High-**P**erformance **C**omputing 20**24**

Graph Partitioning - An Overview

Aryan Eftekhari, Olaf Schenk | **U**niversità della **S**vizzera **i**taliana (USI) | **I**nstitute of **C**omputing (CI)

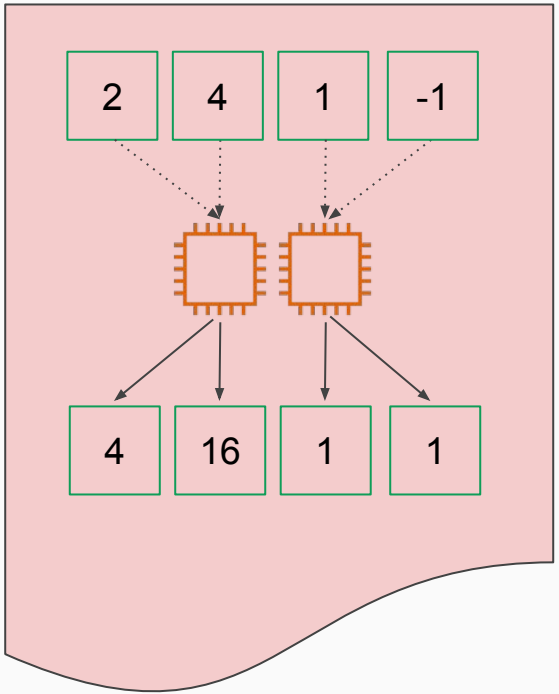
Distributed/Shared-Memory Parallelism

Parallelization paradigms

Shared Memory Parallelization

*Computation is distributed along **threads**.*

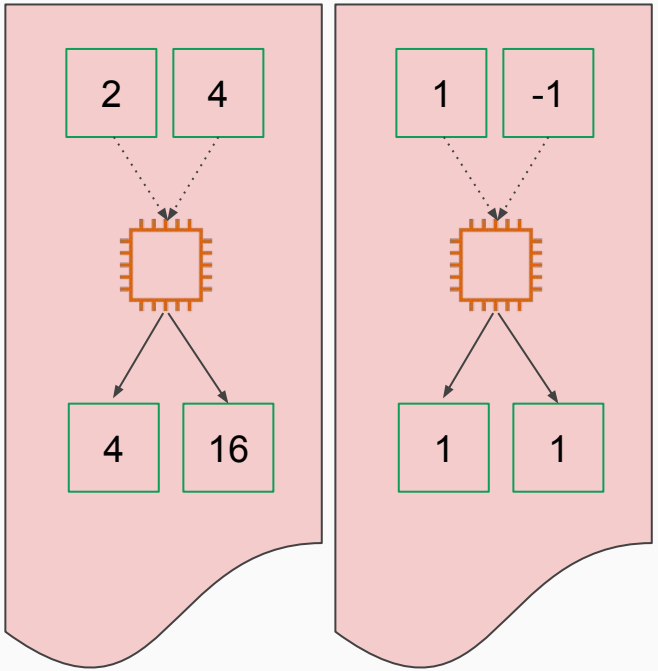
Synchronization between threads.



Distributed Memory Parallelization

*Computation is distributed along **processes**.*

Communication via message passing.



On to “Graph Partitioning”

Combinatorial matrix theory is a rich branch of mathematics that combines combinatorics, **graph theory**, and linear algebra.

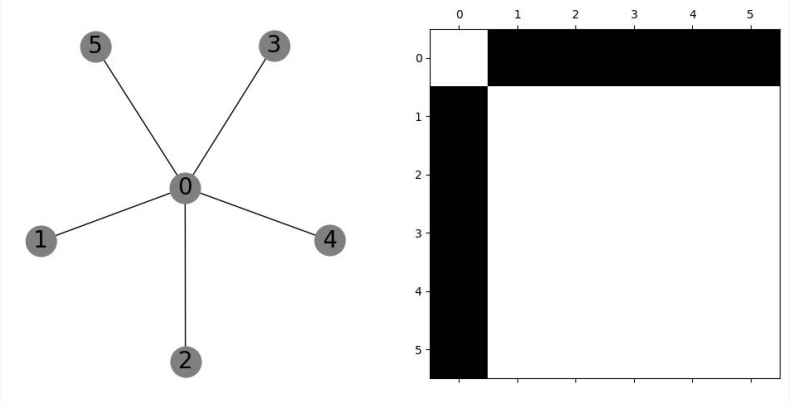
Motivation
What is a graph?

Simple undirected graph is defined as $\mathcal{G}=(\mathcal{V},\mathcal{E})$

- The vertices $\mathcal{V}$ are $\{1,2,\ldots n\}$.
- The edge $\{i,j\}\in\mathcal{E}$ connect an unordered pair of vertices.

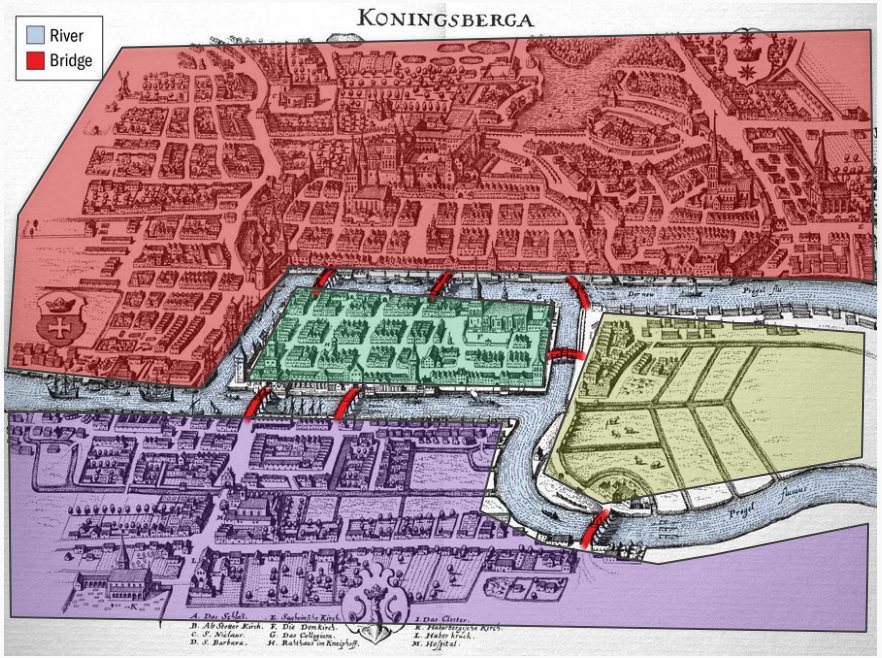
What if the connections between nodes differ in magnitude? (weighted)

What if $i\rightarrow j$ but not the other way? (*directed*)



Graphical Representation Adjacency Matrix

Motivation
What is a graph theory about?



Königsberg Bridge Problem

"... walk through the city cross each bridges once and only once"

Leonhard Euler: *Solutio problematis ad geometriam situs pertinentis [Link]*

This work is considered to be beginning of modern "Graph Theory".

Motivation
What is a graph partitioning?

"Bisection" to form two partitions …
Repeated for each partition for recursive partitions.

Desired attributes of bisection

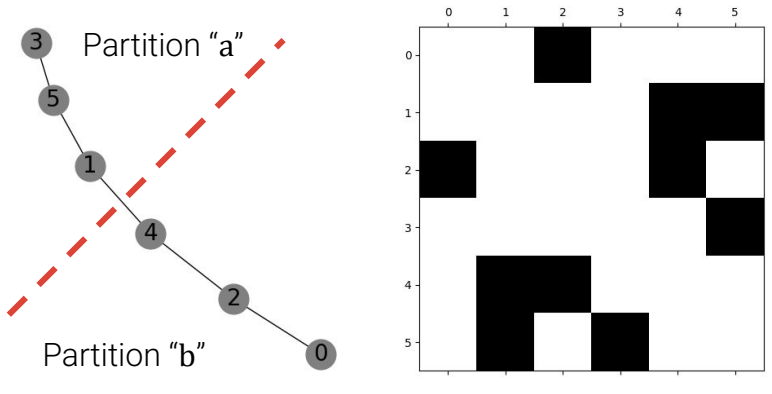
1) Make two partitions of (roughly) equal size:

$$\mathcal{V} = \mathcal{V}^a \cup \mathcal{V}^b \text{ st. } |\mathcal{V}^a| = |\mathcal{V}^b|$$

2) With the minimum edge-cut:

$$\min \{\{i,j\} \in \mathcal{E} : i \in \mathcal{V}^a \text{ and } j \in \mathcal{V}^b\}$$

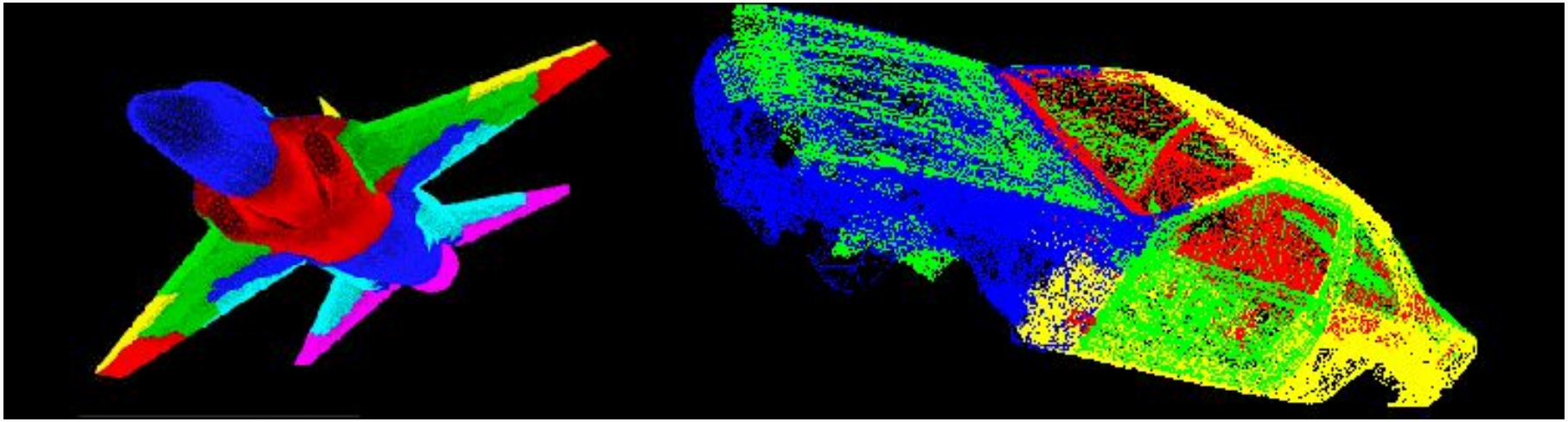
A good partition



Think of a bad or the worst partitioning?

Parallel computing and load balancing

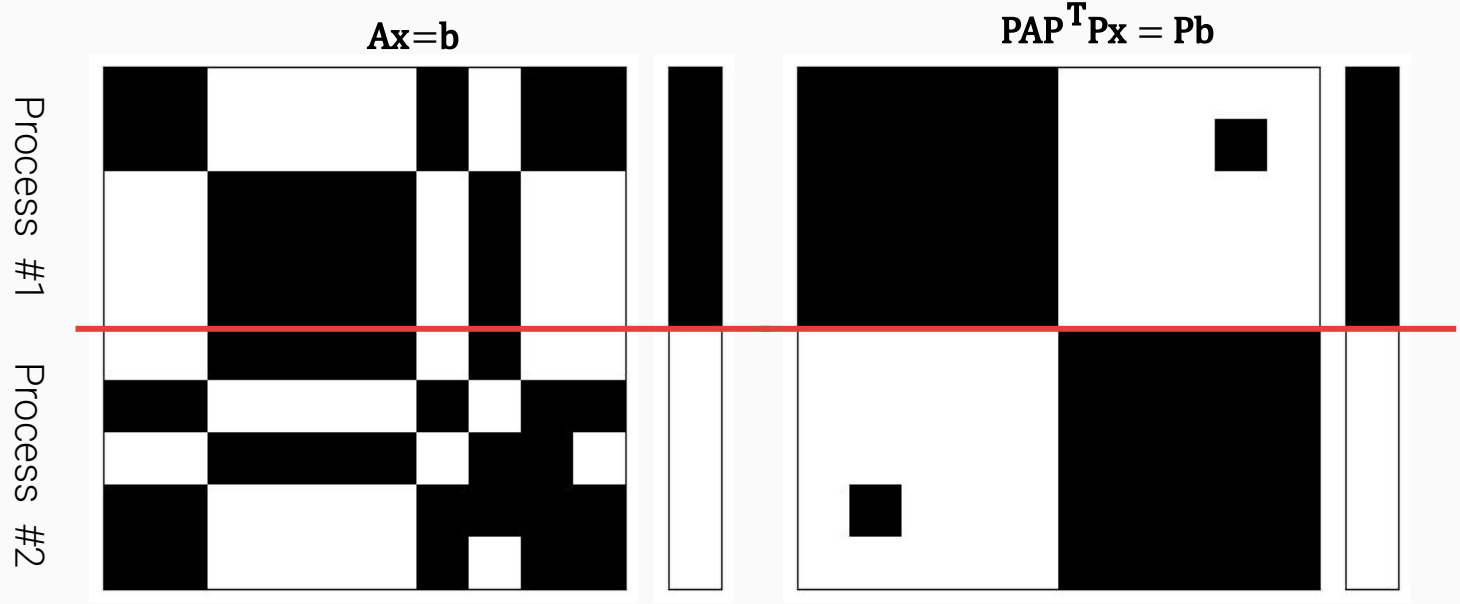
Perform parallel computation, while minimizing communication.



You already did a basic version of this in the MPI PDF mesh discretization.

Matrix Reordering

(i) Reduce fill-in (during Cholesky factorization), (ii) improve cache performance, and (iii) reduce communication.



Classification of Graphs
Coordinate vs Non-Coordinate base

Coordinate Graphs:

- Vertices are in physical space with coordinates (2D or 3D).
- Example, PDE meshes and particle simulations, where spatial positioning is crucial.
- Partitioning often emphasizes spatial node distribution rather than edge-cut.

Non-Coordinate Graphs:

- Vertices are not in physical space.
- Example, abstract and relationship-based, common in social networks.
- Partitioning prioritizes minimizing edge-cut to reduce communication overhead, crucial due to complex connectivity patterns without spatial constraints.

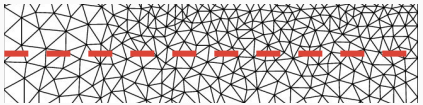
We will look at key algorithms for different types of graphs.

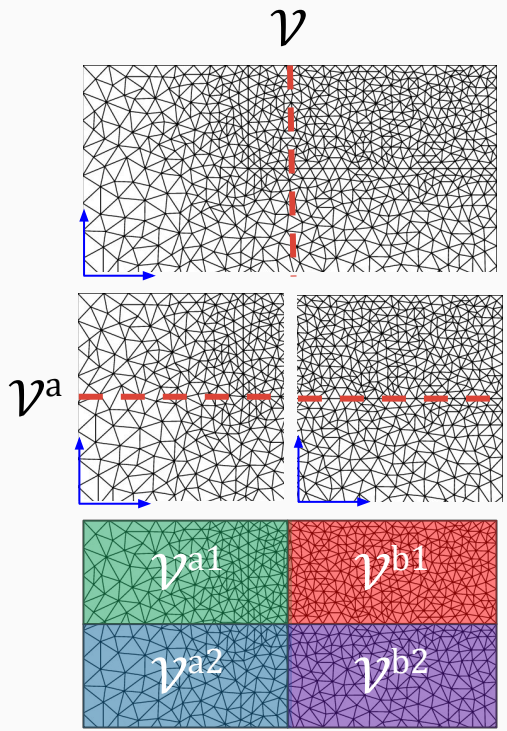
Coordinate Based
Coordinate Bisection

Partition along line orthogonal to the **standard basis.**

1. **Cut** with even division of points $\mathbf{X}$ (median). We can alternate **basis** between the different axis to keep recursive partitions.

Where is edge-cut calculated?
What is this procedure in 1D?
What can we say about this cut?

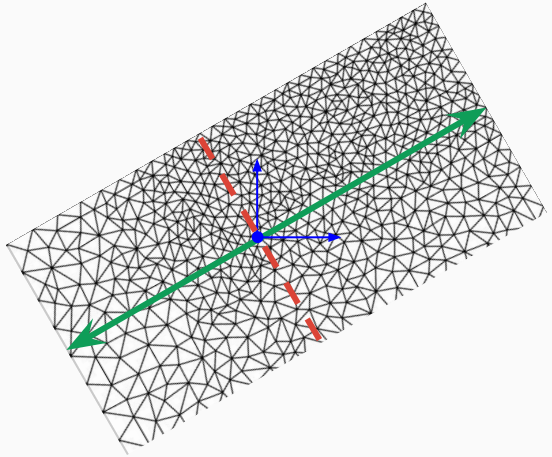




Coordinate Based
Inertial Partitioning—Minimal Rotational Inertia Line

Partition along line orthogonal to **some other line**.

1. **Center** mesh points $\mathbf{X}$.
2. Compute the **line** which minimizes the sum of the squared distance of $\mathbf{X}$ and the line.
3. Projecting $\mathbf{X}$ on the line.
4. The **cut** is than taken at median of the line, partition is orthogonal to the line.

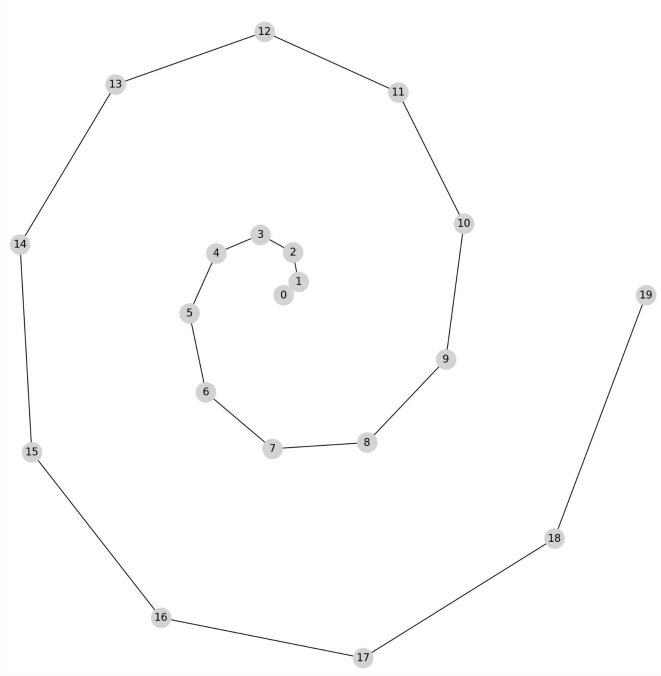


Project $\mathbf{X}$ on the eigenvector corresponding to the largest eigenvalue of $\mathbf{X}\mathbf{X}^{\mathrm{T}}$, than preformn 1D “Coordinate Bisection.”

Where is edge-cut calculated?
What is the complexity of this solution method?

What does Inertial Bisection do here ?

Nothing good…



Non-Coordinate Base
Greedy-algorithm for “Refinement”

Kernighan-Lin (70’s) cost $\mathcal{O}(|\mathcal{V}|^3)$ & **Fiduccia-Matteyes** (80’s) cost $\mathcal{O}(|\mathcal{E}|)$

Refine the edge-cut of the partitions $\mathcal{V}^a$ and $\mathcal{V}^b$

- Both assign nodes to partitions based on local rules.
- The KL algorithm operates on a pairwise exchange basis, while the FM algorithm simplifies this approach by not relying on pairwise exchanges.

