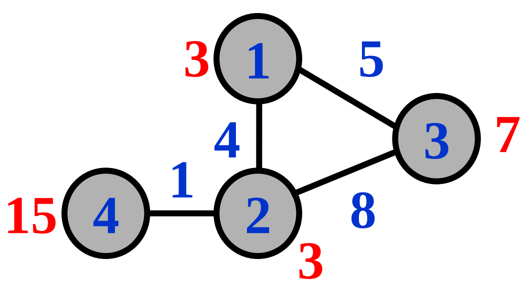
Definition of Graph

- 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“).

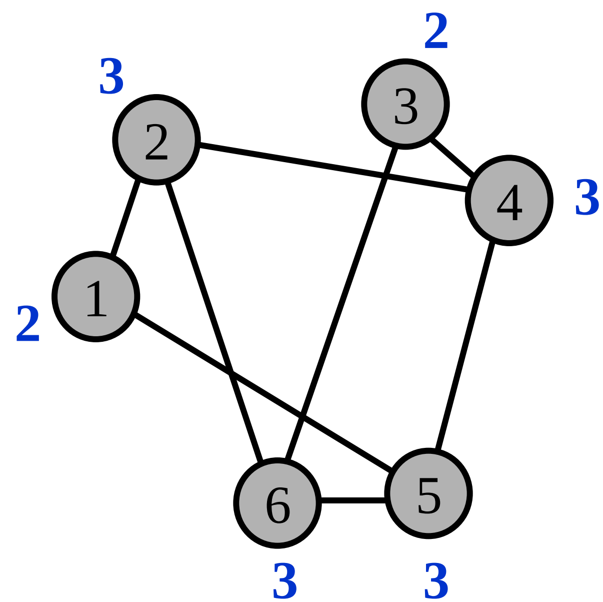


$W_v = \{ 3, 3, 7, 15\}$, $W_e = \{ 4, 5, 8, 1\}$

Examples for Graphs

- Symmetric sparse matrix and Graph G_A

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



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

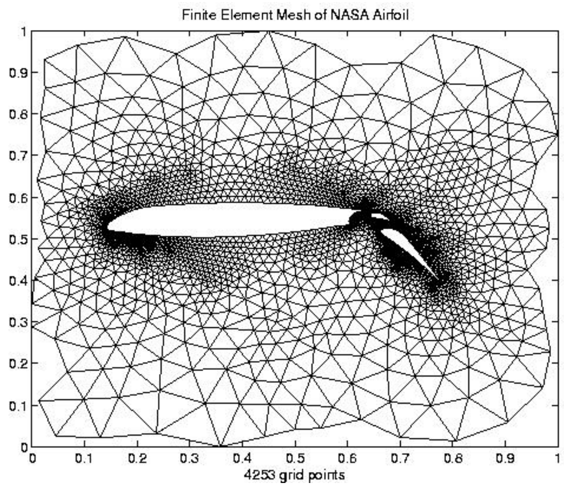
  $E = \{ (1,2), (1,5), (2,4), (2,6), (3,4), (3,6), (4,5), (5,6) \}$

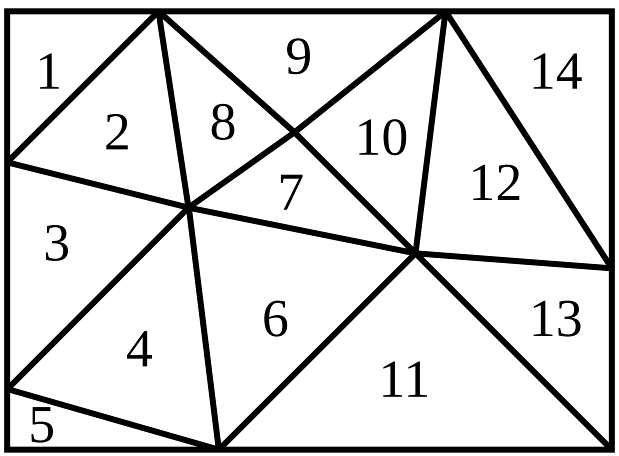
  $W_v = \{2, 3, 2, 3, 3, 3\}$ e.g. numbers of nonzeros in each row

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

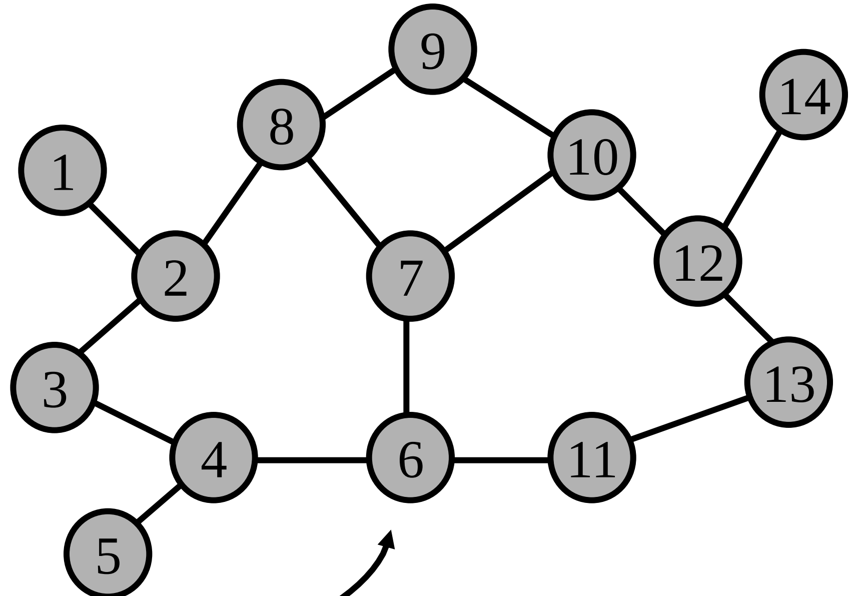
Examples for Graphs

- Finite-Element Simulations





Finite-Element Mesh



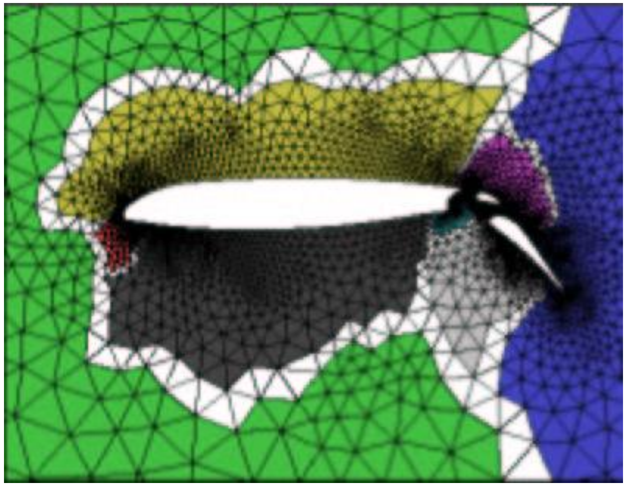
$G_{FE} = (V, E)$, $V = \{1, \ldots, 14\}$