Non-Coordinate Base
Fiduccia-Matteyes–A walk through

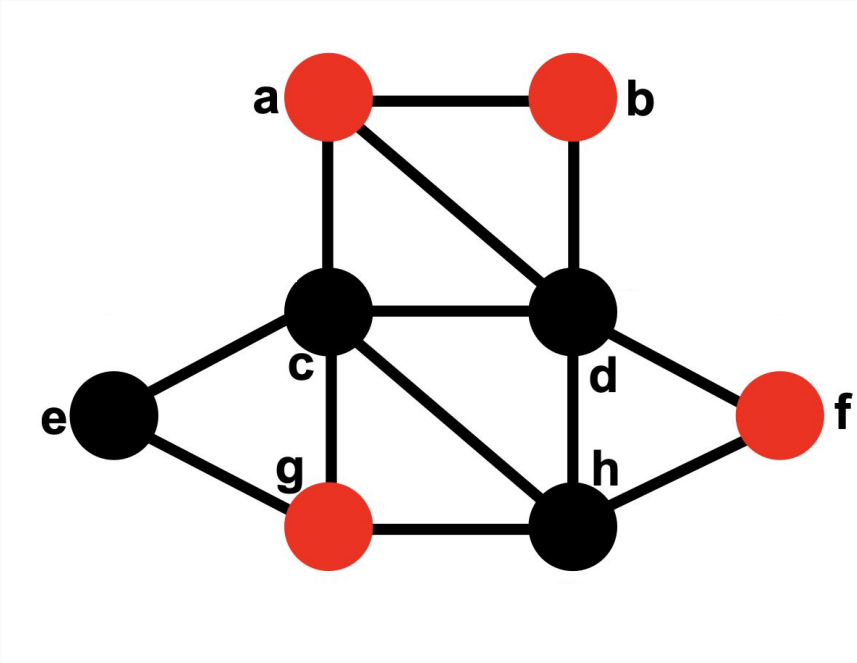
We ask "what if we swapped a vertex"...

Fiduccia-Matteyes Algorithm

1. Sweep vertices:
   a. For $v_1 \in \mathcal{V}^a$ select the most decrease in edge cut "*if*" $v_1 \in \mathcal{V}^b$.
   b. Compute edge-cut.

   c. Get $v_2 \in \mathcal{V}^b$ select the most decrease in edge cut "*if*" $v_2 \in \mathcal{V}^a$.
   d. Compute edge-cut.

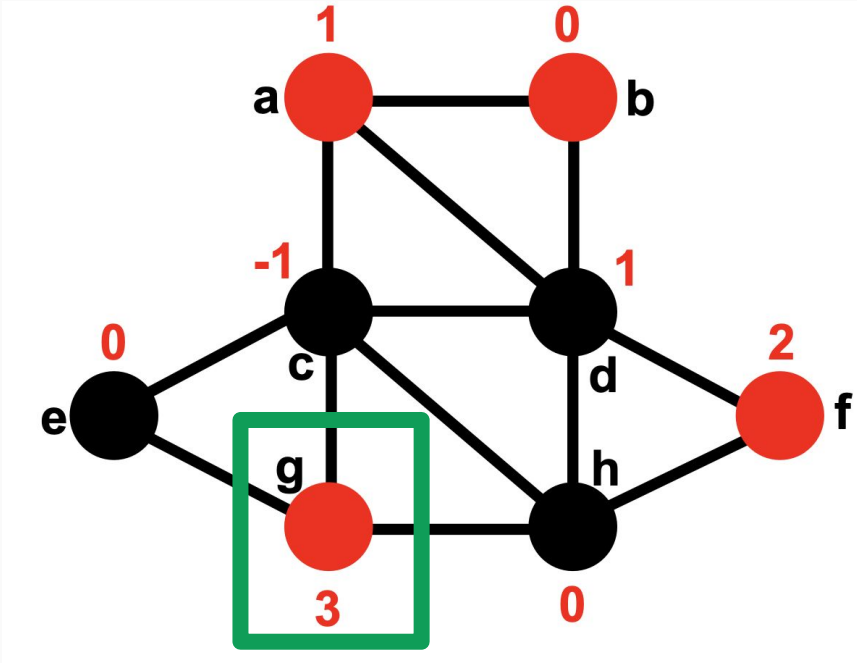
2. After sweep of all vertices, select configuration with the lower edge-cut.



Non-Coordinate Base
Fiduccia-Matteyes–A walk through

Partition $\mathcal{V}^a$

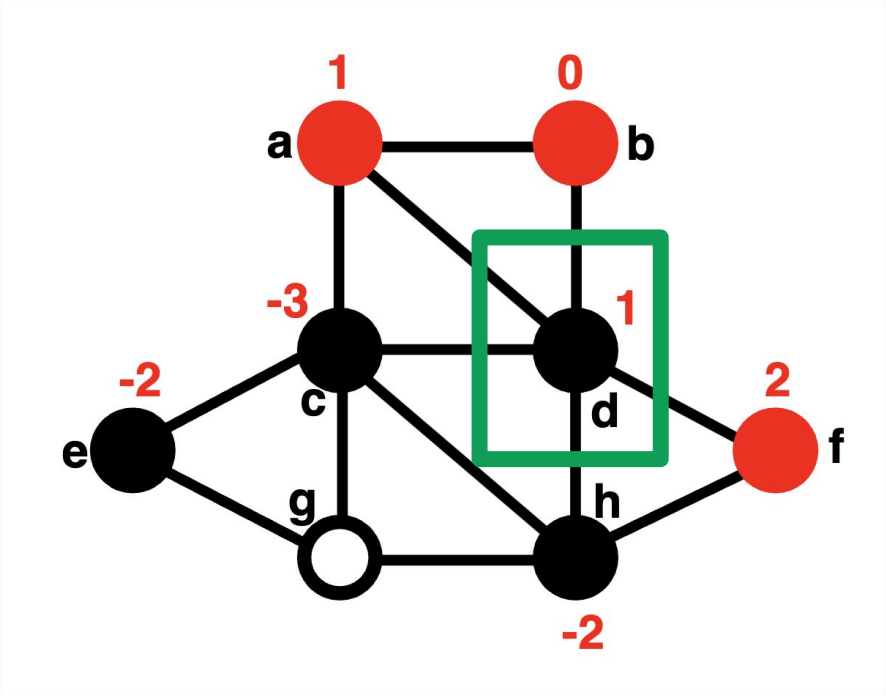
- The edge-cut of the graph is 8.
- In **red**, we show the values of swapping partitions or the “diff-value” of a vertex.
- Consider vertex “a”: the direct neighbors of “a” form a graph with an edge-cut of 2. If “a” changes partitions to black, the edge-cut becomes 1. Thus, the diff-value of “a” is 1.
- The biggest decrease is “g”.



Non-Coordinate Base
Fiduccia-Matteyes–A walk through

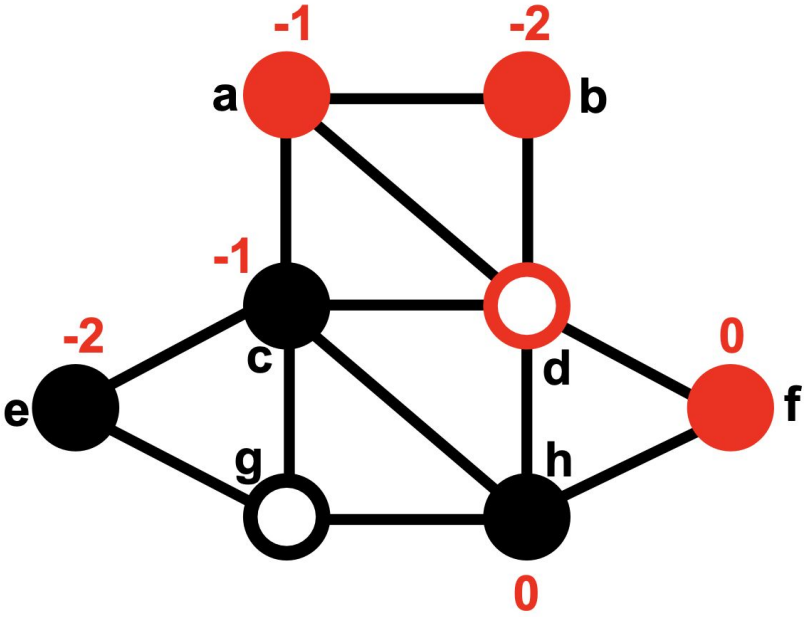
Partition $\mathcal{V}^b$

- We look to the **other partition**.
- The edge-cut of the graph is 5.
- The biggest decrease is “d”.



Repeat …

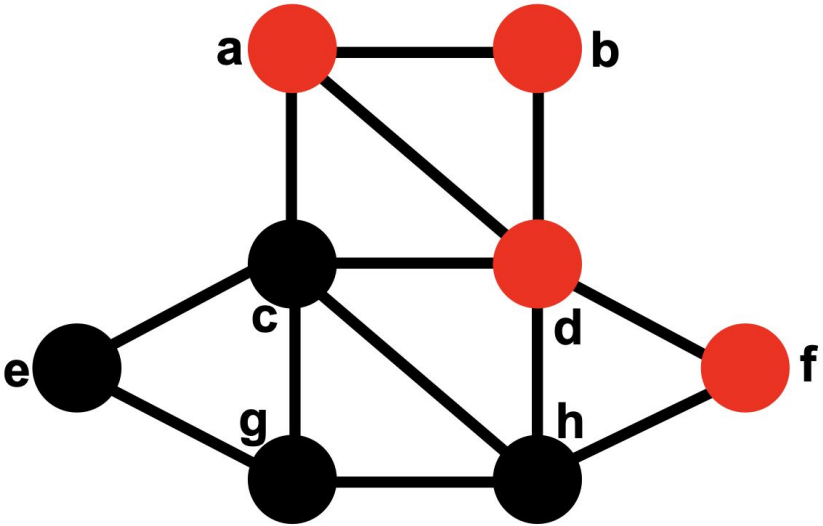
- The edge-cut is now 4.
- Repeat until we have selected all i and j.
- The next value selected would be "f".



Non-Coordinate Base
Fiduccia-Matteyes–A walk through

Select configuration with the lower edge-cut

- The history of edge cuts are $\mathbf{C}=\{8,4,5,9\}$
- Select the configuration $\mathbf{k}=\arg\min(\mathbf{C})=2$
- Selected configuration has edge-cut of 4.

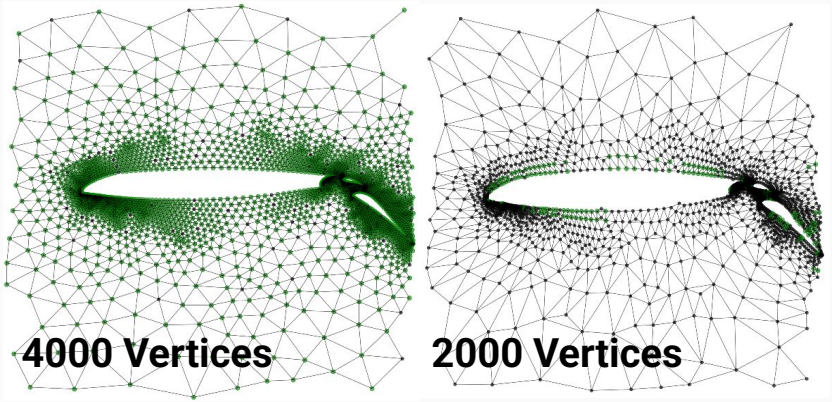


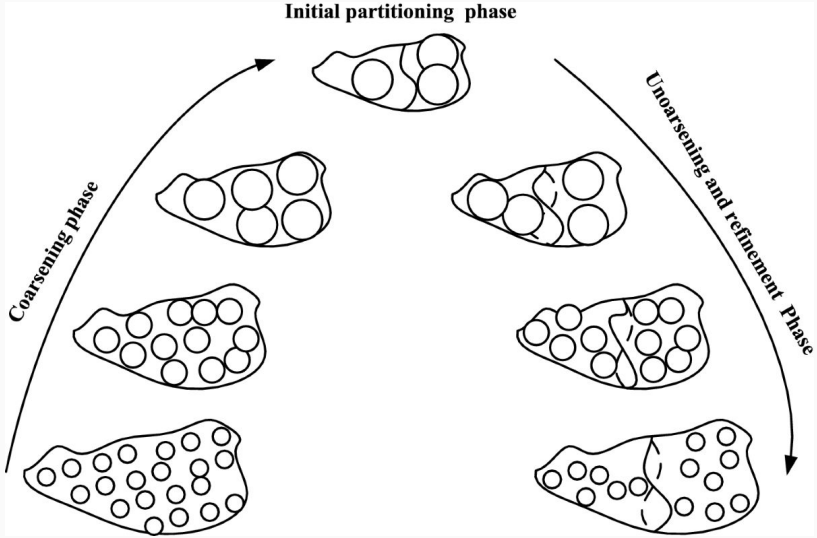
Note the FM operates on local view of the graph, i.e., local selection rules.

Non-Coordinate Base
Multilevel Partitioning

A simplified *global* view – Efficient and scalable graph partitioning.

- **Coarsening phase:** Reduce graph into smaller representations.
- **Uncoarsening & Refinement Phase:** Refine solution during uncoarsening.
- Examples: METIS





Exploring and optimizing partitioning of large designs for multi-FPGA based prototyping platforms (Umer & Bander 2020) :https://link.springer.com/article/10.1007/s00607-020-00834-5

Fielder (70s) using eigenvalues, cost of $\mathcal{O}(|v|^3)$

Transform the discrete partitioning problem into a relaxed, continuous optimization problem we can have global partitioning method.

See here for derivation.

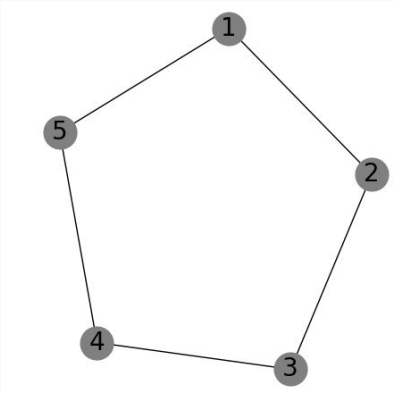
Non-Coordinate Base

Spectral Partitioning—An Eigenvalue Problem.

The Laplacian Matrix $\mathbf{L} := \ell(\mathcal{G}) \in \mathbb{R}^{n \times n}$

$\mathbf{L}$ is the negative adjacency matrix($\mathbf{A}$), with the "degrees" of $v \in \mathcal{V}$ on its diagonal ($\mathbf{D}$).

- Defined as $\mathbf{L} := \mathbf{D} - \mathbf{A}$
- Positive semi-definite $\mathbf{L} \succ 0$
- Row sum is 0, $\mathbf{Null}(\mathbf{L}) = \mathbf{1}$, i.e. $\mathbf{L1} = \mathbf{0}$
- Symmetric $\mathbf{L} = \mathbf{L}^T$




```
Adjacency Matrix
[[0 1 0 0 1]
 [1 0 1 0 0]
 [0 1 0 1 0]
 [0 0 1 0 1]
 [1 0 0 1 0]]

Laplacian Matrix
[[ 2 -1  0  0 -1]
 [-1  2 -1  0  0]
 [ 0 -1  2 -1  0]
 [ 0  0 -1  2 -1]
 [-1  0  0 -1  2]]
```

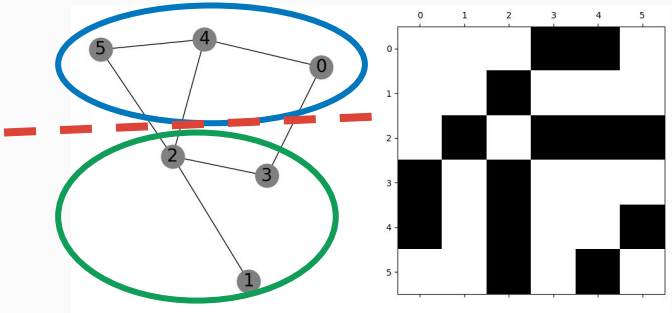
The construction of this matrix is very important ...

Non-Coordinate Base

Spectral Partitioning—An Eigenvalue Problem.

Consider $\mathbf{x} \in [1,-1]^n$ as the partition "label", $\mathbf{x}^T\mathbf{L}\mathbf{x} =$ "4 time the edge-cut".

$$\text{nnz}(\mathbf{A}) = 2|\mathcal{E}|$$
$$\text{nnz}(\mathbf{A}) = 2|\mathcal{E}^a| + 2|\mathcal{E}^b| + 2|\mathcal{E}^\Gamma|$$
$$14 \quad = 2(2+2+3)$$



Discrete Combinatorial Problem

1. Two partitions of equal size:

$$|\mathcal{V}^a| = |\mathcal{V}^b| \Leftrightarrow \mathbf{x}^T\mathbf{1} = 0$$

2. Minimum edge-cut:

$$\mathbf{x}^* = \arg\min \{\mathbf{x}^T \mathbf{L}\, \mathbf{x}\}$$

No polynomial-time algorithm is known to exist.

Lets relax the problem … now with $\mathbf{z} \in \mathbb{R}^n$

Continues Problem

1. Two partitions of (roughly) equal size:

$$|\mathcal{V}^a| = |\mathcal{V}^b| \Leftrightarrow \mathbf{z}^T\mathbf{1} = 0$$

2. Minimum edge-cut:

$$\mathbf{z}^* = \text{argmin}\{\mathbf{z}^T \mathbf{L}\, \mathbf{z}\}$$

Eigenvalue Problem (equivalent)

$$\mathbf{z}^* = \text{argmin}\{\mathbf{z}^T \mathbf{L}\, \mathbf{z}/\mathbf{z}^T\mathbf{z}\} \text{ st. } \mathbf{z}^T\mathbf{1} = \mathbf{0}$$

- $\mathbf{z}^*$ is the eigenvector associated with the smallest nonzero eigenvalue of $\mathbf{L}$ (Fiedler Vector).
- For every zero eigenvalue, there is a $\mathbf{z}$ corresponding partition with zero edge-cut.

How to convert $\mathbf{z}^*$ to labels, e.g., $\{0,1\}$? (using median)
What is the difference between this eigenvalue problem and inertial partitioning?

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

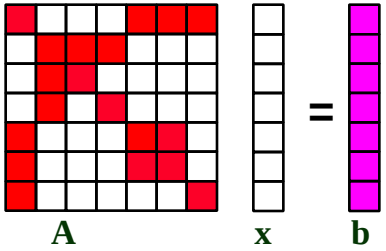
    Inertial Partitioning
- Partitioning without nodal coordinates
  - Ex: In model of WWW, nodes are web pages

    Fiduccia-Matteyes

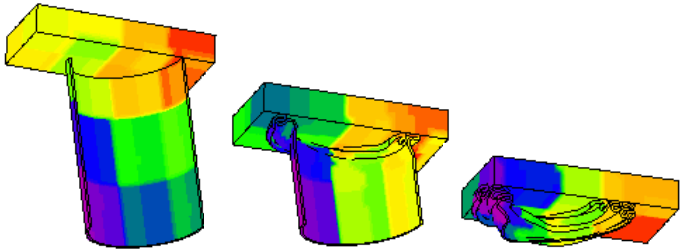
    Spectral Methods
- Multilevel acceleration
  - **BIG IDEA**, appears often in scientific computing
- Available implementations
- Beyond Graph Partitioning: Hypergraphs

Partitioning and Load Balancing

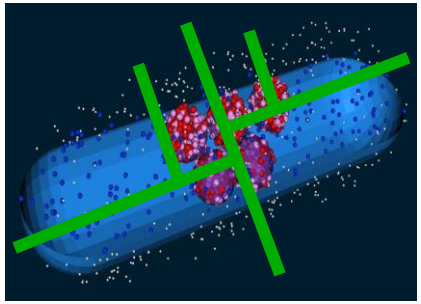
- **Goal:** assign data to processors to
  - minimize parallel application runtime
  - maximize utilization of computing resources
- **Metrics:**
  - minimize processor idle time (balance workloads)
  - keep inter-processor communication costs low
- Impacts performance of a wide range of simulations



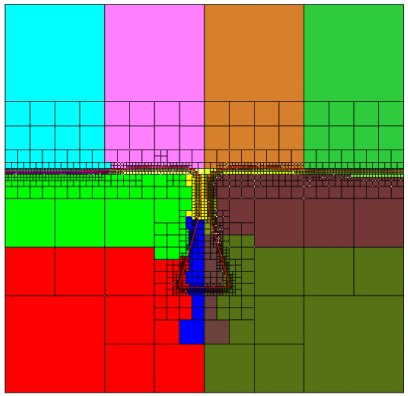
Linear solvers & preconditioners



Contact detection



Particle simulations

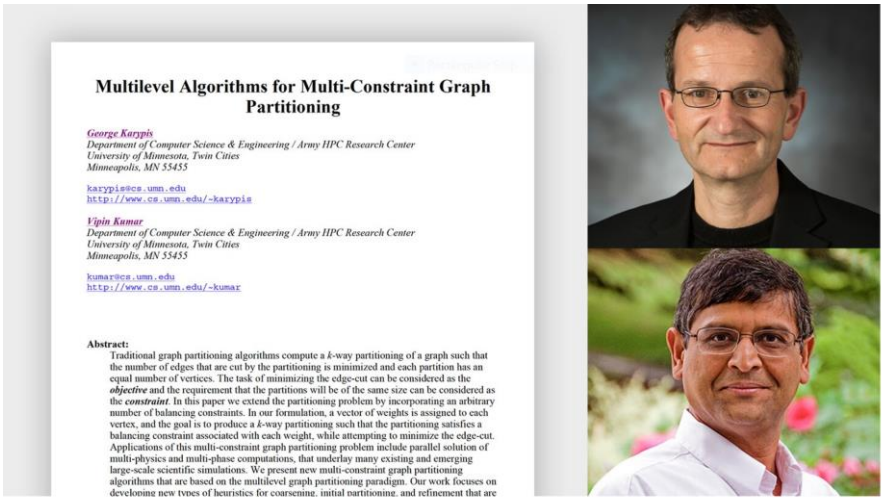


Adaptive mesh refinement

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)
- Goal: 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

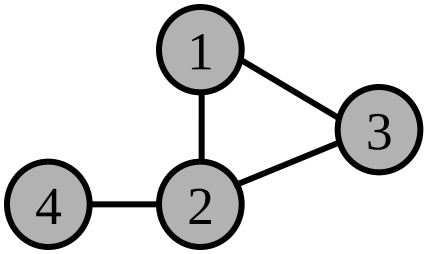
Abstract:

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.

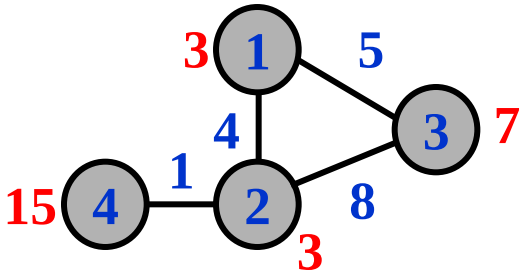
Definition of Graph

- Given a graph G = (V, E) with
  - Vertices V = { v_i | i=1,…,n}
  - Edges E = { e_{ij} | v_i and v_j 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= { $w_v(v_i)$ | v_iϵV} („weight of vertices“).
  - W_e= { $w_e(e_{ij})$ | e_{ij}ϵE} („weight of edges“).



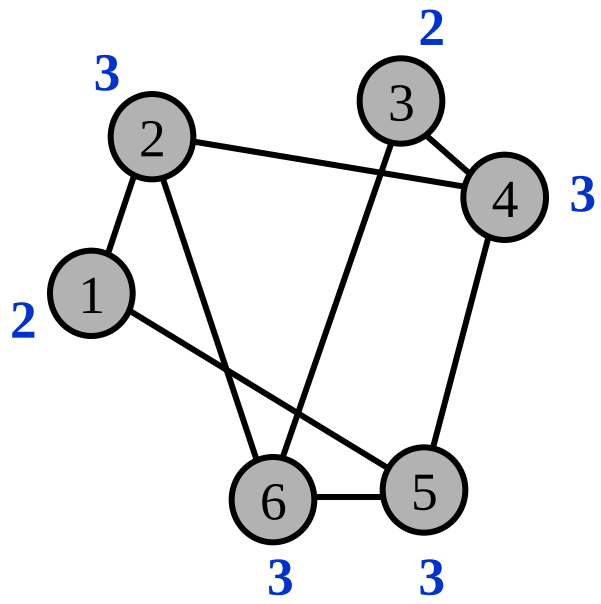
W_v = { 3, 3, 7, 15}, W_e= { 4, 5, 8, 1}

Examples for Graphs

- Symmetric sparse matrix and Graph G_A

$A =$

	1	2	3	4	5	6
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



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

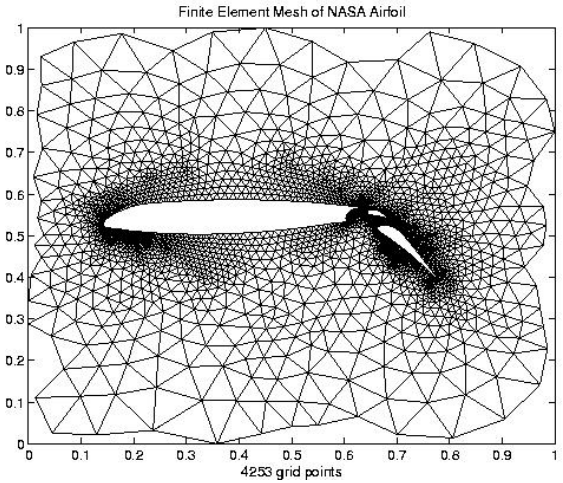
  $E = \{ (1,2), (1,5), (2,4), (2,6), (3,4), (3,6), (4,5), (5,6) \}$

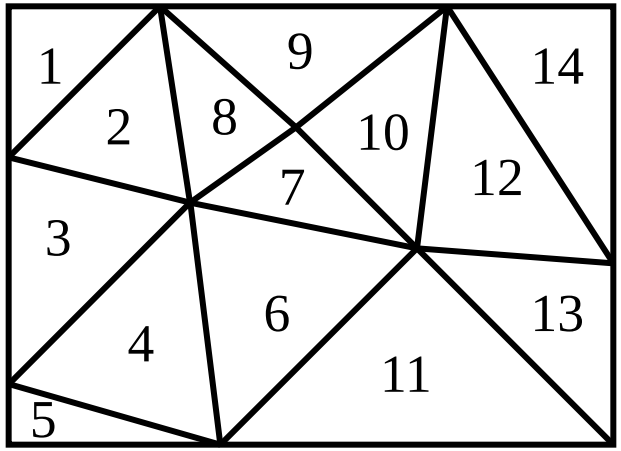
  $W_v = \{2, 3, 2, 3, 3, 3\}$ e.g. numbers of nonzeros in each row

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

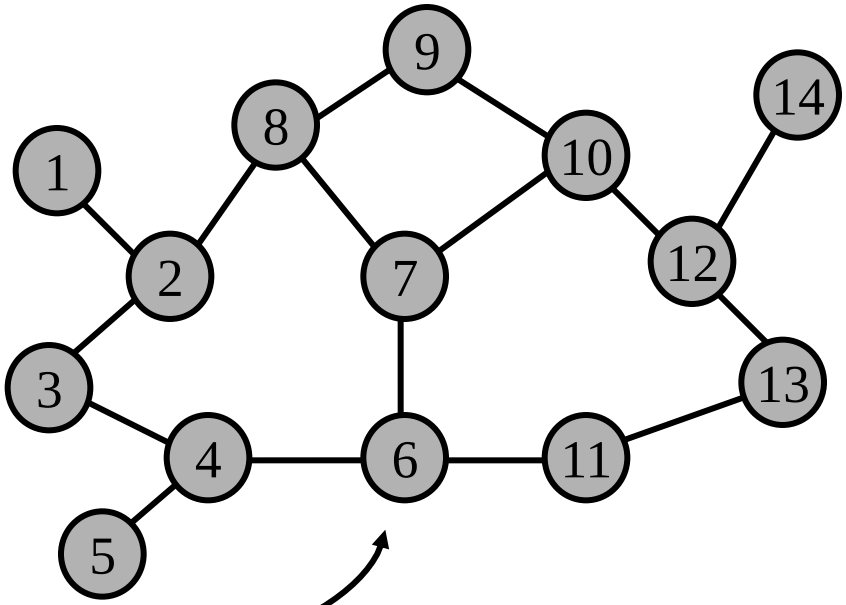
Examples for Graphs

- Finite-Element Simulations





Finite-Element Mesh



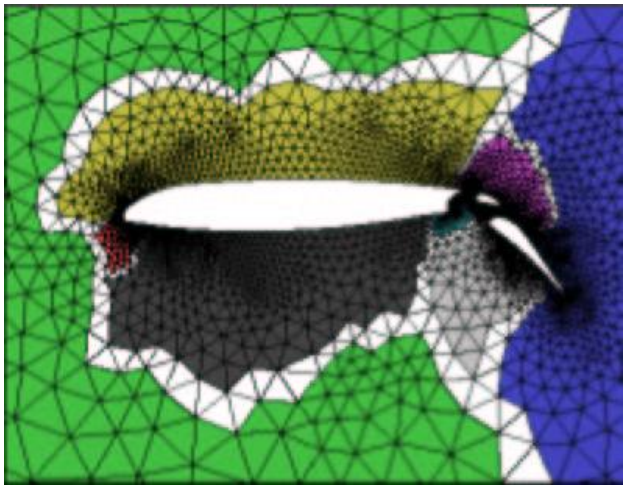
$G_{FE} = (V, E)$, $V = \{1, \ldots, 14\}$

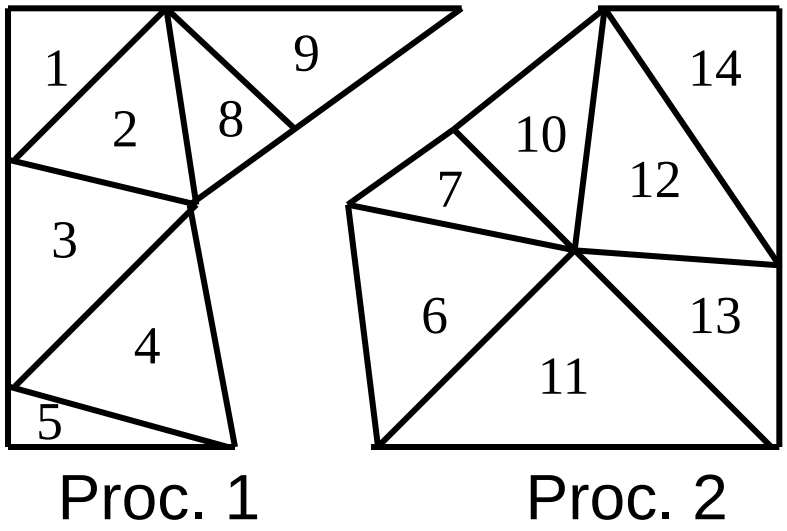
$E = \{(1,2), \ldots, (12,14)\}$

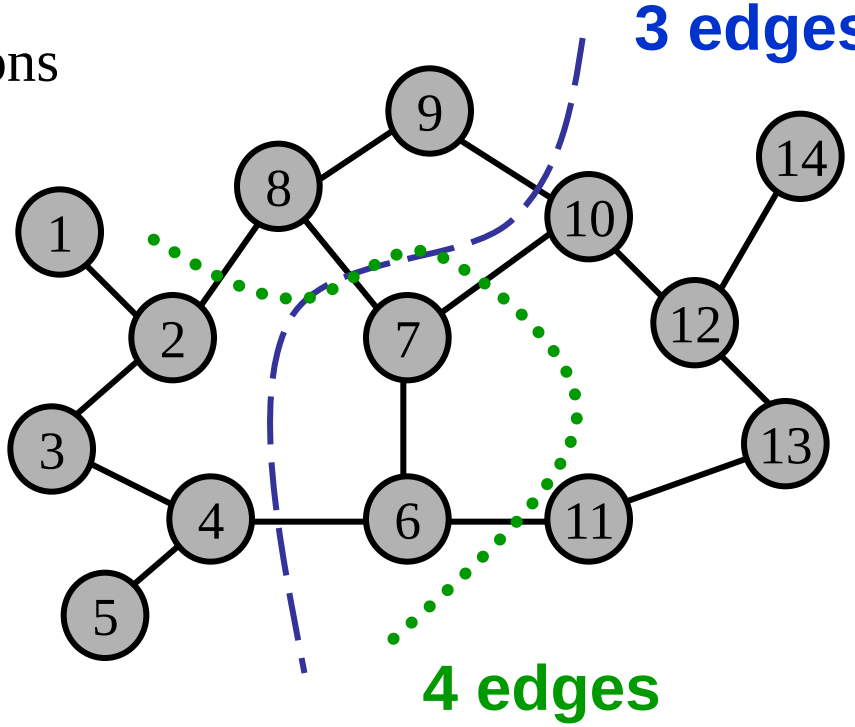
$W_e \equiv 1$, $W_v \equiv 1$

Examples for Graph Partitioning

- Parallel Finite-Element Simulations



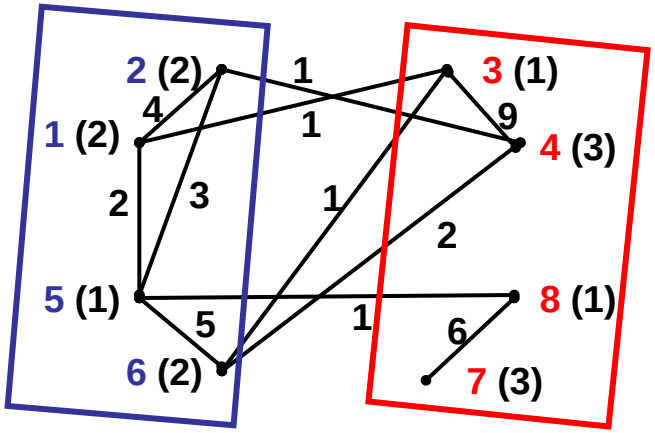




A good partitioning G_{FE} results in
- equal #elements/processor („load“ and „storage balancing“).
- Minimal #edges between P1 and P2 (minimal communication volume).

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

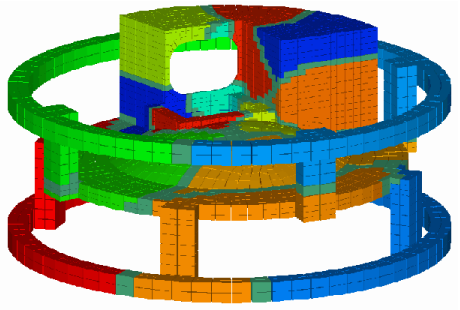
$$\min |\{e_{IJ} \epsilon E \mid v_i \epsilon V_1 \text{ und } v_j \epsilon V_2\}|$$

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

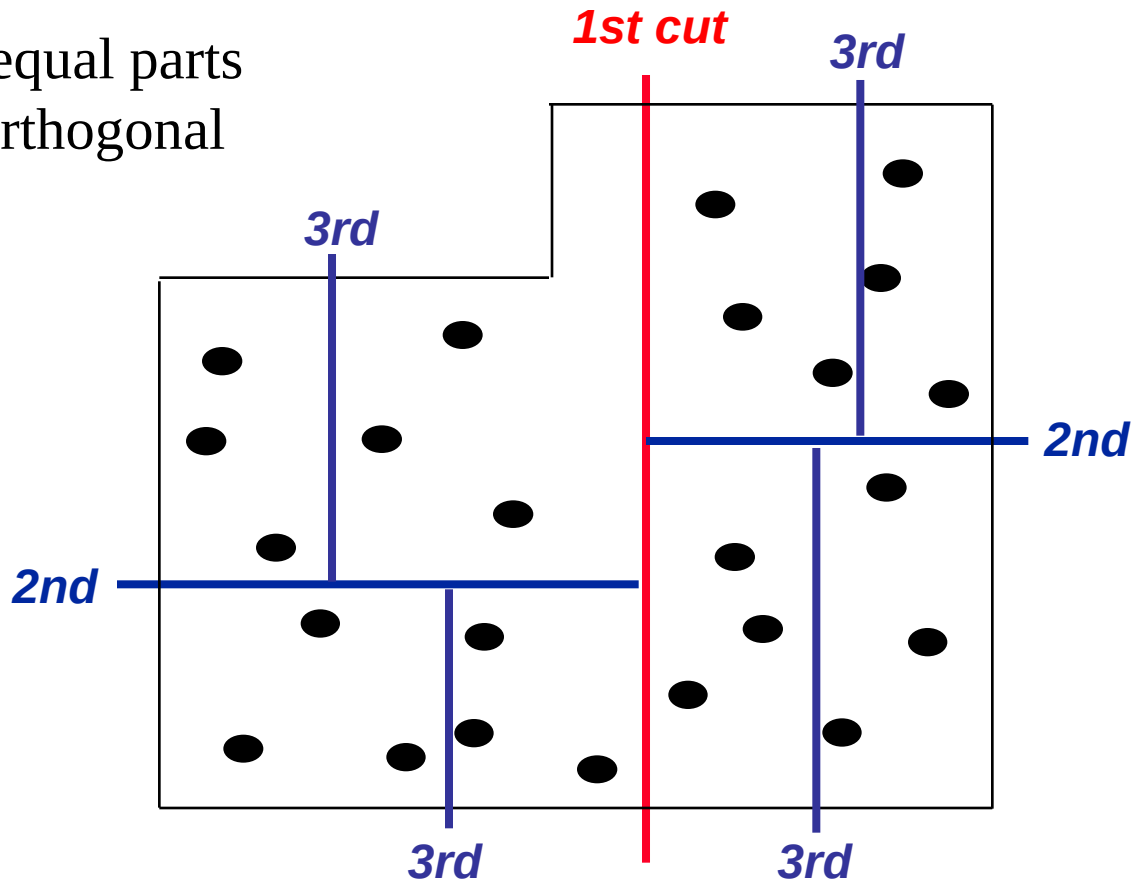
  Algorithms: **Fiduccia-Matteyes**

  Spectral Methods

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

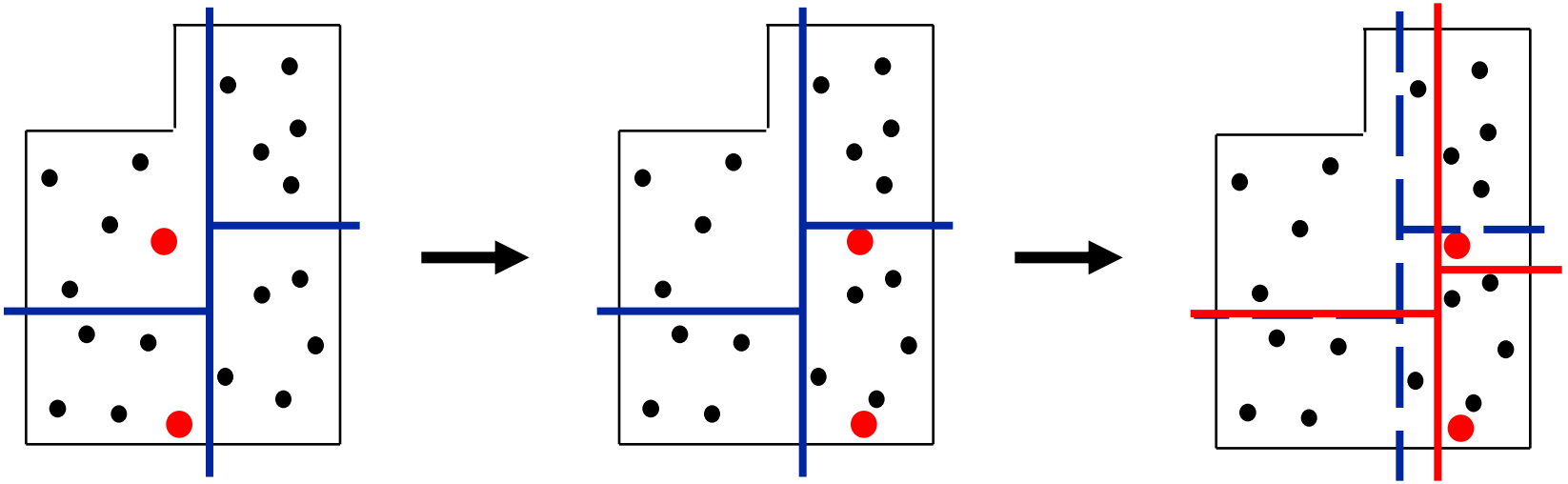
Nodal Coordinates — Recursive Coordinate Bisection (RCB)

- 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.