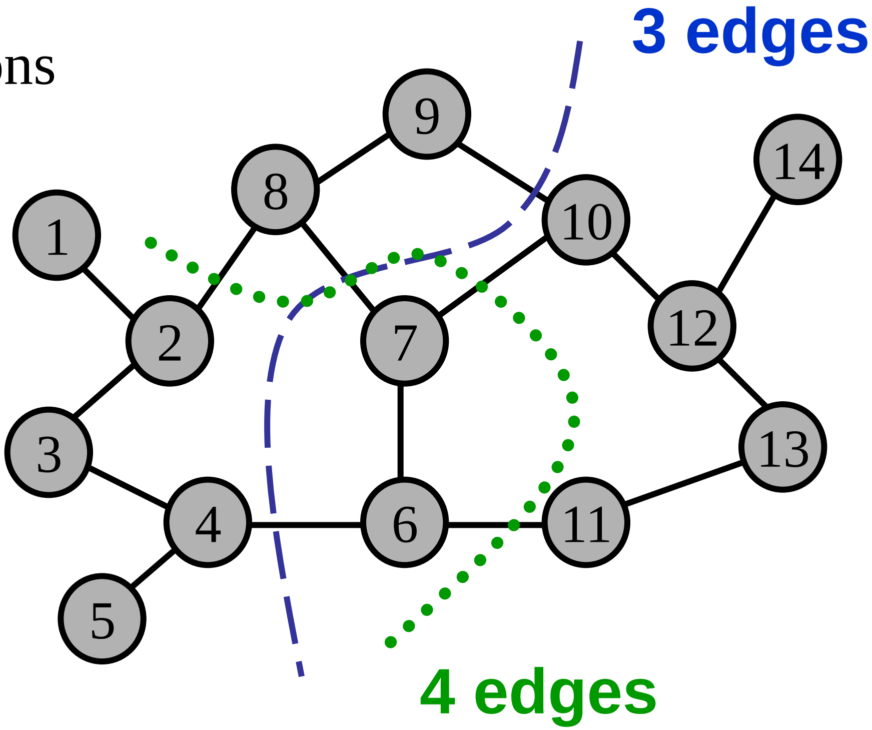
$E = \{(1,2), \ldots, (12,14)\}$

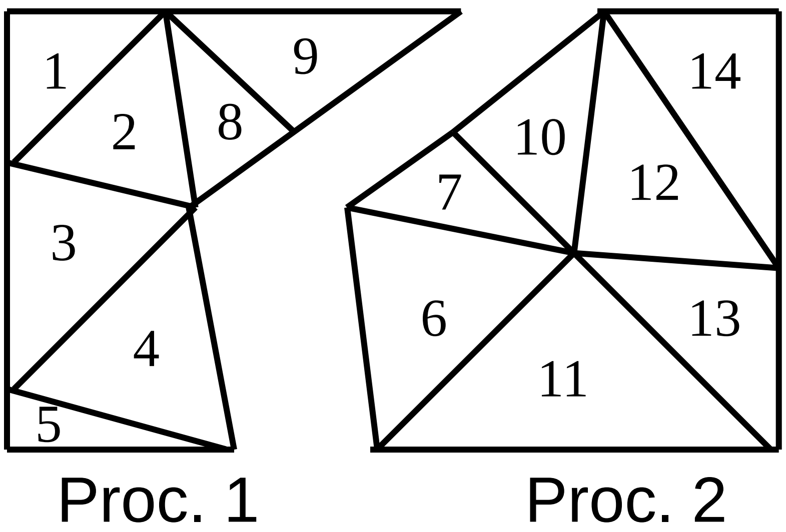
$W_e \equiv 1$, $W_v \equiv 1$

Examples for Graph Partitioning

- Parallel Finite-Element Simulations



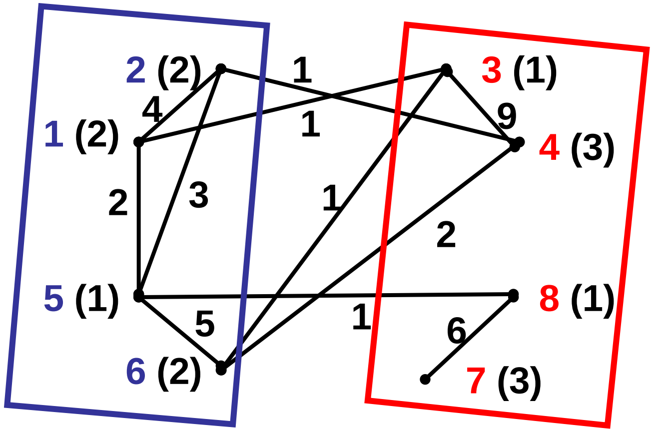




A good partitioning G_{FE} results in
- equal #elements/processor
  („load“ and „storage balancing“).
- Minimal #edges between P1 and P2
  (minimal communication volume).

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"

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

The sum of edge connecting pairs V_1 and V_2 is minimized

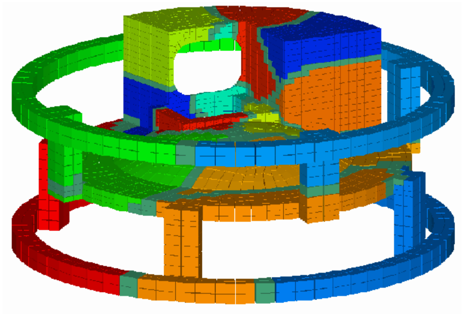
$$\min |\{e_{IJ} \in E \mid v_i \in V_1 \text{ und } v_j \in V_2\}|$$

Heuristics — Overview of Bisection Algorithms

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

  Algorithms: **Recursive Coordinate Bisection**

  Inertial Partitioning



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

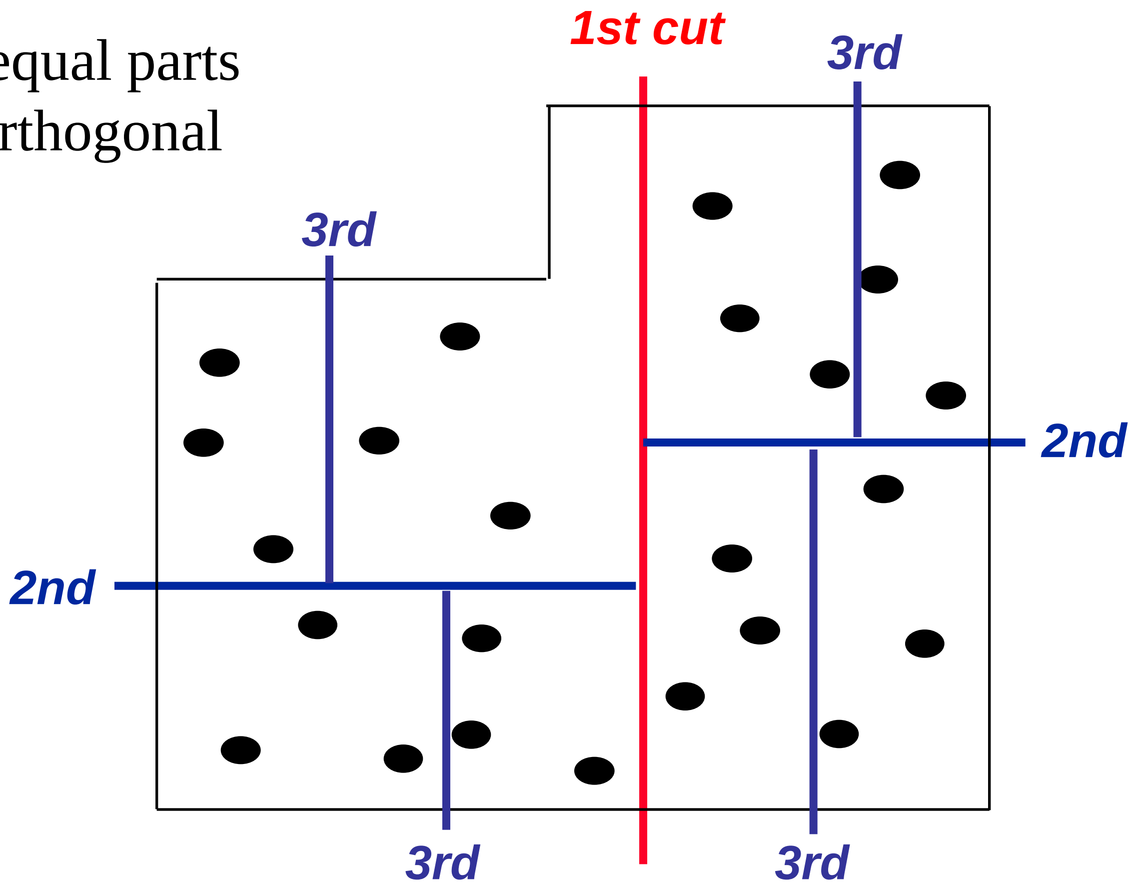
  Algorithms: **Fiduccia-Matteyes**

  Spectral Methods

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

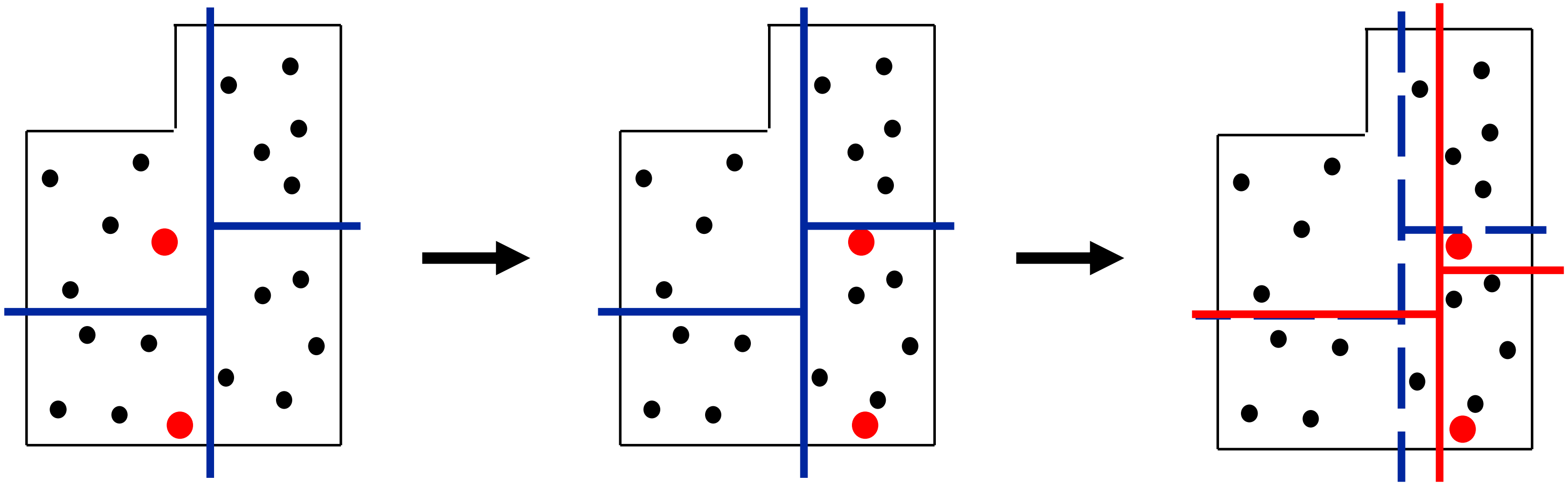
Nodal Coordinates — Recursive Coordinate Bisection (RCB)

- 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.



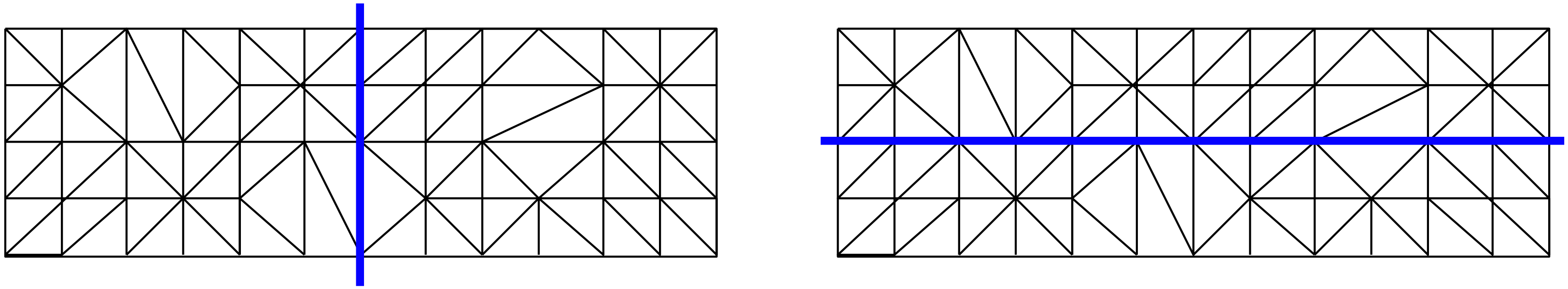
Nodal Coordinates — RCB Advantages

- 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.

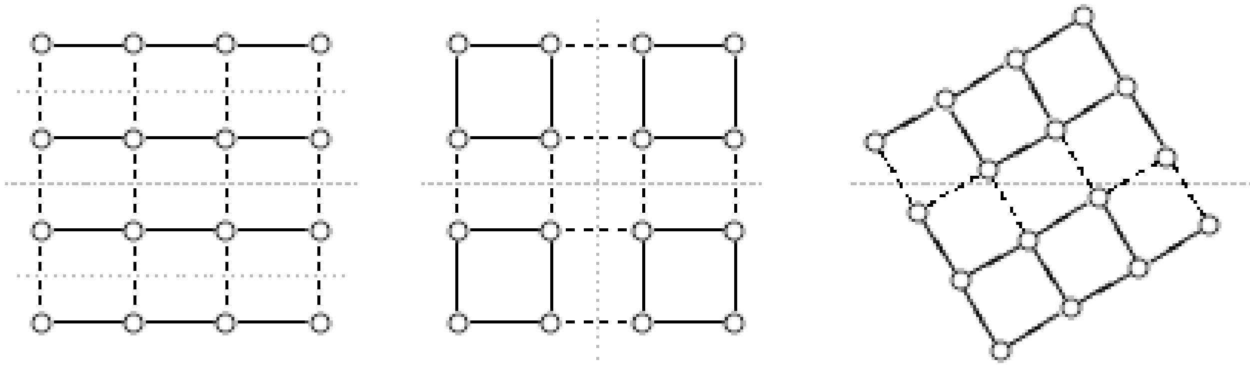


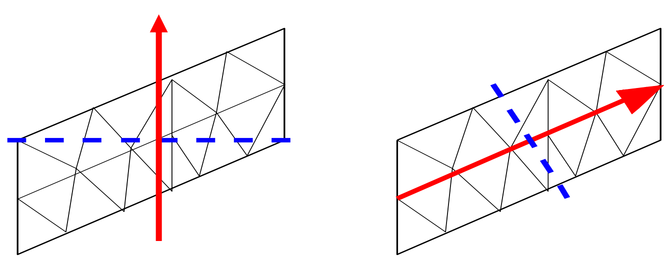
Nodal Coordinates — Coordinate Bisection