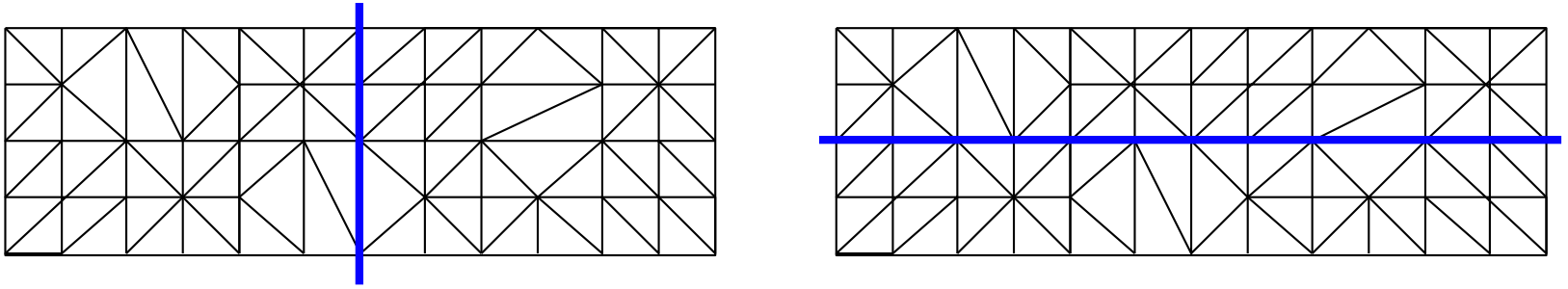
Nodal Coordinates — RCB Advantages

- 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.

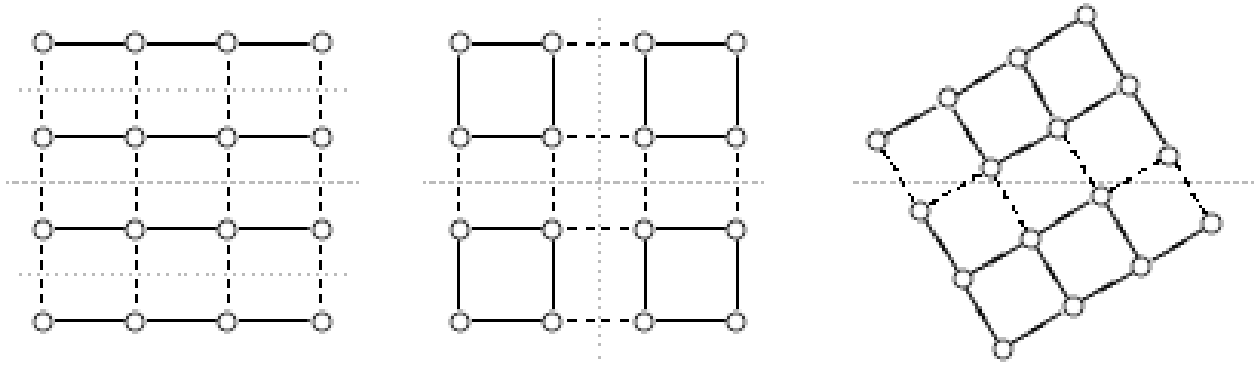


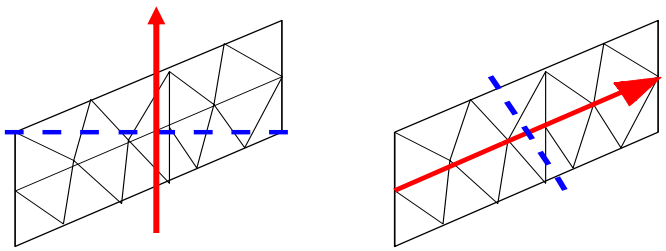
Nodal Coordinates — Coordinate Bisection

- Partition the domain along hyperplanes with node coordinates



- Change the coordinate systems

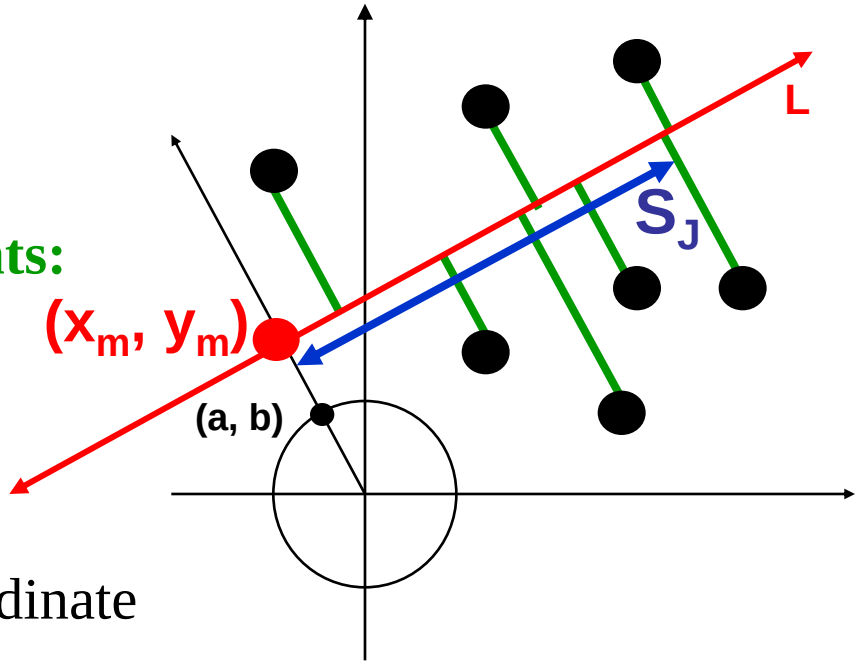




Good choice of coordinate system leads to inertial bisection

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

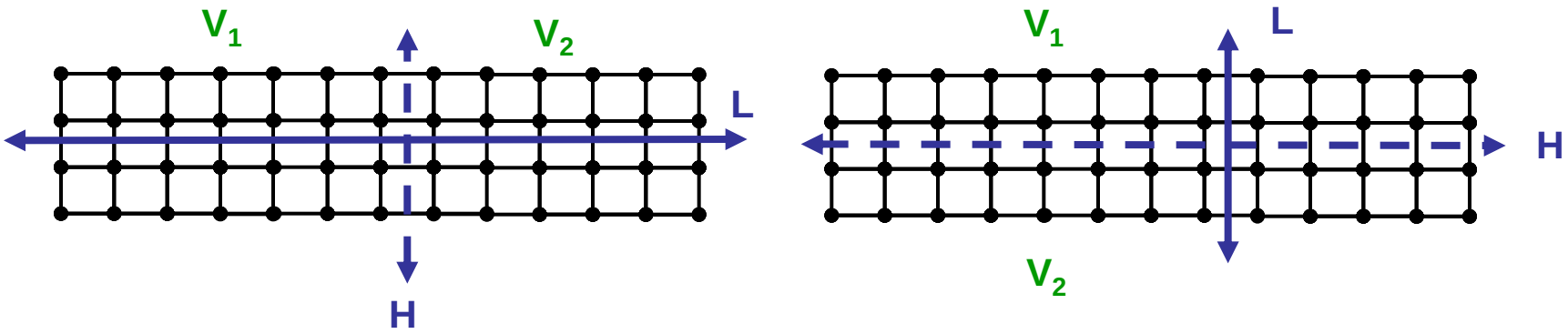
- Let S_k = median(S_1,…, S_n)

(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

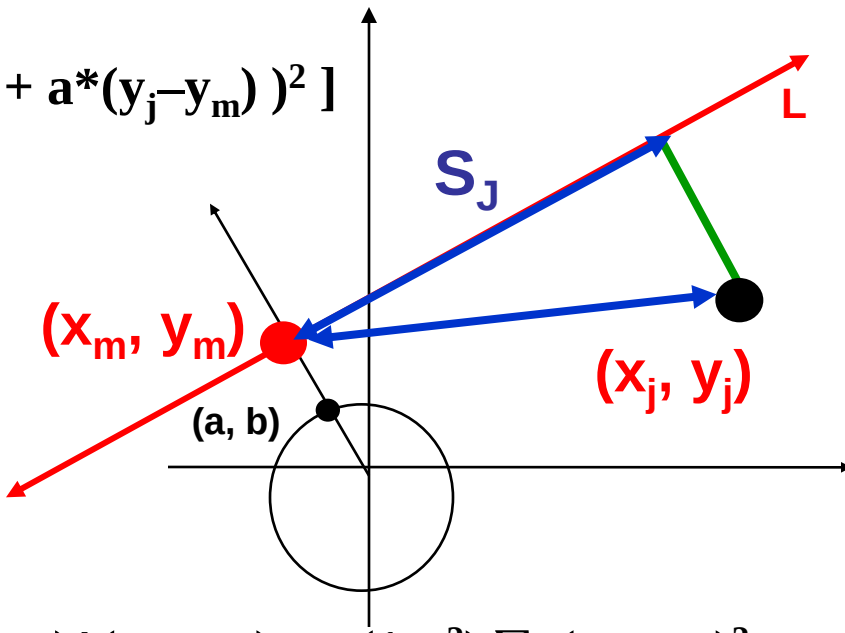
- Clearly prefer L on left below



- 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



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

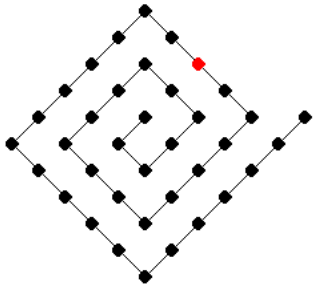
$M\,u = \lambda\,u \leftrightarrow$ Minimizing λ ?
$u^T M\,u = u^T \lambda\,u = \lambda\,u^T u = \lambda$

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

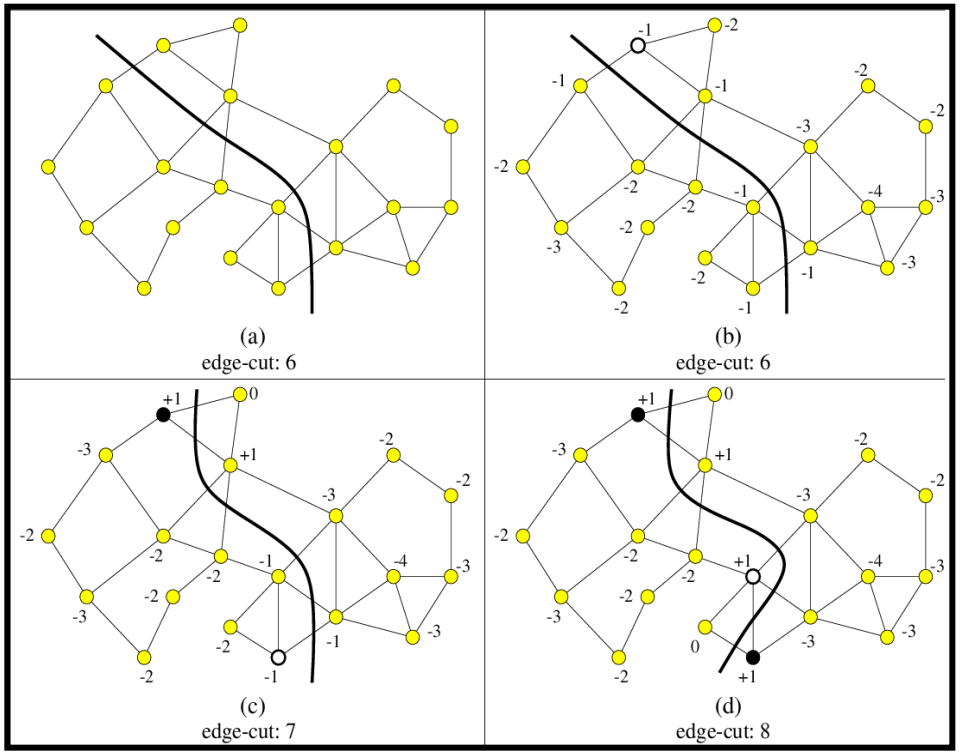
Coordinate-Free: Kernighan/Lin

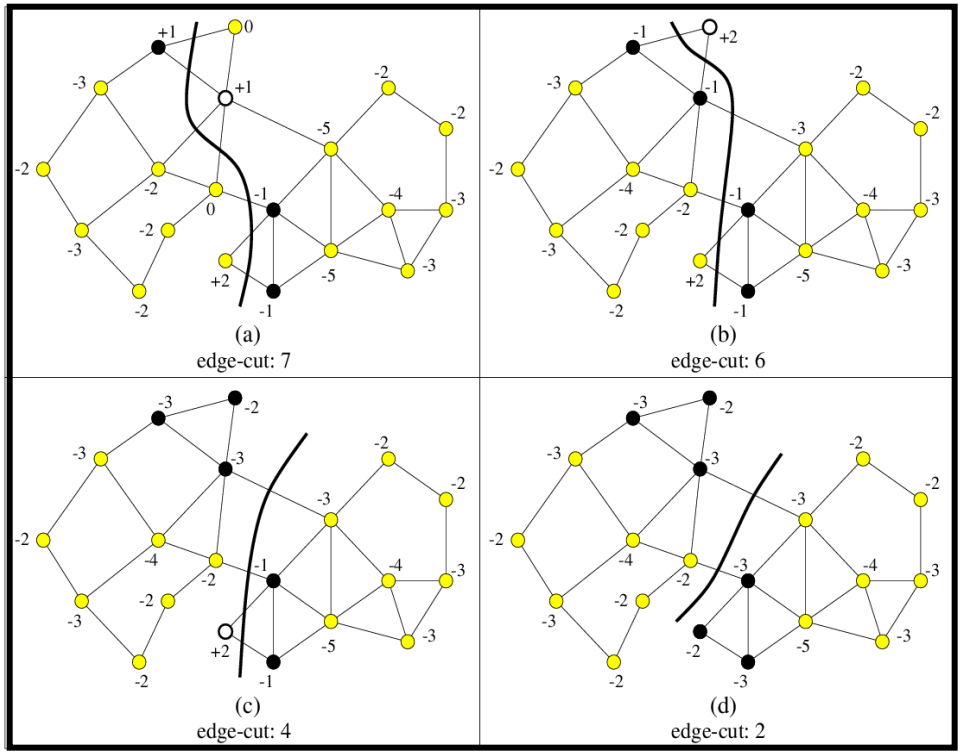
- Take a initial partition and iteratively improve it
  - Kernighan/Lin (1970), cost = O(|N|3) but easy to understand
  - Fiduccia/Mattheyses (1982), cost = O(|E|), much better, but more complicated
- Given G = (N,E,WE) and a partitioning N = A U B, where |A| = |B|
  - **T = cost(A,B) = S {W(e) where 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 — Fiduccia-Matteyes Algorithms (F/M)

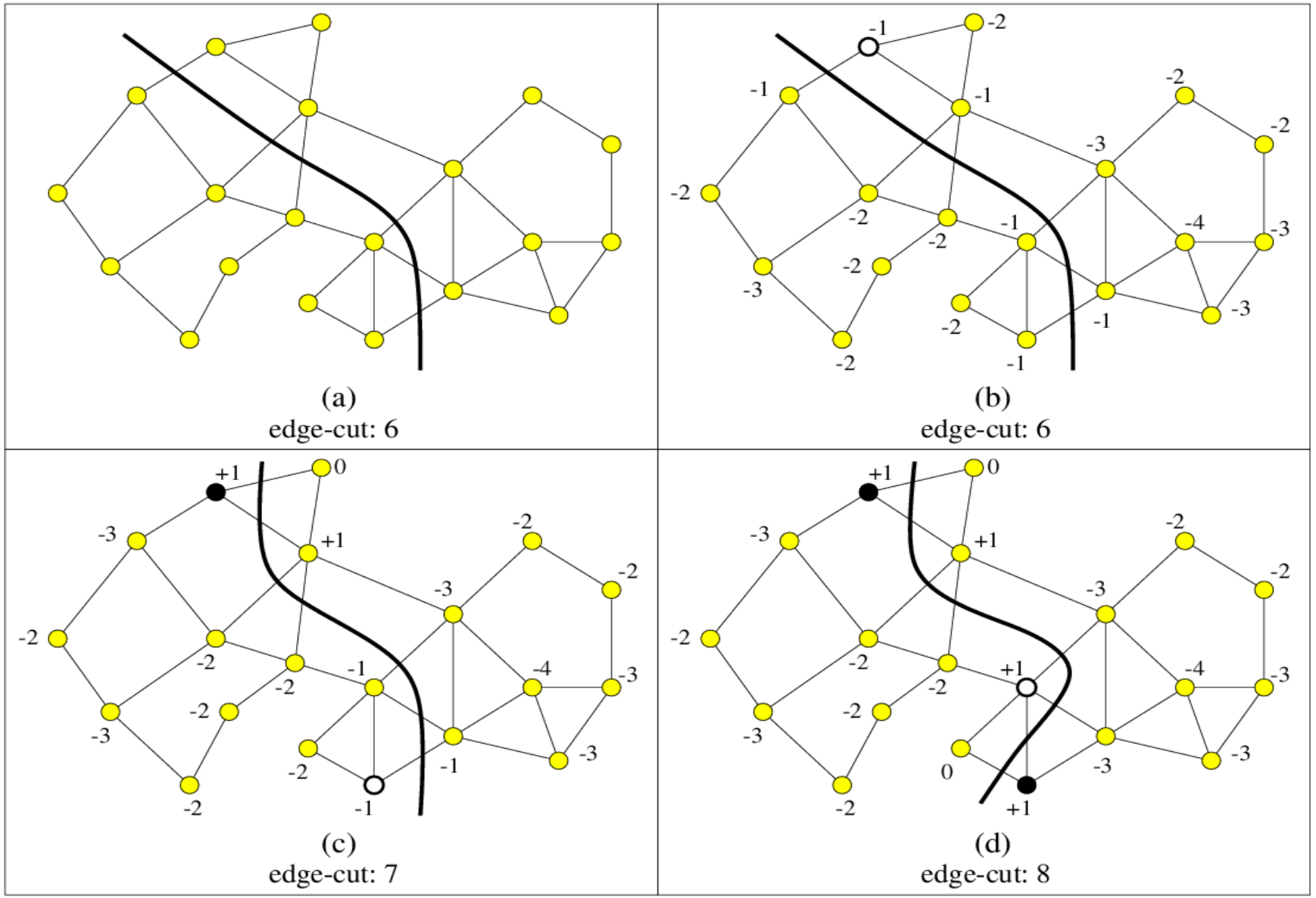
- **Example:**



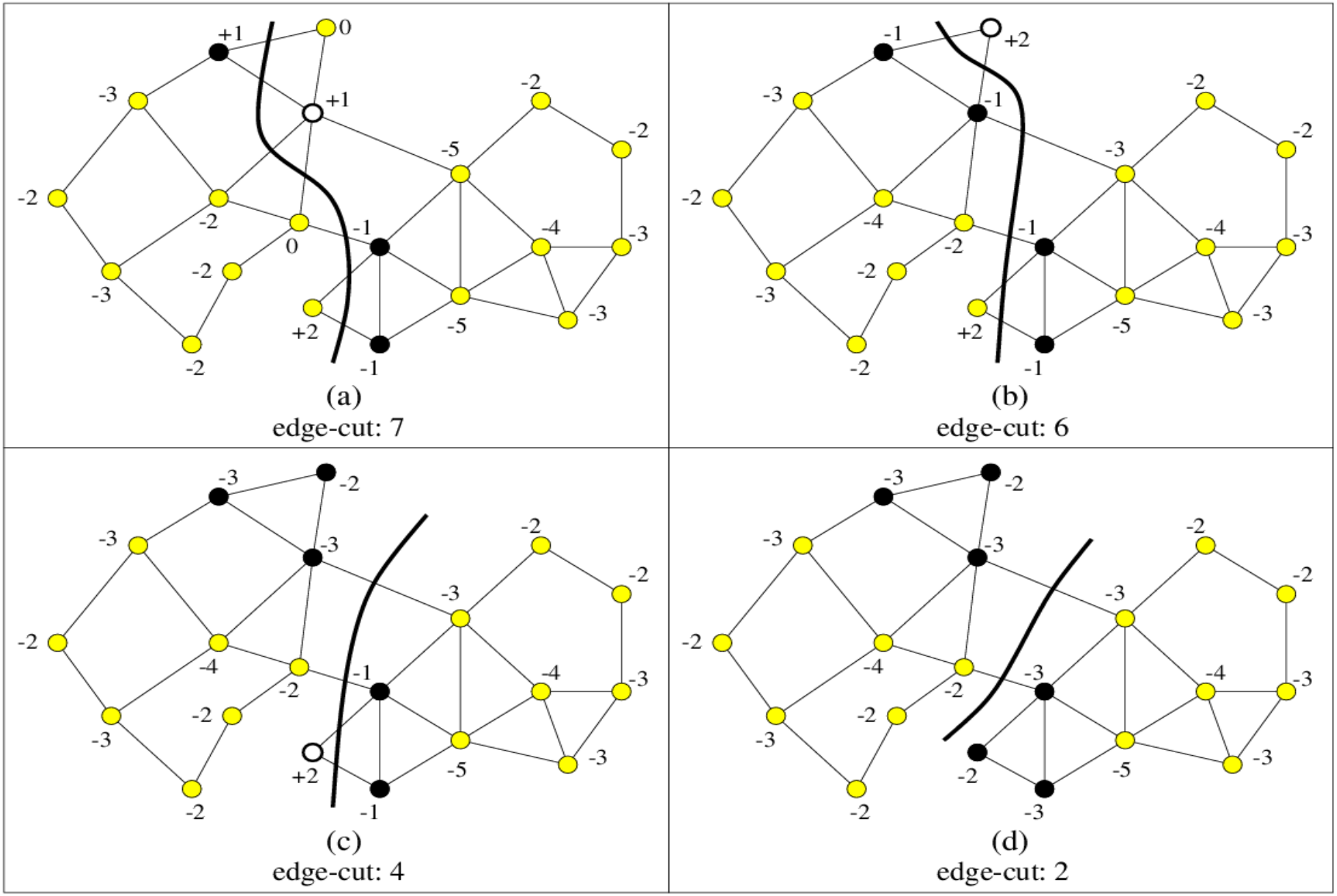


- All vertices v are marked with D(v) — Reduction of edge-cut.
- Load imbalance: 10% - therefore in iteration (c) the vertex marked with D(v) = +1 can not move into other domain.

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



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

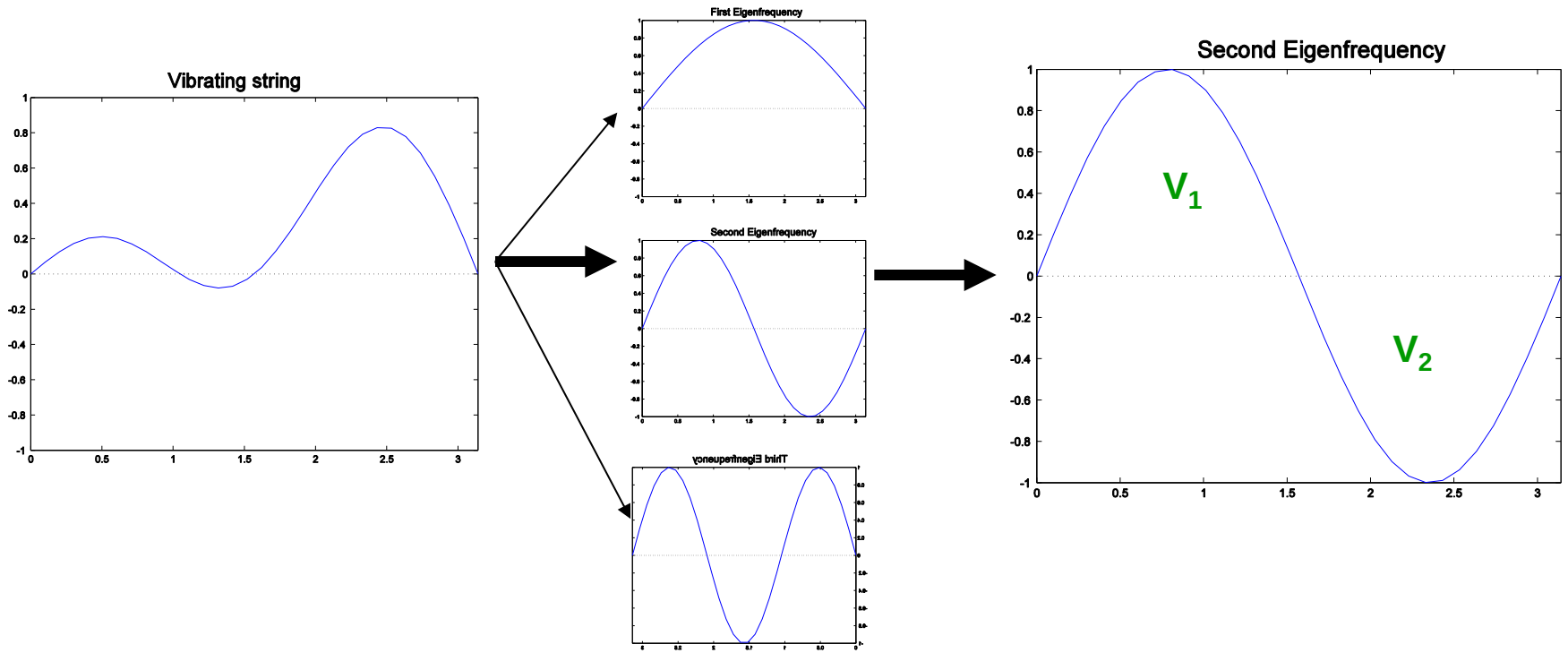


Coordinate-Free — Spectral Methods

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

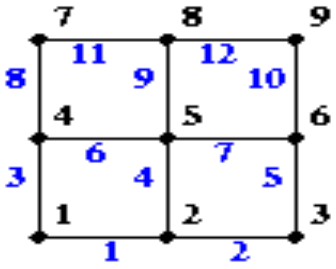
Coordinate-Free — Spectral Methods

- 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



	1	2	3	4	5	6	7	8	9
1	2	-1		-1					
2	-1	3	-1		-1				
3		-1	2			-1			
4	-1			3	-1		-1		
5		-1		-1	4	-1		-1	
6			-1		-1	3			-1
7				-1			2	-1	
8					-1		-1	3	-1
9						-1		-1	2

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_-

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

Proof: (next slide)

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

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

Relation between Laplace Matrix and Graph Partitioning

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

$$= \sum_{i} degree(i)$$

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

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

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

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

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

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

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

Relation between Laplace Matrix and Graph Partitioning

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

1.	$\lvert V_1\rvert = \lvert V_2\rvert$	$\Leftrightarrow \quad \sum_i x(i) = 0$
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	$x_I = \{+/- 1\}, \quad x^T * x = n$
	$x^T * e = 0$ 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 → L(G) has n orthonormal eigenvectors $u_1, \ldots, u_n$ with eigenvalues $0 = \lambda_1 \leq \ldots \leq \lambda_n$ and $u_1 = \text{sqrt}(n) * e$, $e = [1, 1, \ldots, 1]^T$.
- 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
  $z^T * u_1 = (\sum \alpha_i u_i)^T * u_1 = \alpha_1 u_1^T * u_1 = \alpha_1 \Rightarrow \mathbf{\alpha_1 = 0}$

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

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

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 $z = sqrt(n)*u_2$.

- **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

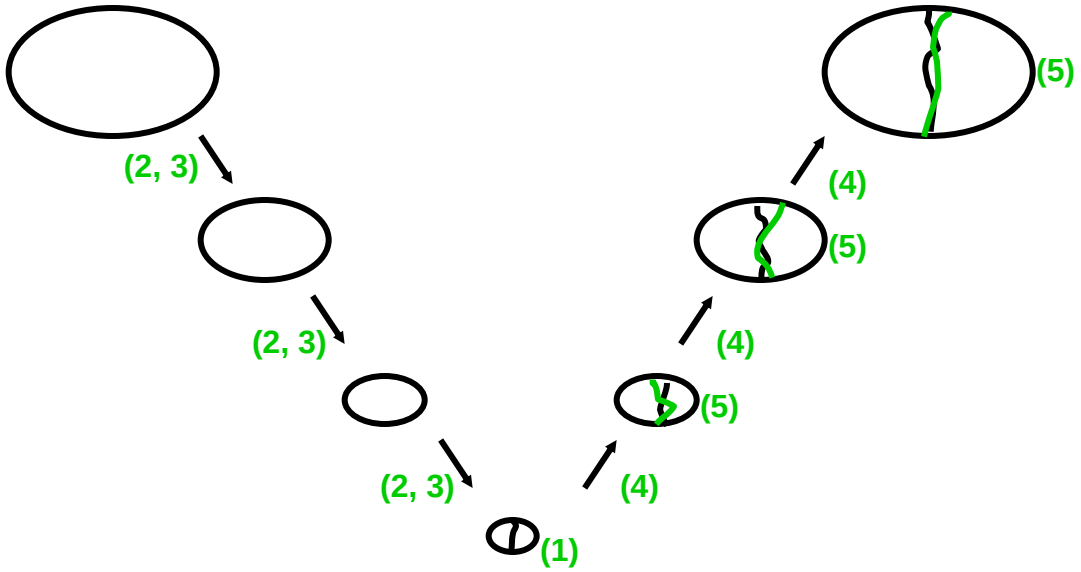
- If we want to partition G(V, E), but it is too big to do efficiently, what can we do?
  - 1) Replace G(V, E) by a **coarse approximation** G_C (V_C, E_C), and partition G_C instead
  - 2) Use partition of G_C to get a rough partitioning of G, and then iteratively improve it
- What if G_C still too 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 G_C = (V_C, E_C)
(3)         (V_C+ , V_C- ) = Multilevel_Partitionn( V_C, E_C )
(4)         Expand (V_C+ , V_C- ) to a partition  (V+ , V- ) of V
(5)         Improve the partition ( V+ , V- )
            Return ( V+ , V- )
        endif
```

How do we

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



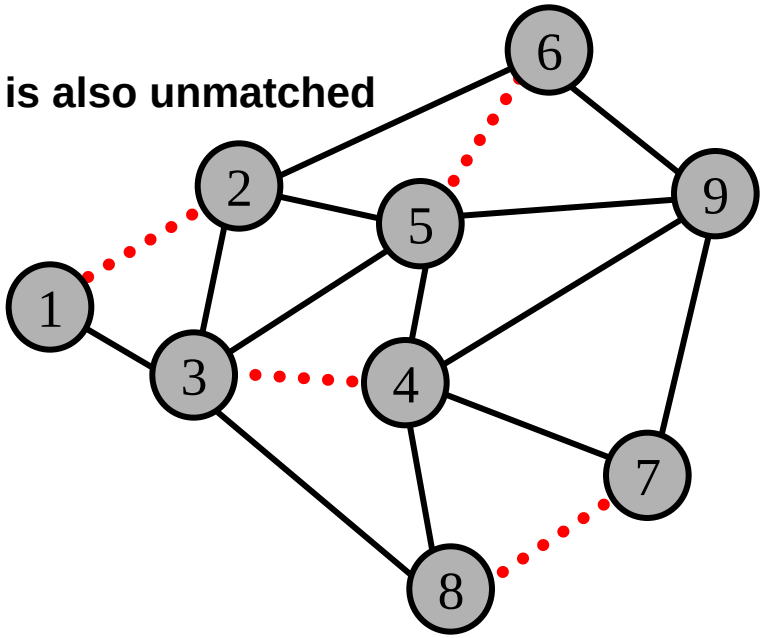
Multilevel Partitioning — Multilevel Fiduccia-Matteyes

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

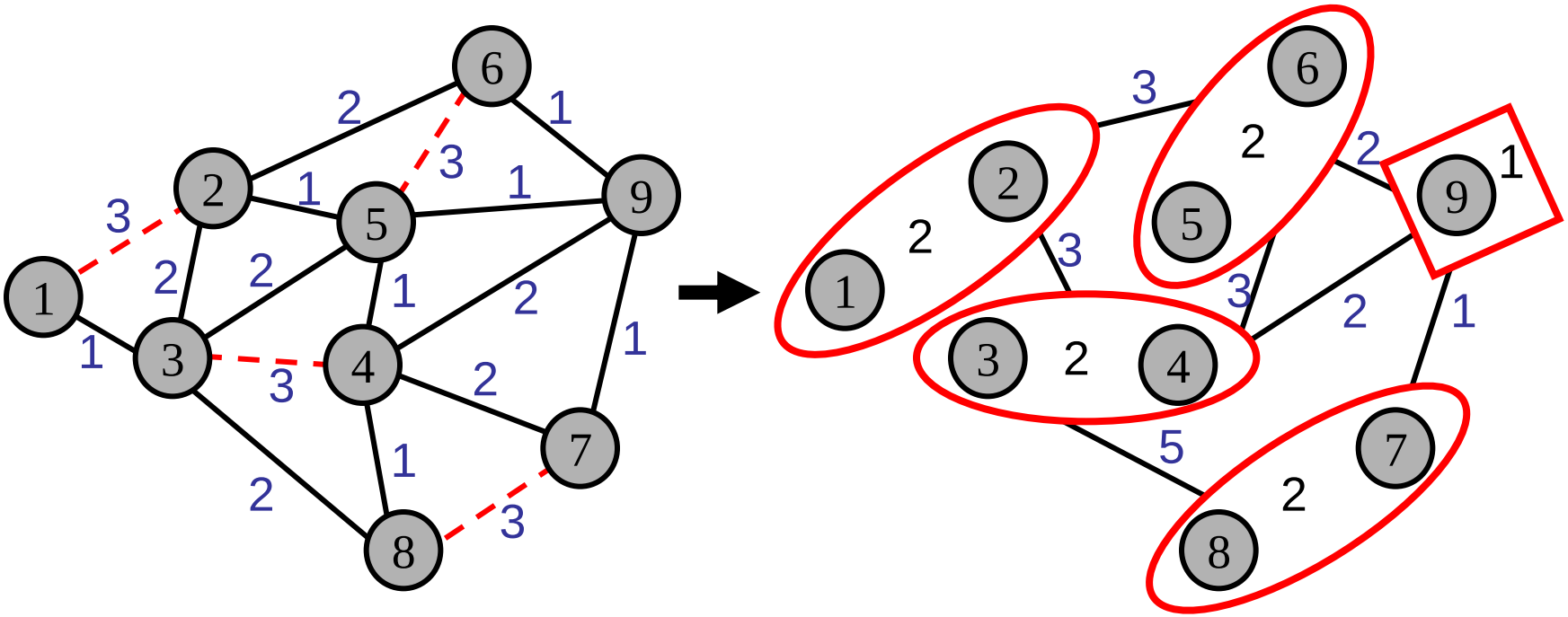
Multilevel Partitioning — Maximal Matching

- *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

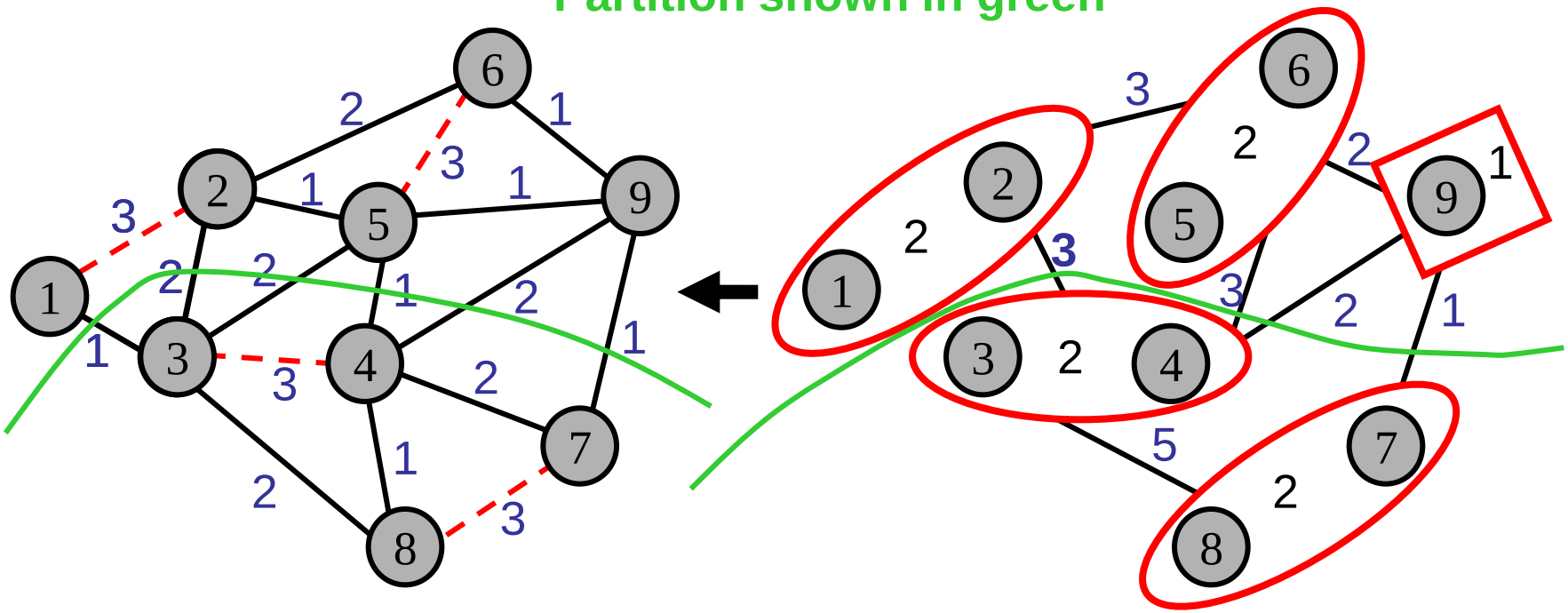
Put edge ee = (v(e), v(r)) in E_c

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

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

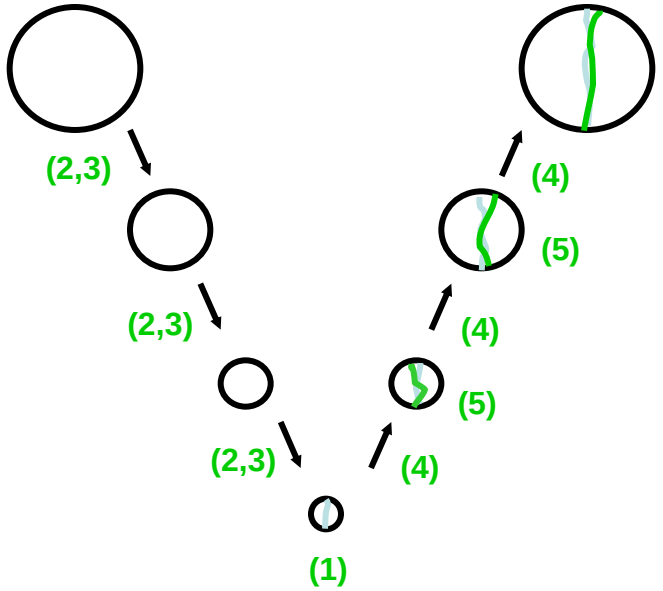
Vertex weights are black

Multilevel Spectral Bisection

```
f = Fiedler ( V, E )
    … Recursive computation of Fiedler Vector of Laplacian  L(G)
    if |V| is small
(1)     Calculate f=u2 using  eigenvalue/eigenvector algorithms
        Return f
    else
(2)     Coarsen G to get an approximation Gc = (Vc, Ec)
(3)     f ´ = Fiedler (Vc, Ec )
(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?**