- Partition the domain along hyperplanes with node coordinates



- Change the coordinate systems

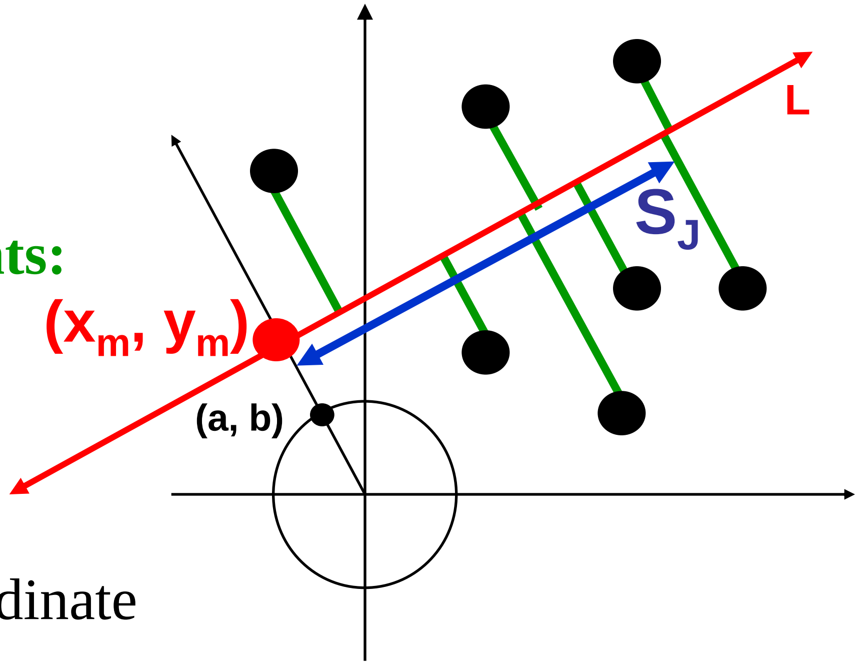




Good choice of coordinate system leads to inertial bisection

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

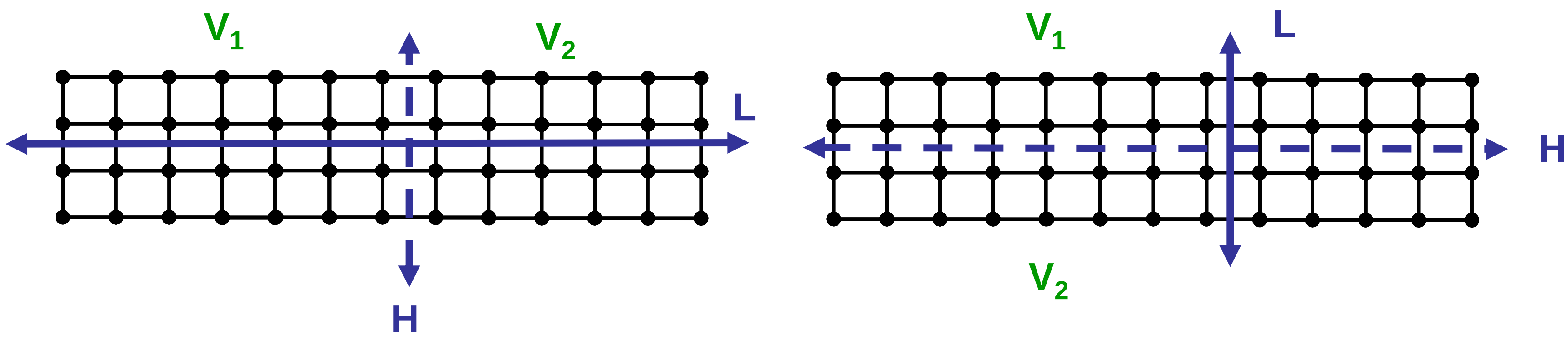
- Let $S_k = \text{median}(S_1,\ldots, S_n)$

(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

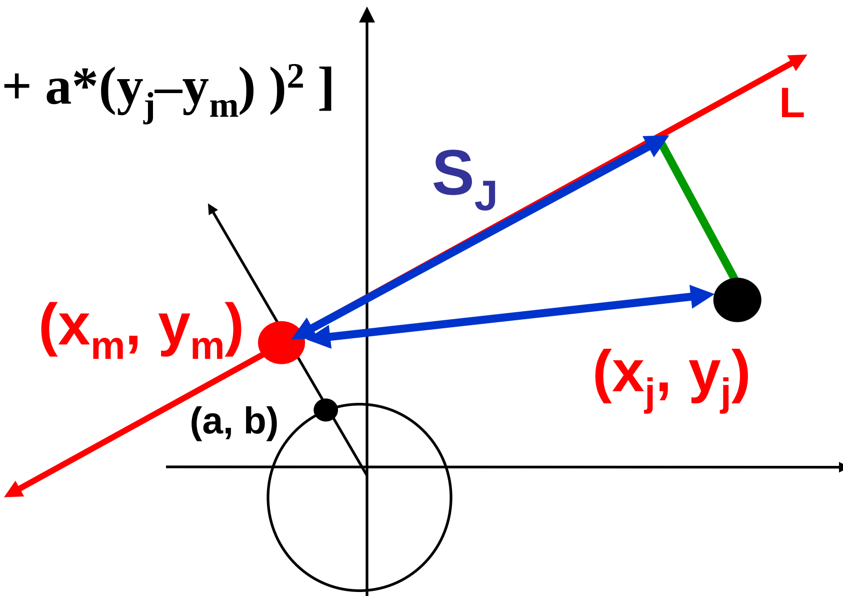
- Clearly prefer L on left below



- 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



$$= (1 - b^2) * \sum_j (x_j - x_m)^2 + 2*a*b* \sum_j (x_j - x_m)*(y_j - y_m) + (1-a^2) \sum_j (y_j - y_m)^2$$
$$= a^2 * \sum_j (x_j - x_m)^2 + 2*a*b* \sum_j (x_j - x_m)*(y_j - y_m) + b^2 \sum_j (y_j - y_m)^2$$
$$= a^2 * \quad X_1 \quad + 2*a*b* \quad X_2 \quad + b^2 * \quad X_3$$
$$= \begin{vmatrix} a & b \end{vmatrix} * \begin{vmatrix} X1 & X2 \\ X2 & X3 \end{vmatrix} * \begin{vmatrix} a \\ b \end{vmatrix} = \underline{\text{minimum}} = \lambda$$

$M\,u = \lambda\,u \leftrightarrow$ Minimizing λ ?

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

Minimized by choosing

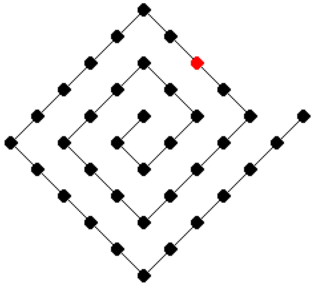
$(x_m , y_m) = (\sum_j x_j , \sum_j y_j) / n$ = center of mass

(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

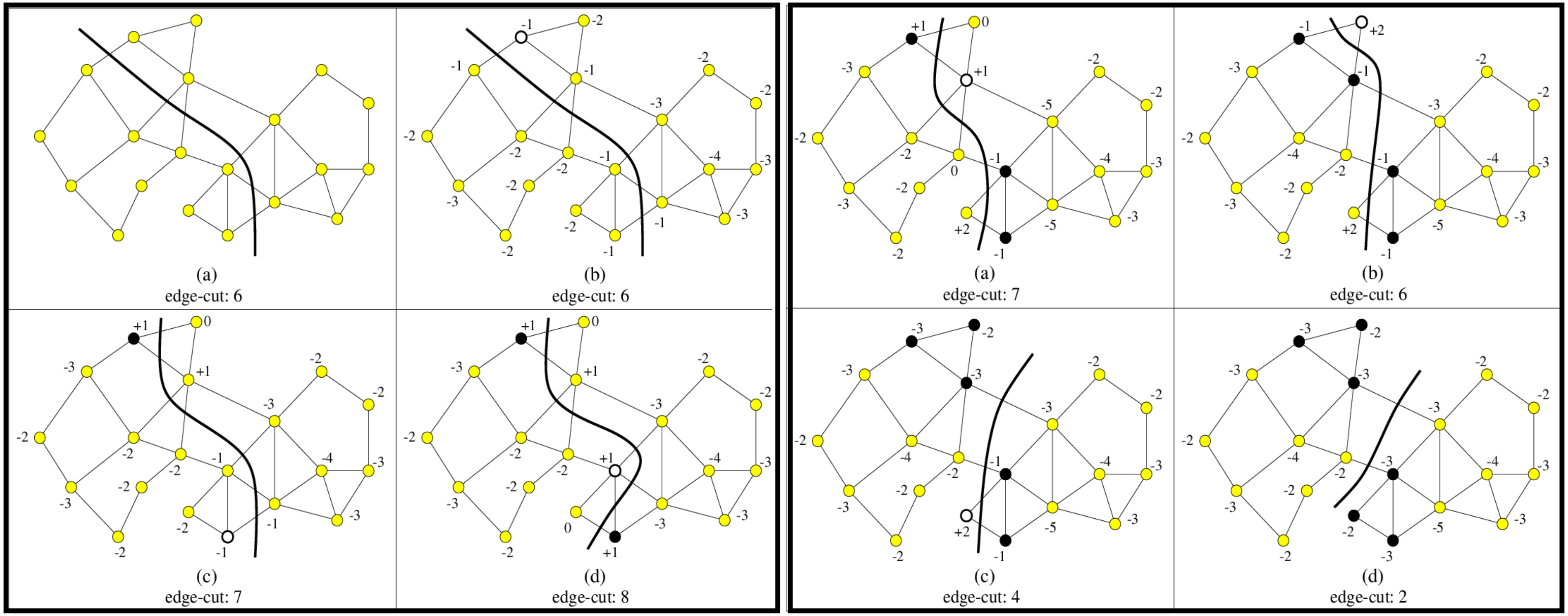
- Motivation for graph partitioning
- Overview of heuristics
- Partitioning with nodal coordinates
  - Ex: In finite element models, node at point in (x, y, z) space
    Recursive Coordinate Bisection
    Inertial Partitioning
- **Partitioning without nodal coordinates**
  - **Ex: In model of WWW, nodes 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)

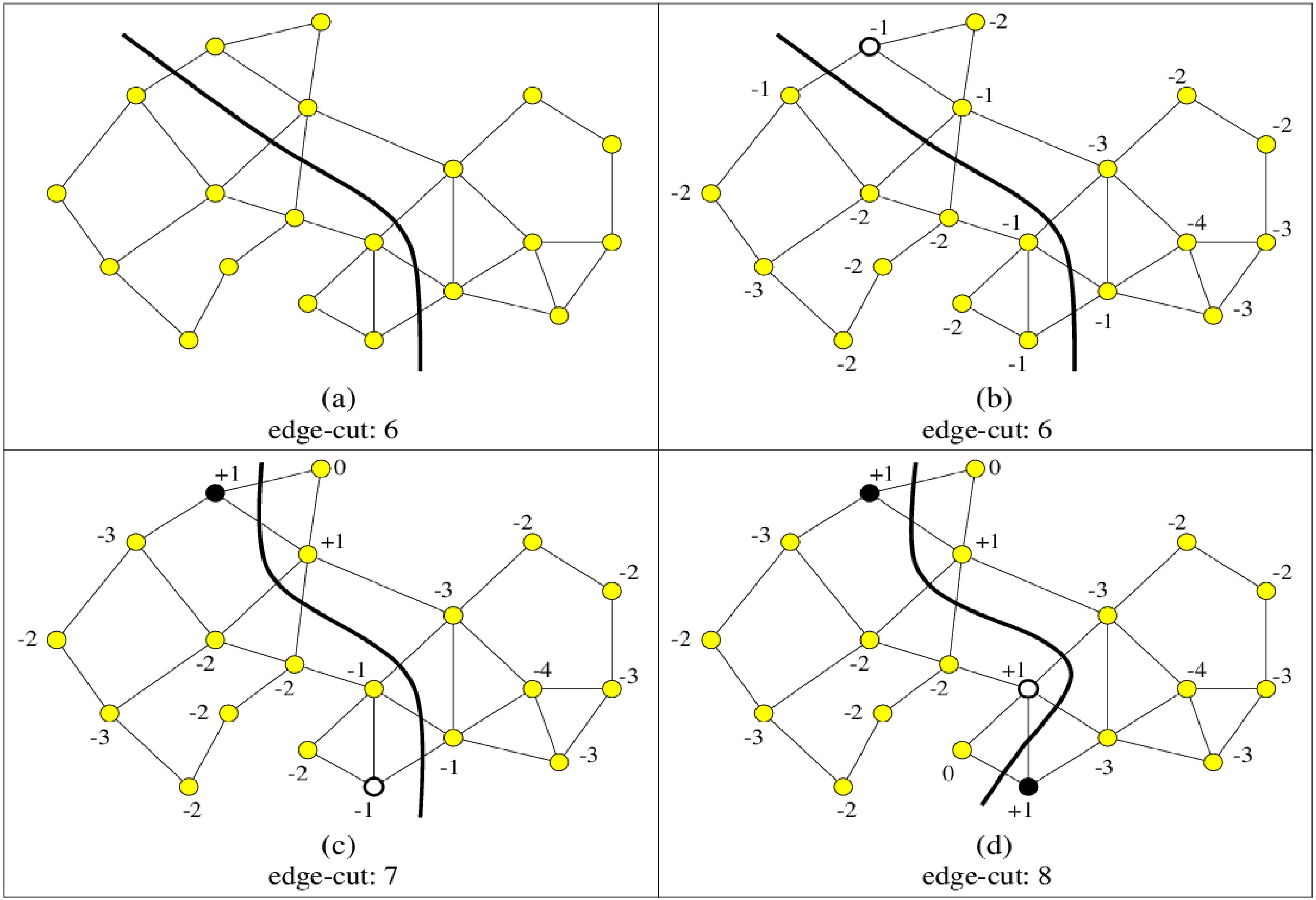
Coordinate-Free — Fiducca-Matteyes Algorithms (F/M)