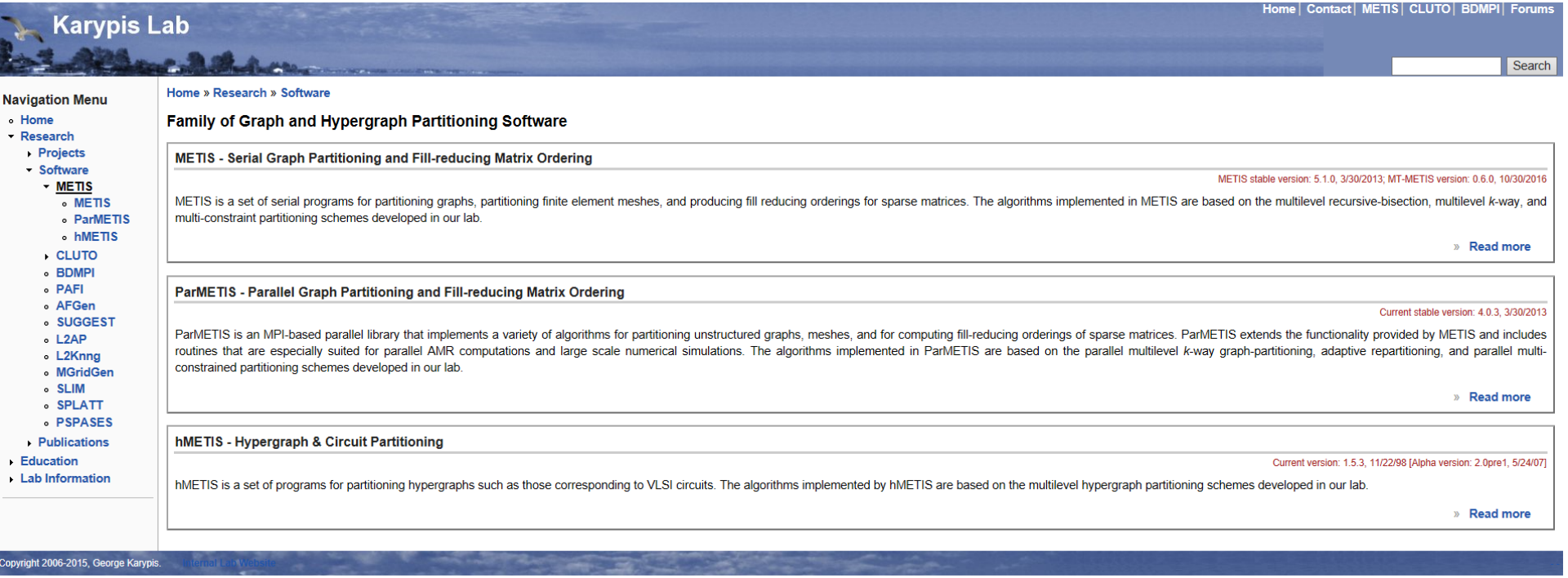
Content

- 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

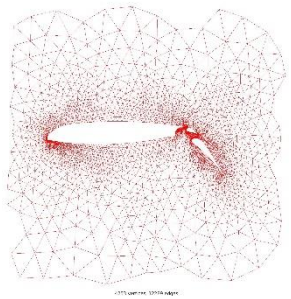
- Multilevel Kernighan/Lin
  - METIS and ParMETIS (glaros.dtc.umn.edu/gkhome/views/metis)
  - SCOTCH and PT-SCOTCH (www.labri.fr/perso/pelegrin/scotch/)
- Matlab toolbox for geometric and spectral partitioning by Gilbert, Tang, and Li: https://github.com/YingzhouLi/meshpart
- Multilevel Spectral Bisection
  - 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/)

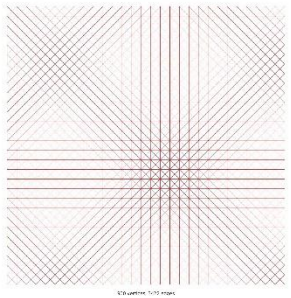
METIS - Family of Graph and Hypergraph Partitioning Software

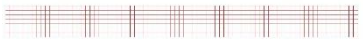


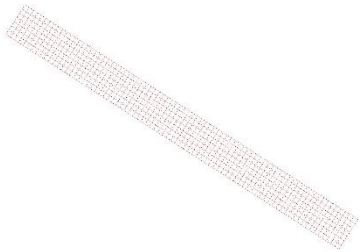
http://glaros.dtc.umn.edu/gkhome/views/metis

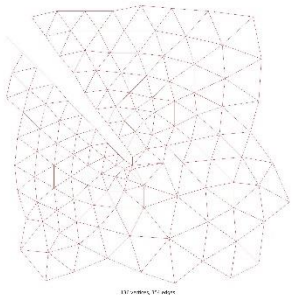
Demo – Partitioning in Matlab

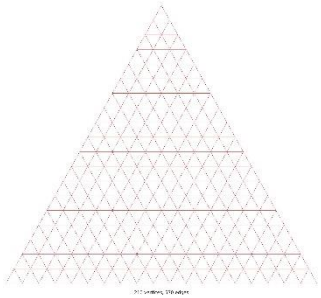


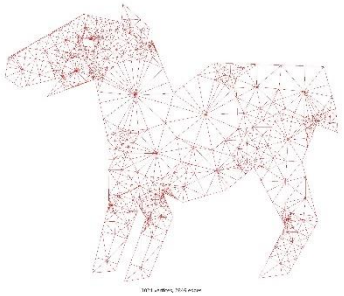


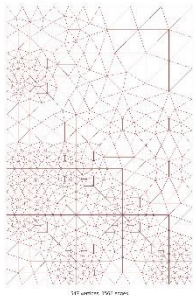












Università della Svizzera italiana
Institute of Computing CI

High-Performance Computing, Fall 2024
Lecturer: Dr. A. Eftekhari, Prof. O. Schenk
Assistants: M. Lechekhab, D. Vega,
D. Santarsiero, J. Palumbo, I. Ecevit

Project 6
Large Scale Graph Partitioning

Due date: 11 December 2024 at 23:59 (See iCorsi for updates)

In this project, we will investigate the graph partitioning problem within the context of domain decomposition for high-performance computing (HPC) applications.
You may do this project in groups of students (max four or five). In fact, we prefer that you do so.

1 Graph Partitioning with Matlab: Load balancing for HPC

Partitioning a computational problem into sub-problems that can be solved efficiently in parallel is a recurring task in HPC. In order to be efficient, the sub-problems should be of near equal size, to ensure load balancing, and the required communication between the sub-problems should be minimized. We have already encountered this problem in a few instances during the course, mainly in the form of domain decomposition. However, the domains were mostly simple and rather square. In this project, we consider more complex grids or meshes. We abstract domain decomposition as a graph partition problem.

The purpose of this sub-project is to familiarize yourself with some elementary graph partitioning algorithms and to implement them in the Matlab. We compare the resulting partitions with the METIS[1] graph partitioning software [7, 8, 9]. METIS was developed by Karypis and Kumar and is probably the most widely used graph partitioning software. In terms of computational efficiency, numerical experiments have demonstrated that graphs with several millions of vertices can be partitioned to 256 parts in a few seconds on current generation workstations and laptops using METIS. We will interface Matlab and METIS through the precompiled Matlab interface (`metismex.mexmaci64`) for Mac and a precompiled version for Linux (`metismex.mexa64`, Ubuntu, glibc 2.27)[2] which allow us to use the functions `METIS_PartGraphRecursive` and `METIS_PartGraphKway` with the same arguments and argument order described in the Metis manual.

We refer to Elsner [5] (see the course web page) for a comprehensive introduction to graph partitioning. For (much) more on graph partitioning, we refer to Bichot and Siarry [1]. Many other practical applications and the recent advances in the field of graph partitioning can be found in Buluç et al. [2] and references therein.

1.1 Graph partitioning: Generalities

Consider the mesh with 10 cells in Fig. 1a. Assuming that for the computation only cells sharing an edge have to exchange data[3], this mesh can be represented by the dependency graph shown in Fig. 1b. To decompose the mesh in two parts suitable for parallel processing, one tries to partition the mesh into two sub-meshes with the same number of cells and with a minimum number of edges between. This is equivalent to partition the graph in Fig. 1b into two complementary sub-graphs with equal number of vertices and a minimum number of edges between the two sub-graphs. In other words, we partition the graph in two equal parts by cutting the least possible number of edges.

For the simple example in Fig. 1, the solution is obvious. However, the solution is less obvious in general for larger meshes. In the following, we formalize the graph partitioning problem in a suitable manner for the scope of this project and present three bisection algorithms, where bisection restricts the problem to the partition into two sub-graphs. Partitioning into a power of two partitions $p = 2^l$ can then be achieved recursively.

Let $\mathcal{G} = (V, E)$ be an undirected graph with n vertices[4] $V = \{v_1, \ldots, v_n\}$ connected by edges $E = \{e_{i,j} \mid \text{edge between } v_i \text{ and } v_j\}$. The goal is to bisect the graph $\mathcal{G}$ into two complementary sub-graphs

[1] https://github.com/KarypisLab/METIS
[2] For Windows we recommend using the Matlab Online interface, which you can access through your web browser.
[3] This is a common situation in finite volume or discontinuous Galerkin methods.
[4] Depending on the context, vertices may also be called nodes or points.

$\mathcal{G}_1 = (V_1, E_1)$ and $\mathcal{G}_2 = (V_2, E_2)$, that is $V_1 \cup V_2 = V$ and $V_1 \cap V_2 = \emptyset$, of near equal size, $|V_1| \approx |V_2|$, such that the number of edges between the parts,

$$\text{cut}(V_1, V_2) = |\{e_{i,j} \in E \mid v_i \in V_1 \text{ and } v_j \in V_2\}|, \tag{1}$$

is minimized. The latter quantity is also known as the cut-size, edge-cut or cost of the partition.

We represent the dependency graph with the so-called adjacency matrix $\mathbf{A} = (a_{ij})_{1 \le i,j \le n}$ whose elements are defined by

$$a_{ij} = \begin{cases} 1 & \text{there is an edge } e_{i,j} \text{ between } v_i \text{ and } v_j, \\ 0 & \text{otherwise.} \end{cases} \tag{2}$$

Note that for an undirected graph the adjacency matrix is symmetric. The so-called degree of a vertex $\deg(v_i) = \sum_{j=1}^{n} a_{ij}$ is the number of edges that are connected to the vertex v_i. This is encoded in the degree matrix $\mathbf{D} = \text{diag}(d_1, \ldots, d_n)$. With the adjacency and degree matrix, we can form the Laplacian matrix

$$\mathbf{L} = \mathbf{D} - \mathbf{A}. \tag{3}$$

The latter will play an essential role in the spectral bisection algorithm presented below.

As an example, let us consider again the mesh and corresponding dependency graph in Fig. 1. The adjacency and degree matrices are given by

$$\mathbf{A} = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{pmatrix}, \quad \mathbf{D} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 3 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 3 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

and the Laplacian matrix

$$\mathbf{L} = \mathbf{D} - \mathbf{A} = \begin{pmatrix} 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 3 & -1 & 0 & 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & -1 & 2 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 3 & -1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 3 & -1 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 3 & -1 & 0 & -1 \\ 0 & -1 & 0 & 0 & 0 & 0 & -1 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 1 \end{pmatrix}.$$

This can easily be formalized to an arbitrary number of partitions p and weighted vertices and edges. See Elsner [5] for details.

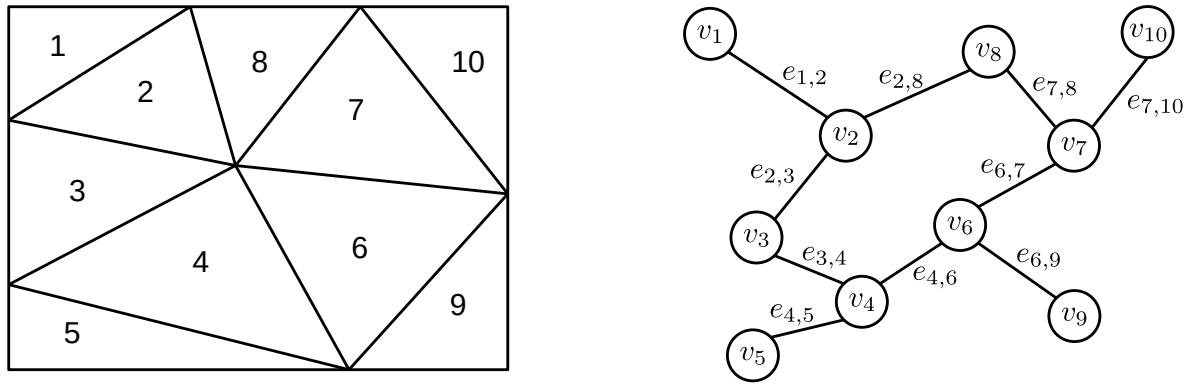
1.2 Graph partitioning: Simple algorithms

1.2.1 Partitioning with geometric information: Coordinate and inertial bisection

We start with two bisection algorithms that assume that the geometric layout of the graph is known. This is for instance the case in our application of domain decomposition of a given mesh.

Coordinate bisection

Coordinate bisection simply consists of finding the hyperplane orthogonal to a coordinate axis that divides the graph vertices in two (almost) equal parts with the smallest edge-cut. This is achieved by computing the median $\overline{x}_i$ over each coordinate axis x_i (e.i., $x_1 \equiv x$ and $x_2 \equiv y$), that is $\overline{x}_i$ separates all vertices of the graph in two with half having x_i coordinates less and half larger than $\overline{x}_i$. Then one computes the edge-cut for each of the coordinate axes and bisects along the one with the smallest edge-cut. This algorithm is summarized in Algorithm 1 for two dimensions.



(a) (b)

Figure 1: Mesh with 10 (simplex/triangle) cells (left) and corresponding dependency graph (right).

Coordinate bisection is a very simple bisection algorithm. However, the quality of the resulting partition may depend strongly on the coordinate systems (e.g., a simple coordinate axes rotation may lead to very different results). We refer to Elsner [5] for further information.

Algorithm 1 Coordinate bisection.

Require: $\mathcal{G}(V, E), P_i = (x_i, y_i)$, $i = 1, \ldots, n$ the coordinates of the vertices
Ensure: A bisection of $\mathcal{G}$
1: **function** InertialBisection(graph $\mathcal{G}(V, E), P_i$)
2: For each coordinate axis, x and y, find the median value $\overline{x}$ and $\overline{y}$ that divides the vertices in two equal parts.
3: Partition the vertices of the graph around median coordinate line having the smallest edge-cut.
4: **return** V_1, V_2 ▷ bisection of $\mathcal{G}$
5: **end function**

Inertial bisection

Inertial bisection improves upon the axes dependence of coordinate bisection by choosing the dividing hyperplane orthogonal along a direction that runs through the center of mass of the vertices. In two dimensions, such a line $\mathbf{L}$ is chosen such that the sum of squares of the distance of the vertices to the line is minimized. It is defined by a point $\overline{P} = (\overline{x}, \overline{y})$ and a unit vector $\mathbf{u} = [u_1, u_2]^T$ with $\|\mathbf{u}\|_2 = \sqrt{u_1^2 + u_2^2} = 1$ such that $\mathbf{L} = \{\overline{P} + \alpha \mathbf{u} \,|\, \alpha \in \mathbb{R}\}$ (see Fig. 2). We set

$$\overline{x} = \frac{1}{n} \sum_{i=1}^{n} x_i, \quad \overline{y} = \frac{1}{n} \sum_{i=1}^{n} y_i, \tag{4}$$

as the center of mass lies on the line L. Finally, we need to compute $\mathbf{u}$ in order to minimize the sum of distances

$$\begin{aligned} \sum_{i=1}^{n} d_i^2 &= \sum_{i=1}^{n} (x_i - \overline{x})^2 + (y_i - \overline{y})^2 - (u_1(x_i - \overline{x}) + u_2(y_i - \overline{y}))^2 \\ &= u_2^2 \sum_{i=1}^{n} (x_i - \overline{x})^2 + u_1^2 \sum_{i=1}^{n} (y_i - \overline{y})^2 + 2u_2 u_1 \sum_{i=1}^{n} (x_i - \overline{x})(y_i - \overline{y}) \\ &= \mathbf{u}^T \begin{bmatrix} S_{yy} & S_{xy} \\ S_{xy} & S_{xx} \end{bmatrix} \mathbf{u} = \mathbf{u}^T M \mathbf{u}, \end{aligned} \tag{5}$$

where S_{xx}, S_{xy}, S_{yy} are the sums as defined in the previous line. The resulting matrix M is symmetric, thus the minimum of Eq. (5) is achieved by choosing $\mathbf{u}$ to be the normalized eigenvector corresponding to the smallest eigenvalue of M. The procedure of bisecting a graph using inertial partitioning is summarized in Algorithm 2. We refer again to Elsner [5] and references therein for a thorough overview of the inertial bisection algorithm.

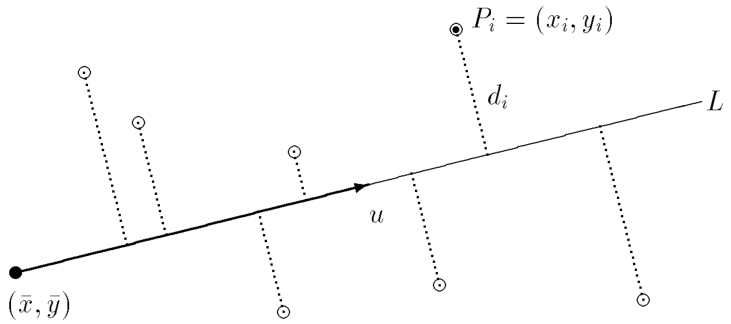


Figure 2: Illustration of inertial bisection in 2D (from Elsner [5]).

Algorithm 2 Inertial bisection.

Require: $\mathcal{G}(V, E), P_i = (x_i, y_i)$, $i = 1, \ldots, n$ the coordinates of the vertices
Ensure: A bisection of $\mathcal{G}$
1: **function** InertialBisection(graph $\mathcal{G}(V, E), P_i$)
2: Calculate the center of mass of the points ▷ acc. to (4)
3: Compute the eigenvector associated with the smallest eigenvalue of $\mathbf{M}$ ▷ $\mathbf{M}$ acc. to (5)
4: Partition the vertices of the graph around line L.
5: **return** V_1, V_2 ▷ bisection of $\mathcal{G}$
6: **end function**

1.2.2 Partitioning without geometric information: Spectral bisection

Spectral bisection was initially considered the standard for solving graph partitioning problems. Unlike the previously introduced bisection algorithms, it does not use any geometric information associated with the graph to partition. A bisection is computed using the eigenvector associated with the second smallest eigenvalue of the Laplacian matrix L of the graph. The graph Laplacian L is a symmetric positive semi-definite matrix[5], with its smallest eigenvalue being $\lambda^{(1)} = 0$, and the associated eigenvector $\mathbf{u}^{(1)} = c\mathbf{1}$ being the constant one. The eigenvector $\mathbf{u}^{(2)}$ associated with the second smallest eigenvalue $\lambda^{(2)}$ is Fiedler's celebrated eigenvector, and is crucial for spectral graph partitioning. Each node v_i of the graph is associated with one entry in $\mathbf{u}^{(2)}$. Thresholding the values of $\mathbf{u}^{(2)}$ around 0 results in two roughly balanced (equal sized) partitions, with minimum edge-cut, while thresholding around the median value of $\mathbf{u}^{(2)}$ produces two strictly balanced partitions. The procedure to compute a bisection using spectral partitioning is summarized in Algorithm 3. For a much more detailed description of the algorithm, we again refer to Elsner [5].

Algorithm 3 Spectral bisection

Require: $\mathcal{G}(V, E)$,
Ensure: A bisection of $\mathcal{G}$
1: **function** SpectralBisection(graph $\mathcal{G}(V, E)$)
2: Form the graph Laplacian matrix $\mathbf{L}$
3: Calculate the second smallest eigenvalue $\lambda^{(2)}$ and its associated eigenvector $\mathbf{u}^{(2)}$.
4: Set 0 or the median of all components of $\mathbf{u}^{(2)}$ as threshold ϵ.
5: Choose$V_1 := \{v_i \in V | u_i < \epsilon\}$, $V_2 := \{\{v_i \in V | u_i \geq \epsilon\}$.
6: **return** V_1, V_2 ▷ bisection of $\mathcal{G}$
7: **end function**

1.3 Graph partitioning: Recursive bisection

A simple and robust algorithm to create a $p = 2^l$ partition, with l being an integer, is recursive bisection and it is presented in Algorithm 4. It uses the recursive function `Recursion` that takes as inputs the part C' of the graph to partition, the number of parts p' into which we will partition C', and the index (an integer) of the first part of the partition of C' into the final partitioning result $\mathcal{G}_p$.

[5] According to the spectral theorem of linear algebra the Laplacian matrix L therefore possesses an orthogonal basis of eigenvectors $\mathbf{u}^{(i)}$ and corresponding real eigenvalues $\lambda^{(1)}$.

Algorithm 4 Recursive bisection.

Require: $\mathcal{G}(V, E)$,
Ensure: p-way partition of $\mathcal{G}$
$\mathcal{G}_p = \{C_1, \ldots, C_p\}$
$p = 2^l$ ▷ p is a power of 2
function RecursiveBisection(graph $\mathcal{G}(V, E)$, number of parts p)
  function Recursion(C', p', index)
    if p' is even **then** ▷ If p' is even, a bisection is possible
      $p' \leftarrow \frac{p'}{2}$
      $(C'_1, C'_2) \leftarrow$ `bisection`(C')
      Recursion(C'_1, p',index)
      Recursion(C'_2, p',index+p')
    else ▷ No more bisection possible, C' is in a partition of $\mathcal{G}$
      $C_{index} \leftarrow C'$
    end if
  end function
  Recursion($C, p, 1$)
  return $\mathcal{G}_p$ ▷ $\mathcal{G}_p$ is a partition of $\mathcal{G}$ in $p = 2^l$ parts
end function

2 Graph partitioning with Matlab: Exercises [85 points]

The goal of this project is to familiarize yourself with various algorithms for graph partitioning applied to domain decomposition. We have chosen Matlab as our tool due to its high-level, dynamic capabilities. For more information about the Matlab, please refer to the Matlab documentation. The `Part_Toolbox`, available on the iCorsi webpage, offers a comprehensive graph partitioning toolbox. It provides code for the creating various graph structures and partitioning methods, such as coordinate bisection. Additionally, it includes routines for generating recursive multiway partitions and visualizing partitioning results.

2.1 METIS and Matlab mex installation

Use the `addpath` command in Matlab to set the path directly to the binary location. If you prefer to compile the package yourself (e.g., for a different OS), follow the instructions at https://github.com/dgleich/metismex, which cover installing METIS 5.0.2 and building a `mex` interface with Matlab using `cmake`.

Once built, place your Matlab interface file `metismex.mexa64` in the `Part_Toolbox` directory for mesh partitioning with METIS. Alternatively, use `addpath` to specify the path to the interface by running the code snippet below:

```
>> A = blkdiag(ones(5),ones(5));
>> A(1,10) = 1; A(10,1) = 1; A(5,6) = 1; A(6,5) = 1;
>> [p1,p2] =  bisection_metis(sparse(A),0,0)
p1 =
1     2     3     4     5
p2 =
6     7     8     9    10
```

2.2 Construct adjacency matrices from connectivity data [10 points]

The first programming task in this assignment is to construct adjacency matrices from a collection of Comma Separated Value files (`.csv`), describing the edge structure and the node coordinates. These files are located in `Datasets/Countries_Meshes` and follow the naming convention "CountryName-NumberOfNodes-FileType.csv". The countries considered here are Great Britain, Greece, Norway, Russia, Switzerland and Vietnam. The files describing the adjacency matrices contain a list of the nodes that are connected through an edge, and the files describing the coordinates of the graph contain a list with the x, y coordinates of each node. The resulting graphs correspond to the continental maps of the above-mentioned countries, with the overseas territories excluded since we are interested in connected graphs. Some examples are illustrated in Figure 3.

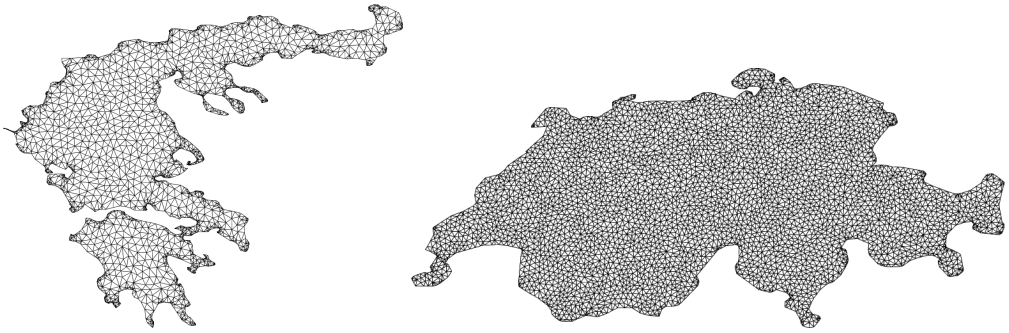


Figure 3: Graphs corresponding to the maps of various countries. Left: Greece with 3117 nodes and 3902 edges. Right: Switzerland with 4468 nodes and 15230 edges.

Run the Matlab script `Source/read_csv_graphs.m` and complete the missing sections of the code:

1. Read the .csv files in Matlab.
2. Construct the adjacency matrix $\mathbf{W} \in \mathbb{R}^{n \times n}$ and the node coordinate list $C \in \mathbb{R}^{n \times 2}$. Note that $\mathbf{W}$ must be symmetric and sparse. [6]
3. Visualize the graphs of Norway and Vietnam using the function `Source/Visualization/gplotg.m`.
4. Save the sparse adjacency matrices and the corresponding coordinates in a new folder, e.g. `/Datasets/Countries_Mat`.

2.3 Implement graph partitioning algorithms [25 points]

1. Run in Matlab the script `Source/Bench_bisection.m` and familiarize yourself with the Matlab codes in the directory `Part_Toolbox`. An overview of all functions and scripts is offered in `Source/Contents.m`.
2. Implement **spectral graph bisection** based on the entries of the Fiedler eigenvector. Use the incomplete Matlab file `Source/bisection_spectral.m` for your solution.
3. Implement **inertial graph bisection**. For a graph with 2D coordinates, this inertial bisection constructs a line such that half the nodes are on one side of the line, and half are on the other. Use the incomplete Matlab file `Source/bisection_inertial.m` for your solution.
4. Report the bisection edgecut for all toy meshes that are loaded in the script `Bench_bisection.m`. Use Table 1 to report these results.

Table 1: Bisection results

Mesh	Coordinate	Metis 5.0.2	Spectral	Inertial
mesh1e1	18			
mesh2e1	37			
mesh3e1				
mesh3em5				
airfoil1				
netz4504_dual				
stufe				
3elt				
barth4				
ukerbe1				
crack				

[6] Check the symmetry of your resulting adjacency matrix with the function `issymmetric.m`. If there is an edge missing and the matrix is non-symmetric, apply the transformation $\mathbf{W}_{\text{sym}} = \frac{\mathbf{W}+\mathbf{W}^T}{2}$.

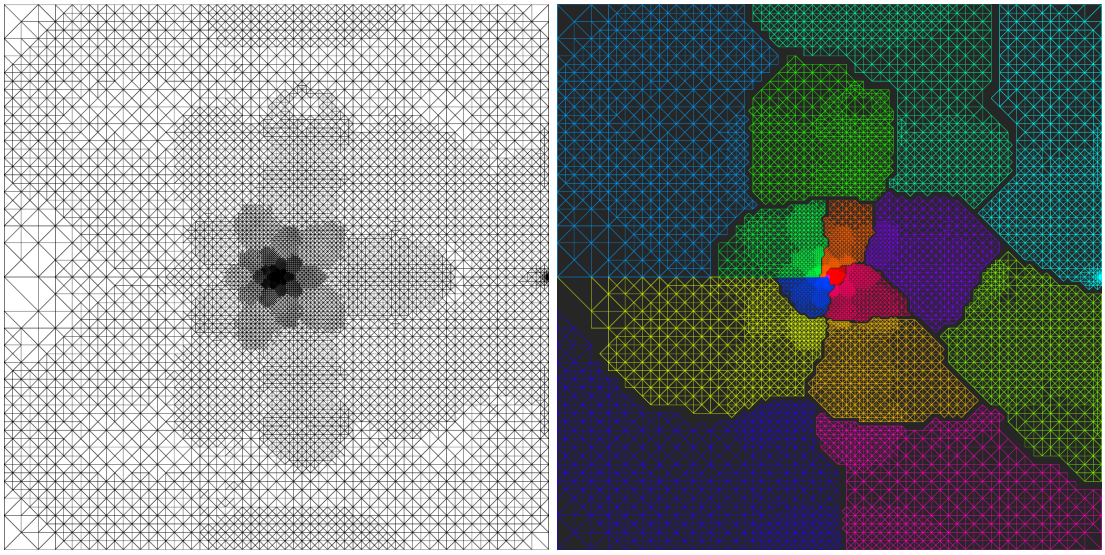


Figure 4: Left: The finite element mesh "crack" with 10240 nodes and 30380 edges. Right: 16-way recursive bisection of the mesh using Metis 5.0.2. Number of cut edges: 1290.

2.4 Recursively bisecting meshes [15 points]

Next, we will partition 2D graphs derived from structural engineering matrices provided by NASA, available in the SuiteSparse Matrix Collection (SSMC) [3]. Algorithm 4 presents a robust approach for creating a 2^l partition, where l is an integer. This algorithm uses the recursive function `Recursion`, which takes as inputs the sub-graph C' to be partitioned, the number of parts p' into which we will partition C', and the starting index of C''s parts in the final partitioning output $\mathcal{G}_p$. In practice, this algorithm exclusively utilizes strongly balanced bisection routines [1].

Algorithm 4 is implemented in the file `rec_bisection.m` of the toolbox. Utilize this function within the script `Bench_rec_bisection.m` to recursively bisect the finite element meshes loaded within the script in 8 and 16 subgraphs. Use your inertial and spectral partitioning implementations, as well as the coordinate partitioning and the METIS bisection routine. Summarize your results in Table 2. Finally, visualize the results for $p = 16$ for the case "crack". An example for spectral recursive partitioning is illustrated in Figure 4.

Table 2: Edge-cut results for recursive bi-partitioning.

Mesh	Coordinate	Metis 5.0.2	Spectral	Inertial
airfoil1				
netz4504_dual				
stufe				
3elt				
barth4				
ukerbe1				
crack				

2.5 Comparing recursive bisection to direct k-way partitioning [10 points]

Recursive bisection is highly dependent on the decisions made during the early stages of the process, and also suffers from the lack of global information. Thus, it may result in suboptimal partitions [11]. This necessitated the development of methods for direct k-way partitioning. Besides recursive bi-partitioning, METIS also employs a multilevel k-way partitioning algorithm. The graph $\mathcal{G} = (V, E)$ is initially coarsened to a small number of vertices, a k-way partitioning of this smaller graph is computed, and then this partitioning is projected back towards the original finer graph by successively refining the partitioning at each intermediate level [6].

We will compare the quality of the cut resulting from the application of recursive bipartitioning and direct multiway partitioning, as implemented in Metis 5.0.2. Our test cases will be the graphs presented in

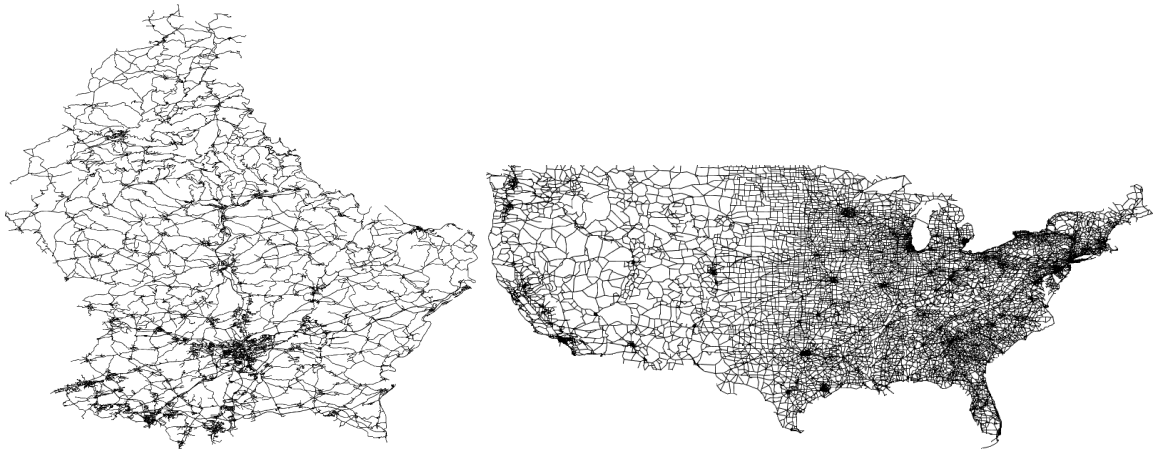


Figure 5: Left: The road network of Luxemburg with 114599 nodes and 119666 edges. Right: A segment of the US road network with 126146 nodes and 161950 edges.

Figure 5. These graphs emerge from the road networks of Luxemburg and the US, with edges representing road segments and node intersections. Graph partitioning is crucial for computing driving directions in such networks, a fundamental problem of practical importance. It can be solved in almost linear time by Dijkstra's shortest-path algorithm, but this is not fast enough for large scale road networks. Various lightweight alternatives have been suggested [4], that use graph partitioning as a preprocessing tool to define the reduced (partitioned) topology of the network. Additionally, we will consider the map-graphs that you created in the second task of this assignment.

Use the incomplete `Source/Bench_metis.m` for your implementation. Compare the cut obtained from Metis 5.0.2 after applying recursive bisection and direct multiway partitioning for the graphs in question. Consult the Metis manual, and type `help metismex` in your Matlab command line to familiarize yourself with the way the Metis recursive and direct multiway partitioning functionalities should be invoked. Summarize your results in Table 3 for 16 and 32 partitions. Comment on your results. Was this behavior anticipated? Visualize the partitioning results for the graphs of i) USA, ii) Luxemburg, and iii) Russia for 32 partitions.

Table 3: Comparing the number of cut edges for recursive bisection and direct multiway partitioning in Metis 5.0.2.

Partitions	Luxemburg	usroads-48	Greece	Switzerland	Vietnam	Norway	Russia
16							
32							

2.6 Utilizing graph eigenvectors [25 points]

Provide the following illustrative results. Use the incomplete script `Source/Bench_eigen_plot.m` for your implementation.

1. Plot the entries of the eigenvectors associated with the first (λ_1) and second (λ_2) smallest eigenvalues of the graph Laplacian matrix $\mathbf{L}$ for the graph "airfoil1." Comment on the visual result. Is this behavior expected?

2. Plot the entries of the eigenvector associated with the second smallest eigenvalue λ_2 of the Graph Laplacian matrix $\mathbf{L}$. Project each solution on the coordinate system space of the following graphs: mesh3e1, barth4, 3elt, crack. An example is shown in Figure 6, for the graph "airfoil1".
   Hint: You might have to modify the functions `gplotg.m` and `gplotpart.m` to get the desired result.

3. In this assignment we dealt exclusively with graphs $\mathcal{G}(V, E)$ that have coordinates associated with their nodes. This is, however, most commonly not the case when dealing with graphs, as they are in fact abstract structures, used for describing the relation E over a collection of entities V. These entities very often cannot be described in a Euclidean coordinate space. Therefore graph drawing is a tool to visualize relational information between nodes. The optimality of graph drawing is

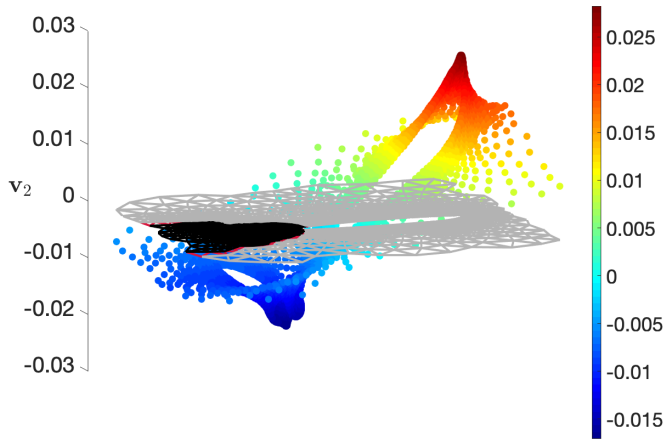


Figure 6: Partitioning the Airfoil graph based on the values of the Fiedler eigenvector. The two partitions are depicted in black and gray, while the cut edges in red respectively. The z-axis represents the value of the entries of the eigenvector.

measured in terms of computation speed the ultimate usefulness of the resulting layout [10]. A successful layout should transmit the clearly the desired message, e.g the subsets of a partitioned graph. We will now see a spectral graph drawing method, which constructs the layout utilizing the eigenvectors of the graph Laplacian matrix $\mathbf{L}$. Draw the graphs mesh3e1, barth4, 3elt, crack, and their **spectral bi-partitioning** results using the eigenvectors to supply coordinates. Locate vertex i at position:

$$x_i = (\mathbf{v}_2(i), \mathbf{v}_3(i)),$$

where $\mathbf{v}_2, \mathbf{v}_3$ are the eigenvectors associated with the 2nd and 3rd smallest eigenvalues of $\mathbf{L}$. Figure 7 illustrates these 2 ways of visualizing the partitions of the "airfoil1" graph.

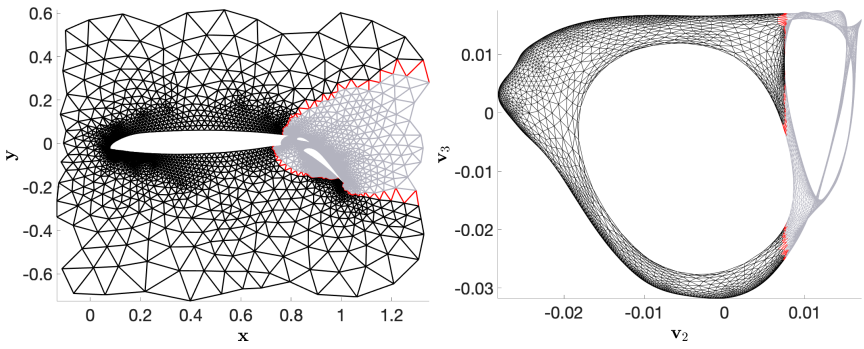


Figure 7: Visualizing the bipartitioning of the graph "airfoil1" with 4253 nodes and 12289 edges. Left: Spatial coordinates. Right: Spectral coordinates.

3 Quality of the Report [15 Points]

Each project will have 100 points (out of 15 point will be given to the general written quality of the report).

Additional notes and submission details

Submit the source code files (together with your used `Makefile`) in an archive file (tar, zip, etc.) and summarize your results and the observations for all exercises by writing an extended Latex report. Use the Latex template from the webpage and upload the Latex summary as a PDF to iCorsi.

- Your submission should be a gzipped tar archive, formatted like project_number_lastname_firstname.zip or project_number_lastname_firstname.tgz. It should contain:
  - All the source codes of your solutions.
  - Build files and scripts. If you have modified the provided build files or scripts, make sure they still build the sources an run correctly. We will use them to grade your submission.
  - project_number_lastname_firstname.pdf, your write-up with your name.
  - Follow the provided guidelines for the report.
- Submit your .tgz through iCorsi.

Code of Conduct and Policy

- Do not use or otherwise access any on-line source or service other than the iCorsi system for your submission. In particular, you may not consult sites such as GitHub Co-Pilot or ChatGPT.
- You must acknowledge any code you obtain from any source, including examples in the documentation or course material. Use code comments to acknowledge sources.
- Your code must compile with a standard-configuration C/C++ compiler.

Please follow these instructions and naming conventions. Failure to comply results in additional work for the TAs, which makes the TAs sad...

References

[1] Charles-Edmond Bichot and Patrick Siarry, editors. *Graph Partitioning*. Wiley, feb 2013. doi: 10.1002/9781118601181.

[2] Aydın Buluç, Henning Meyerhenke, Ilya Safro, Peter Sanders, and Christian Schulz. Recent Advances in Graph Partitioning. Lecture Notes in Computer Science, pages 117–158. Springer International Publishing, Cham, 2016. ISBN 9783319494876. doi: 10.1007/978-3-319-49487-6_4. URL https://doi.org/10.1007/978-3-319-49487-6_4.

[3] Timothy A. Davis and Yifan Hu. The university of florida sparse matrix collection. *ACM Trans. Math. Softw.*, 38(1), December 2011. ISSN 0098-3500. doi: 10.1145/2049662.2049663. URL https://doi.org/10.1145/2049662.2049663.

[4] Daniel Delling and Renato F. Werneck. Faster customization of road networks. In Vincenzo Bonifaci, Camil Demetrescu, and Alberto Marchetti-Spaccamela, editors, *Experimental Algorithms*, pages 30–42, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg. ISBN 978-3-642-38527-8.