- **Example:**

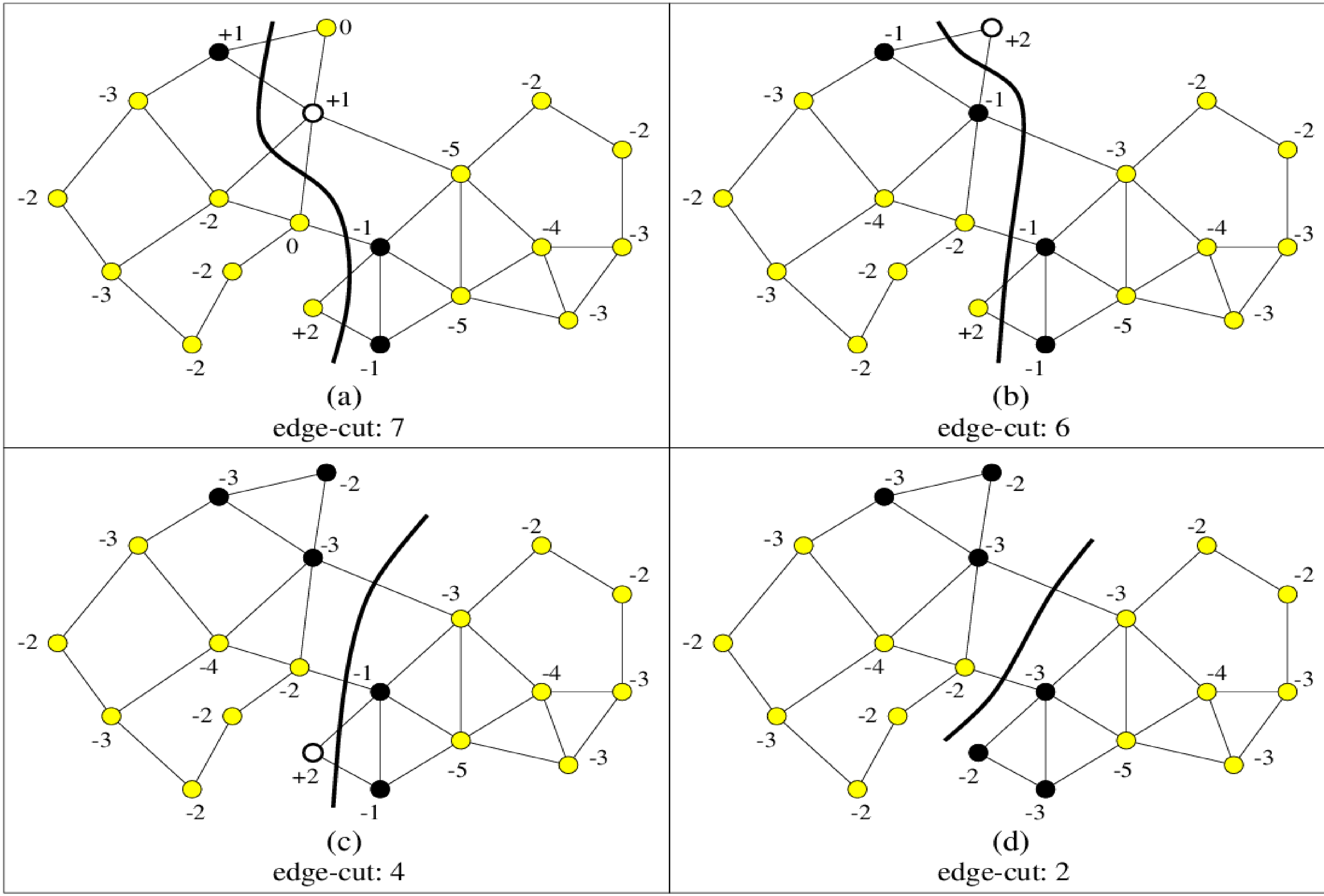


- All vertices v are marked with D(v) — Reduction of edge-cut.
- Load imbalance: 10% - therefore in iteration (c) the vertex marked with D(v) = +1 can not move into other domain.

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



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



(a) edge-cut: 7

(b) edge-cut: 6

(c) edge-cut: 4

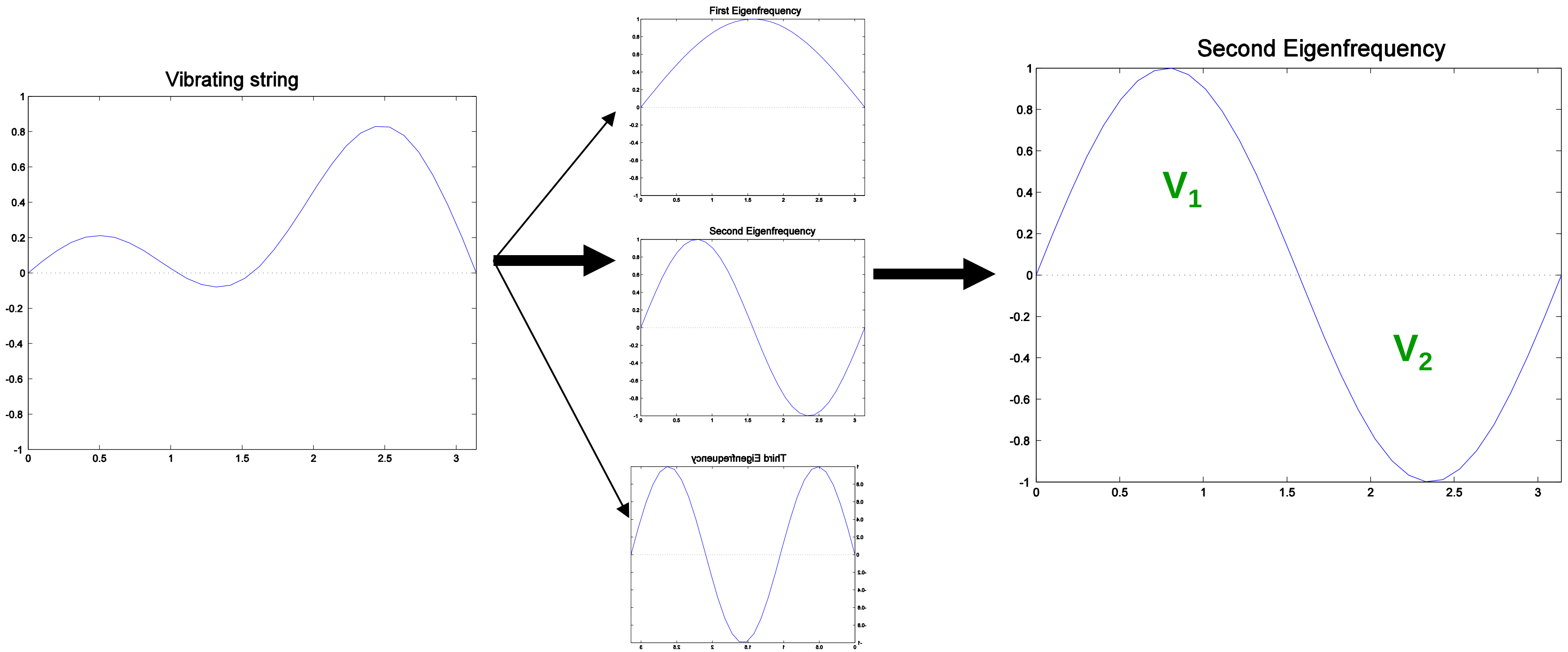
(d) edge-cut: 2

Coordinate-Free — Spectral Methods

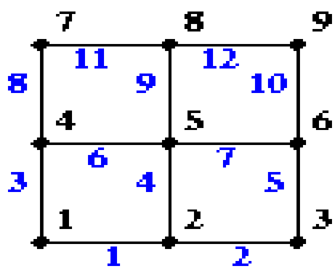
- **Spectral methods** as an example for **global partitioning** algorithms
- Heavily use of Eigenvalue/Eigenvector analysis

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



- **<u>Definition:</u>** 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



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

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, $\lambda_2 != 0$ 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

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

Relation between Laplace Matrix and Graph Partitioning

$$
\begin{aligned}
\boxed{x^T \cdot L(G) \cdot x} &= \sum_{i,\, j} L(G)_{(i,j)} \cdot x_i \cdot x_j \\
&= \sum_{i} degree(i) \\
&\quad + \sum_{i \neq j;\ i,\, j \in N^+} (-1) \cdot (+1) \cdot (+1) + \sum_{i \neq j;\ i,\, j \in N^-} (-1) \cdot (-1) \cdot (-1) \\
&\quad + \sum_{i \neq j;\ i \in N^+,\, j \in N^-} (-1) \cdot (+1) \cdot (-1) \\
&= \quad 2 \cdot \#\textbf{edges in G} \\
&\quad -2 \cdot (\#\textbf{edges connecting node in } N^+ \textbf{ to nodes in } N^+) \\
&\quad -2 \cdot (\#\textbf{edges connecting node in } N^- \textbf{ to nodes in } N^-) \\
&\quad +2 \cdot (\#\textbf{edges connecting node in } N^+ \textbf{ to nodes in } N^-) \\
&= \boxed{4 \cdot (\#\textbf{edges connecting node in } N^+ \textbf{ to nodes in } N^-)}
\end{aligned}
$$

Relation between Laplace Matrix and Graph Partitioning

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

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

or

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

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

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

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

Relation between Laplace Matrix and Graph Partitioning

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

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

- Minimum is at $z = \mathrm{sqrt}(n)*u_2$.
- **Spectral Bisection Algorithm**:
  - Compute eigenvector u_2 corresponding to $\lambda_2(L (G))$
  - 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

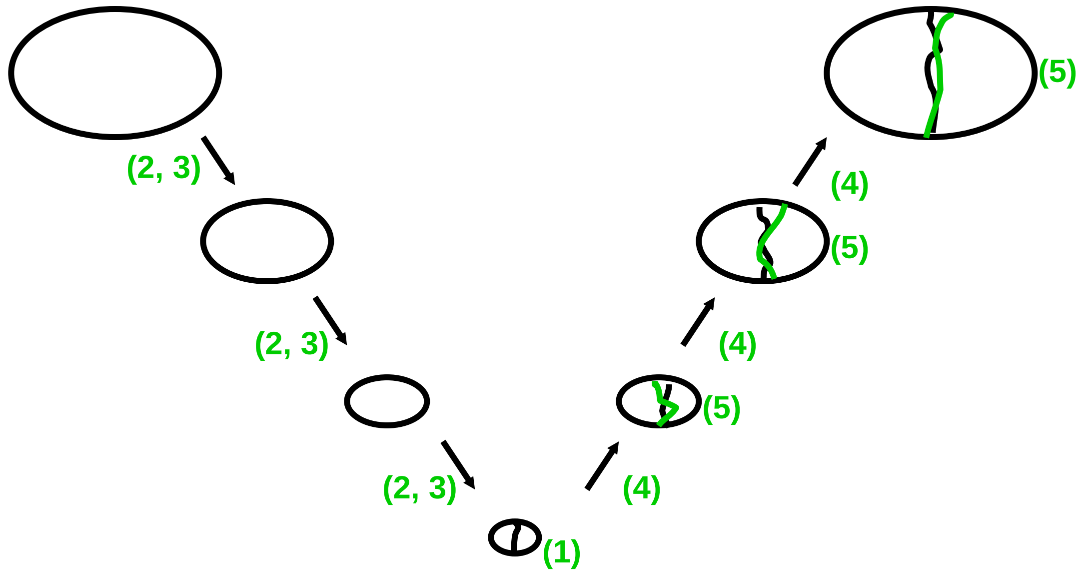
Multilevel Partitioning — Introduction

- 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?



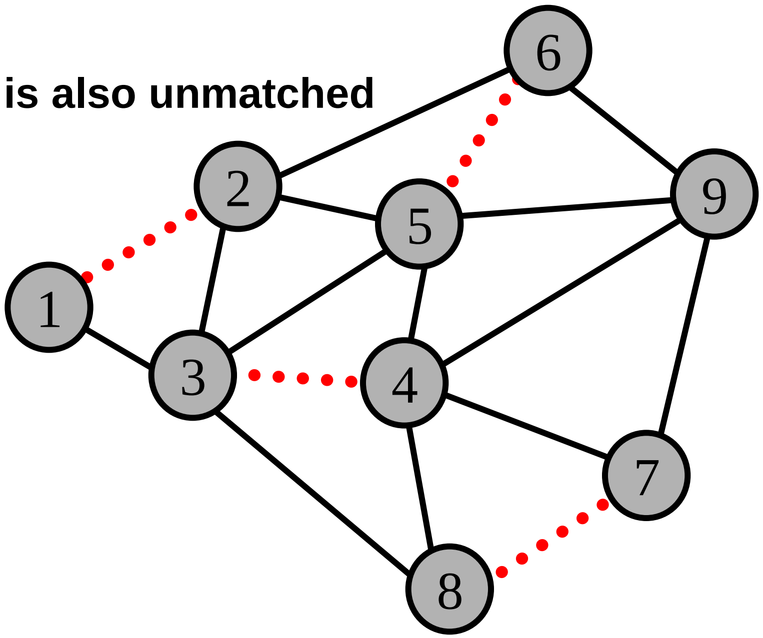
Multilevel Partitioning — Multilevel Fiduccia-Matteyes

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

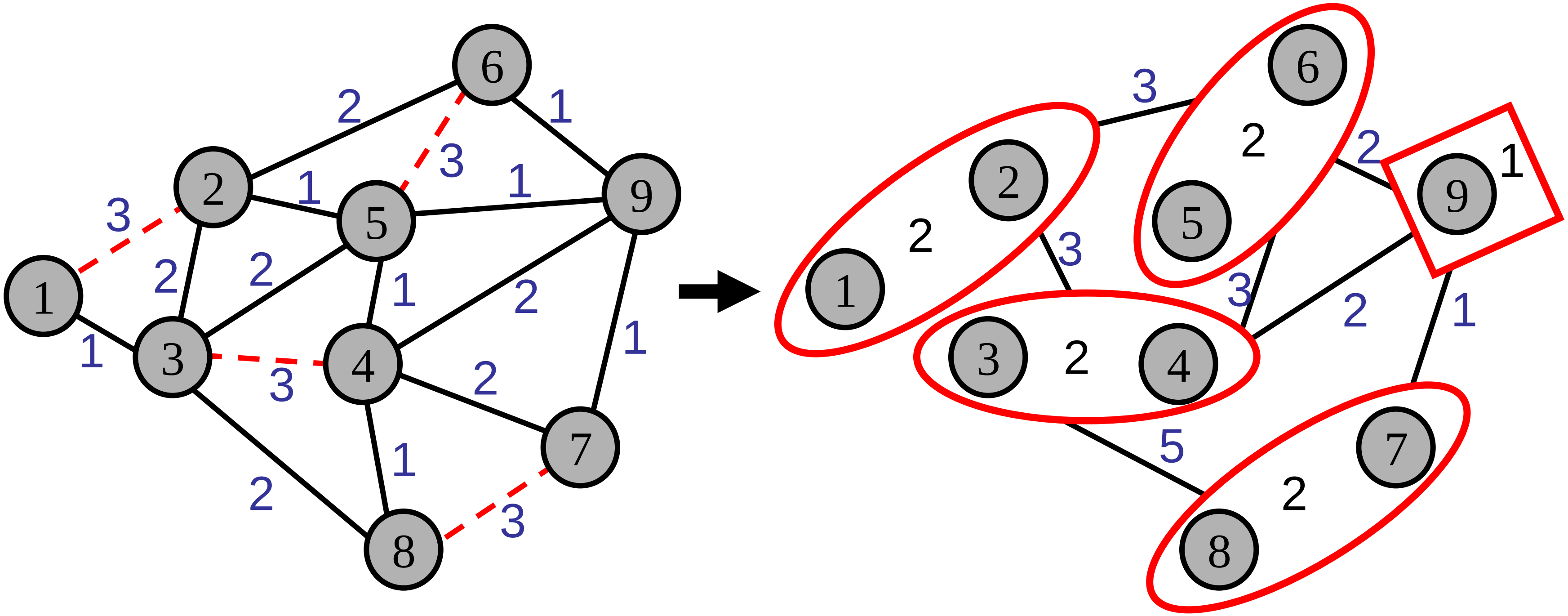
Multilevel Partitioning — Maximal Matching

- *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 Em 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 Em
             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)

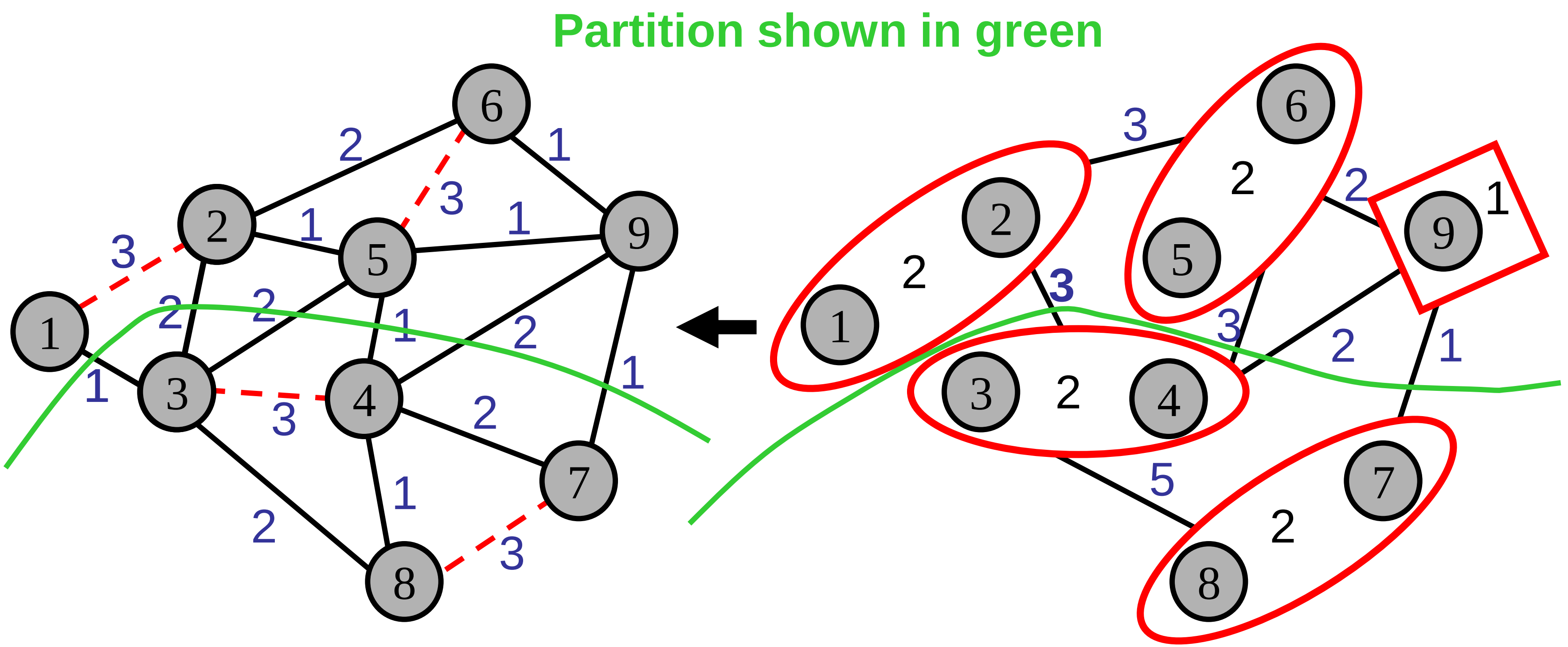
2) Collapse matched nodes into a single one
   for all edges e= (j, k) in E_m
      Put vertex v(e) in V_c
      W(v(e)) = W(j) + W(k)   // update vertex weights

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



$G = (V, E)$

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

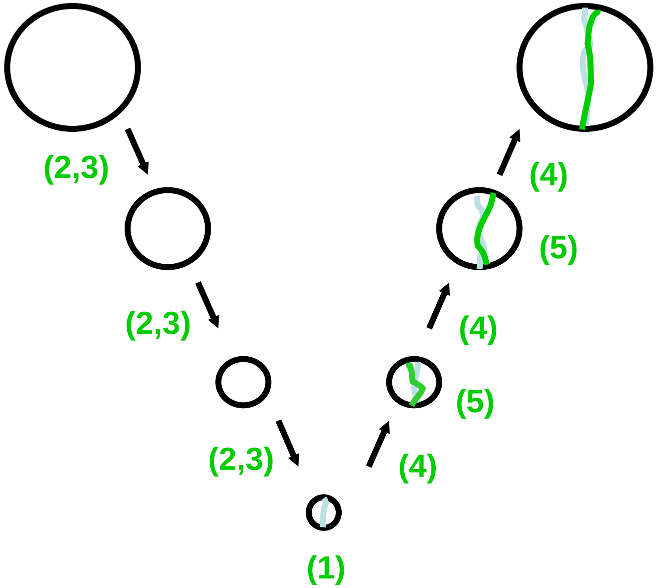
Vertex weights are black

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 ´ = 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?**



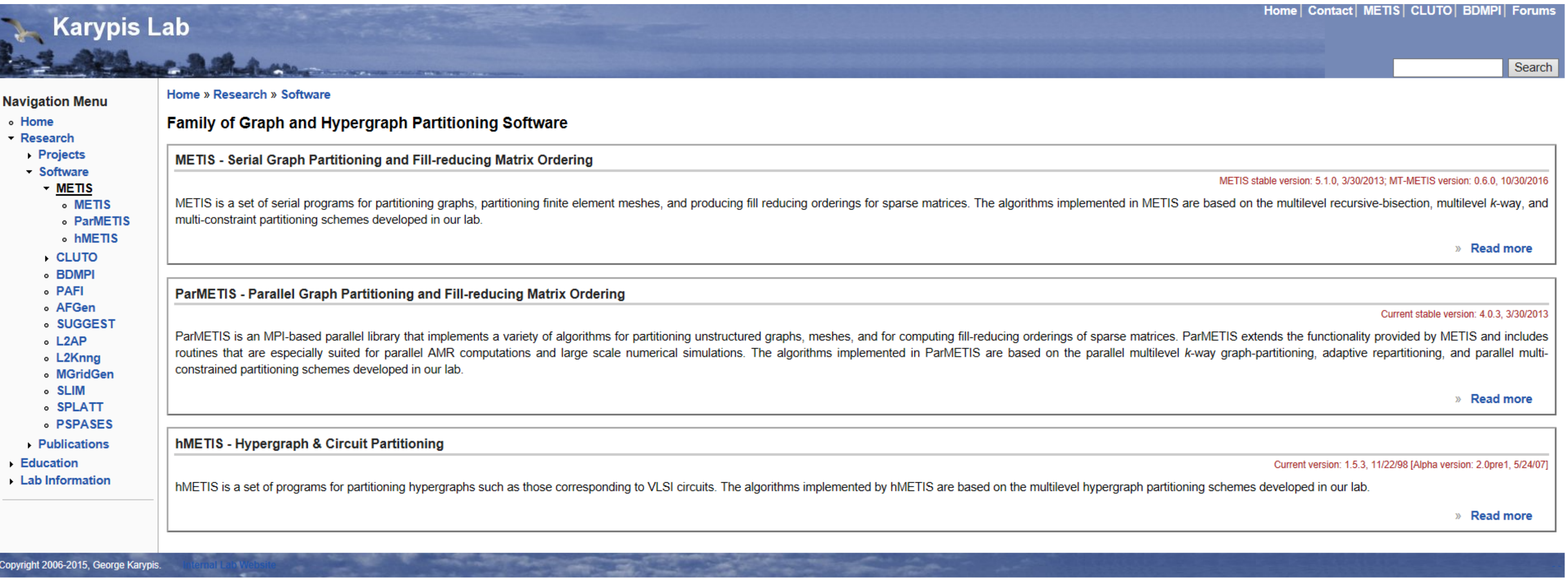
Content

- 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/)
- Matlab toolbox for geometric and spectral partitioning by Gilbert, Tang, and Li: https://github.com/YingzhouLi/meshpart
- Multilevel Spectral Bisection
  - S. Barnard and H. Simon, "A fast multilevel implementation of recursive spectral bisection …", 1993
  - Chaco (SC'14 Test of Time Award)
- Hybrids possible
  - Ex: Use Kernighan/Lin to improve a partition from spectral bisection
- Recent packages with collection of techniques
  - Zoltan (www.cs.sandia.gov/Zoltan)
  - KaHIP (http://algo2.iti.kit.edu/kahip/)

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

Copyright 2006-2015, George Karypis. Internal Lab Website

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

Demo – Partitioning in Matlab