[5] Ulrich Elsner. Graph partitioning - a survey. Technical report, Technische Universität Chemnitz, September 2005. URL https://nbn-resolving.org/urn:nbn:de:swb:ch1-200501047.

[6] G. Karypis and V. Kumar. Parallel Multilevel k-way Partitioning Scheme for Irregular Graphs. In *Supercomputing '96:Proceedings of the 1996 ACM/IEEE Conference on Supercomputing*, pages 35–35, January 1996. doi: 10.1109/SUPERC.1996.183537.

[7] G. Karypis and V. Kumar. Multilevel Algorithms for Multi-Constraint Graph Partitioning. In *SC '98: Proceedings of the 1998 ACM/IEEE Conference on Supercomputing*, pages 28–28, November 1998. doi: 10.1109/SC.1998.10018. URL https://ieeexplore.ieee.org/document/1437315.

[8] George Karypis and Vipin Kumar. Multilevel$k$-way Partitioning Scheme for Irregular Graphs. *Journal of Parallel and Distributed Computing*, 48(1):96–129, January 1998. ISSN 0743-7315. doi: 10.1006/jpdc.1997.1404. URL https://www.sciencedirect.com/science/article/pii/S0743731597914040.

[9] George Karypis and Vipin Kumar. A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs. *SIAM Journal on Scientific Computing*, 20(1):359–392, January 1998. ISSN 1064-8275. doi: 10.1137/S1064827595287997. URL https://epubs.siam.org/doi/abs/10.1137/S1064827595287997.

[10] Y. Koren. Drawing graphs by eigenvectors: theory and practice. *Computers & Mathematics with Applications*, 49(11):1867–1888, 2005. ISSN 0898-1221. doi: https://doi.org/10.1016/j.camwa.2004.08.015. URL https://www.sciencedirect.com/science/article/pii/S089812210500204X.

[11] Horst D. Simon and Shang-Hua Teng. How good is recursive bisection? *SIAM Journal on Scientific Computing*, 18(5):1436–1445, 1997. doi: 10.1137/S1064827593255135. URL https://doi.org/10.1137/S1064827593255135.

Università della Svizzera italiana

Institute of Computing
CI

High-Performance Computing Lab **Institute of Computing**

Student: Dennys Mike Huber Discussed with: Alberto Finardi & Leon Ackermann

Solution for Project 6

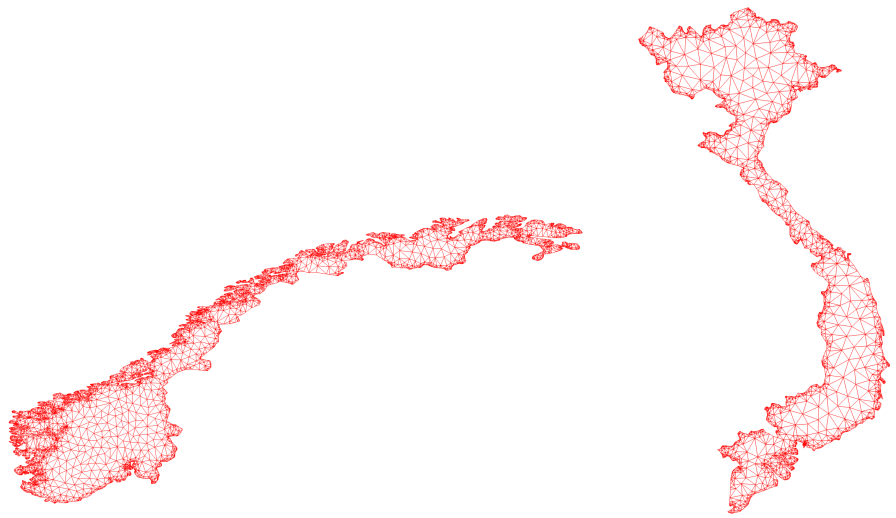
In this project, we will investigate the graph partitioning problem within the context of domain decomposition for high-performance computing (HPC) applications.

1. Construct adjacency matrices from connectivity data [10 points]

For this exercise the goal was to read the csv file of each graph and create a sparse adjacency matrix which must symmetric and sparse. To ensure the first property we applied the code seen in Listing 1, where we check if the matrix is symmetric and if not do the necessary adjustment. To ensure the second property we use the `sparse` function to create a sparse matrix.

```
W = sparse(from,to, 1, nodes, nodes);
if ~issymmetric(W)
  W = (W+W')/2;
  disp('The adjacency matrix has been symmetrized.');
end
```

Listing 1: Ensure symmetric property



(a) Norway (b) Vietnam

Figure 1: Country graphs

After creating the adjacency matrix we save it and the coordinates matrix in folder `./Datasets/Countries_Ma`
As the final step we load the matrices for Norway and Vietnam and visualize it with the provided `gplotg` function. The resulting graphs can be seen in Figure 1.

2. Implement graph partitioning algorithms [25 points]

For this exercise the task was to Implement two different graph bisection algorithms. The first algorithm can be seen in Listing 2 and is called Spectral bisection. The algorithm leverages the spectral properties of the Laplacian Matrix, which can be computed by $L = D - A$, where D is the degree matrix and A is the adjacency matrix. Subsequently, we use this Laplacian matrix to compute its eigen values and eigen vectors. In Matlab we can use the `eigs` function to that for sparse matrices. More precisely we are interested in the second smallest eigenvector, also commonly referred to as Fiedler vector. Using the median of the Fiedler vector we then assign the vertices to one of the partitions depending on if the corresponding entry in the Fiedler vector is smaller or bigger than the median.

```
function [part1,part2, eig_vec] = bisection_spectral(A,xy,picture)

  % 1. Construct the Laplacian.
  D = diag(sum(A, 2));
  L = D - A;
  % 2. Calculate its eigensdecomposition.
  [eig_vec,~] = eigs(L,3,1e-10);
  % 3. Label the vertices with the components of the Fiedler vector.
  fiedler_vec = eig_vec(:,2);
  % 4. Partition them around their median value, or 0.
  fiedler_med = median(fiedler_vec);
  n = size(A,1);
  map = zeros(n,1);
  map(fiedler_vec <= fiedler_med) = 0;
  map(fiedler_vec > fiedler_med) = 1;
  [part1, part2] = other(map);

  if picture == 1
    gplotpart(A,xy,part1);
    title('Spectral bisection (actual) using the Fiedler Eigenvector');
  end
end
```

Listing 2: Spectral graph bisection implementation

The second algorithm that we Implemented was inertial bisection 3, which leverages geometric properties to subdivide the graph into two subpartitions. Inertial bisection does this by computing the center mass in both x and y. Then the matrix M is constructed to reflect the distribution of points relative to the center of mass. This Matrix M is then used to compute its own smallest normalized eigen vector, which represents the direction of minimum variance. This eigen vector is used to define a line that passes through the center of mass. This line is then used to partition the vertices into two disjointed groups.

```
function [part1,part2] = bisection_inertial(A,xy,picture)
  % Steps
  % 1. Calculate the center of mass.
  xy_mean = mean(xy, 1);
  x_mean = xy_mean(1);
  y_mean = xy_mean(2);

  % 2. Construct the matrix M.
  x_shift = xy(:,1) - x_mean;
  y_shift = xy(:,2) - y_mean;
  Sxx = sum(x_shift.^2);
  Syy = sum(y_shift.^2);
  Sxy = sum(x_shift.*y_shift);
  M = [Syy, Sxy; Sxy, Sxx];

  % 3. Calculate the smallest eigenvector of M.
  [eig_vec,~] = eigs(M,1,1e-10);

  % 4. Find the line L on which the center of mass lies.
```

```
  eig_vec = eig_vec/norm(eig_vec);

  % 5. Partition the points around the line L.
  [part1, part2] = partition(xy, eig_vec);

  if picture == 1
    gplotpart(A,xy,part1);
    title('Inertial bisection (actual) using the Fiedler Eigenvector');
  end
end
```

Listing 3: Inertial graph bisection implementation

As a final step we compare the newly implemented algorithms to the Coordinate bisection and Metis bisection methods using a variety of different mashes. The goal is to figure out, which algorithms creates the lowest number of edge-cuts to create the two partitions. The results in Table 1 clearly show that there is not a clear algorithm that always creates the lowest number of edge-cuts and the choice of bisection algorithm has to be made on an individual problem basis.

Mesh	Coordinate	Metis 5.0.2	Spectral	Inertial
mesh1e1	18	18	18	20
mesh2e1	37	34	36	47
mesh3e1	19	20	18	19
mesh3em5	19	20	24	19
airfoil1	94	93	132	93
netz4504_dual	25	19	23	27
stufe	16	17	16	16
3elt	172	96	117	257
barth4	206	111	127	208
ukerbe1	32	36	32	28
crack	353	220	233	384

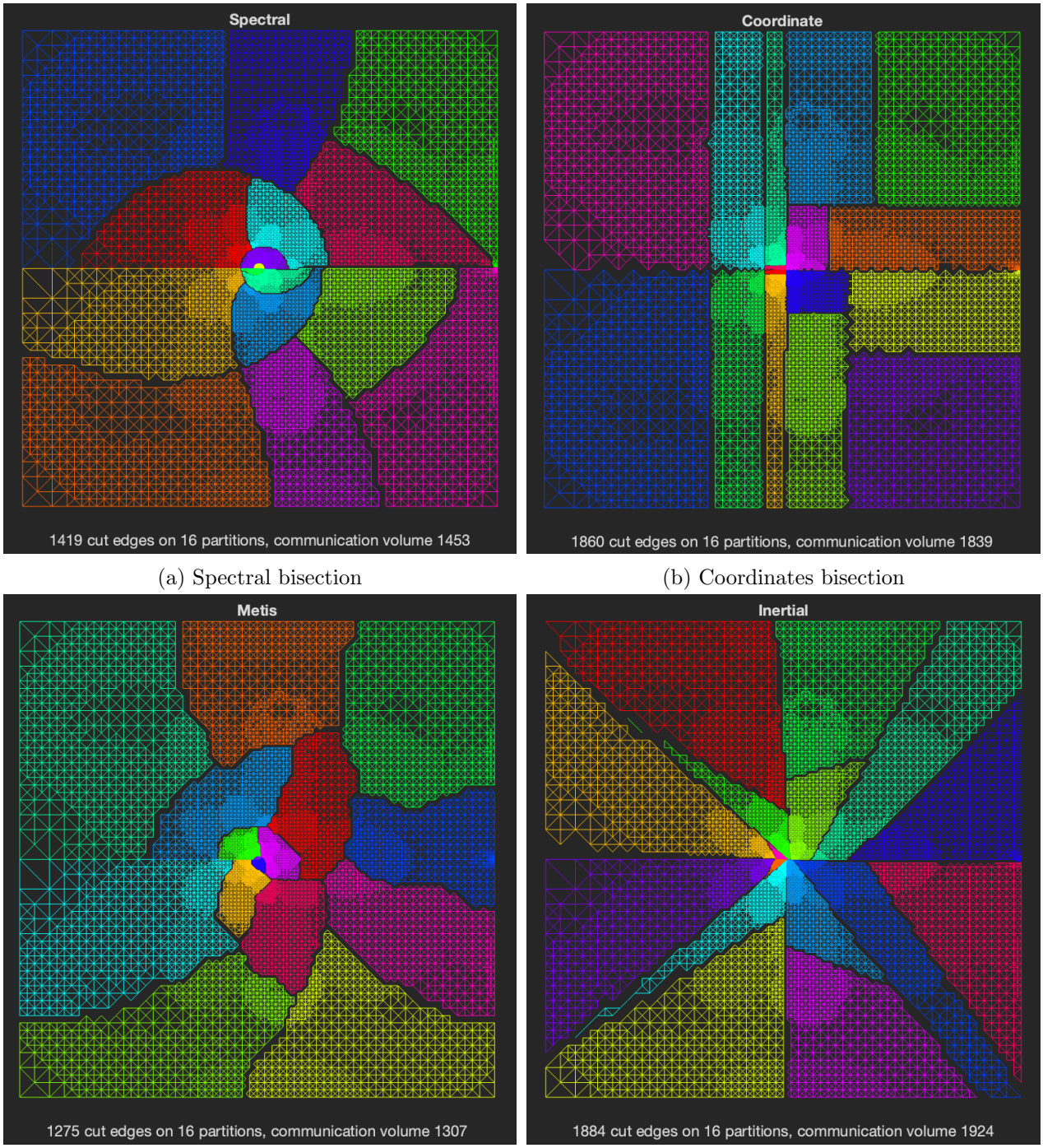
Table 1: Bisection Results

3. Recursively bisecting meshes [15 points]

In this exercise we recursively partition 2D graphs derived by structural engineering matrices provided by Nasa. For that we use the provided function `Recursion`, which takes a bisection algorithm and graph as arguments and then partitions the graph into 2^l subpartitions. In our case we chose l to be 3 and 4 to create 8 and 16 partitions respectively. The resulting edge-cuts for both partition sizes can be found in table 2. Based on said table we can observe that Metis consistently produces the smallest edge cut for both 8 and 16 partitions, outperforming the other mesh cut methods. Furthermore an example for the visual result of different algorithms for the mesh `crack` using 16 partitions can be seen in Figure 2.

Mesh	Edge Cut (8 Partitions)				Edge Cut (16 Partitions)			
	Spectral	Metis 5.0.2	Coordinate	Inertial	Spectral	Metis 5.0.2	Coordinate	Inertial
airfoil1	397	318	516	670	629	561	819	1081
netz4504_dual	112	96	127	165	183	159	198	271
stufe	129	108	123	320	246	193	227	606
3elt	469	418	733	814	752	699	1168	1230
barth4	549	470	875	977	835	743	1306	1492
ukerbe1	781	147	225	340	888	245	374	499
crack	883	808	1343	1351	1419	1275	1860	1884

Table 2: Edge cut results for recursive bi-partitioning using 8 and 16 partitions.



(a) Spectral bisection

(b) Coordinates bisection

(c) Metis bisection

(d) Inertial Bisection

Figure 2: Recursive bisecting of mesh "crack" into 16 partitions with different algorithms.

4. Comparing recursive bisection to direct k-way partitioning [10 points]

In this exercise the goal is to compare and analyze the performance of recursive bisection and direct k-way partitioning by looking at the edge cut result. Both methods are using Metis and the goal is to partition the graphs into 16 (Table 3) and 32 (Table 4) partitions. Comparing the two methods we can clearly see that that direct k-way partitioning consistently outperforms the recursive bisection by creating a smaller edge cut. This is the case because recursive bisection has no global information when doing partitioning decisions, k-way on the other hand overcomes this limitation by coarsening the graph first and the then refining the partitioning. Knowing this coarsening and refinement steps it was anticipated that k-way partitioning would outperform recursive bisection. Another observation that can be made is that the difference in edge cut between the methods increases when comparing 16 to 32 partitions. Indicating that k-way multiway partitioning also scales better in terms of number of partitions.

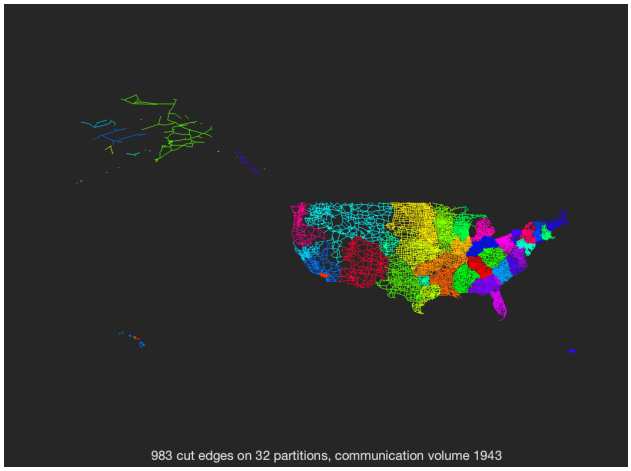
Partitions	Luxemburg	USRoads	Greece	Switzerland	Vietnam	Norway	Russia
16	191	585	318	685	270	271	572
32	317	983	500	1067	445	509	941

Table 3: Cut edges for recursive bisection.

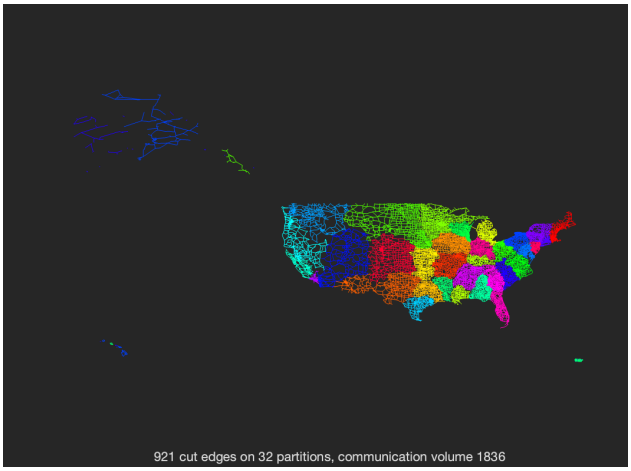
Partitions	Luxemburg	USRoads	Greece	Switzerland	Vietnam	Norway	Russia
16	185	545	301	665	231	241	551
32	304	921	486	1013	418	436	931

Table 4: Cut edges for direct multiway partitioning in Metis 5.0.2.

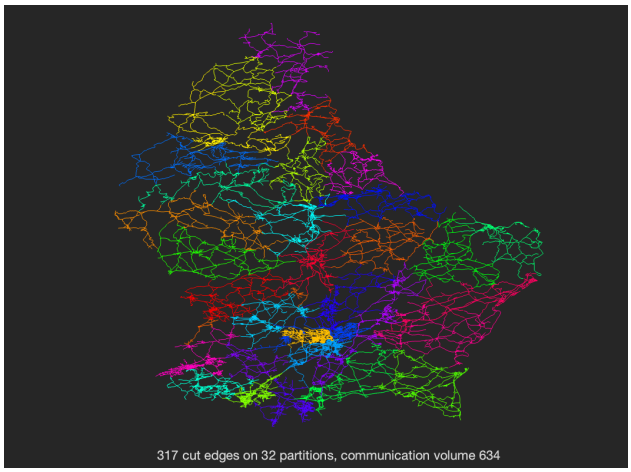
Figure 3 showcases how various examples of graphs are partitioned differently when using recursive or k-way partitioning.



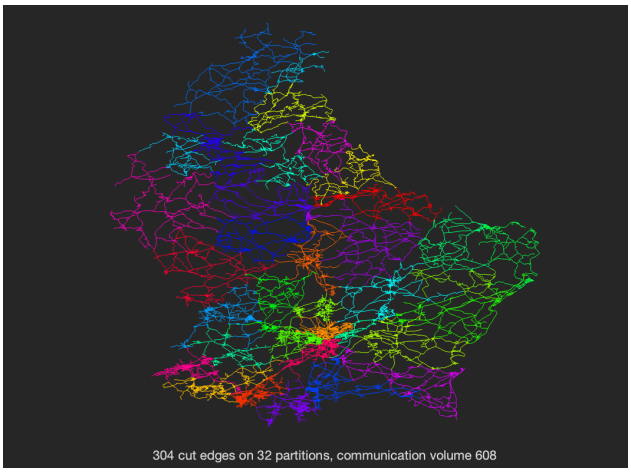
(a) USA with recursive bisection



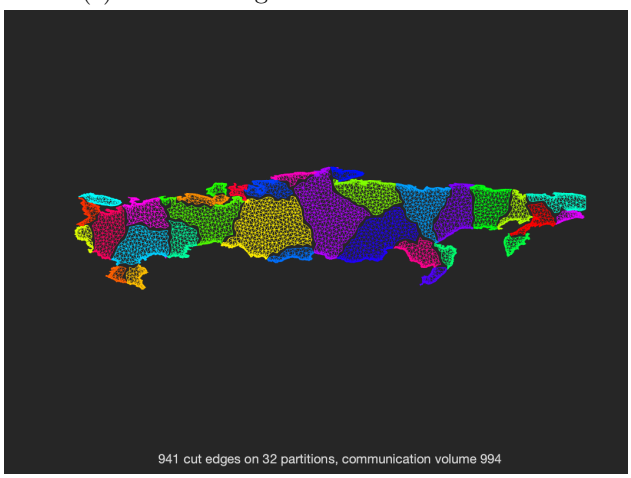
(b) USA with direct multiway partitioning



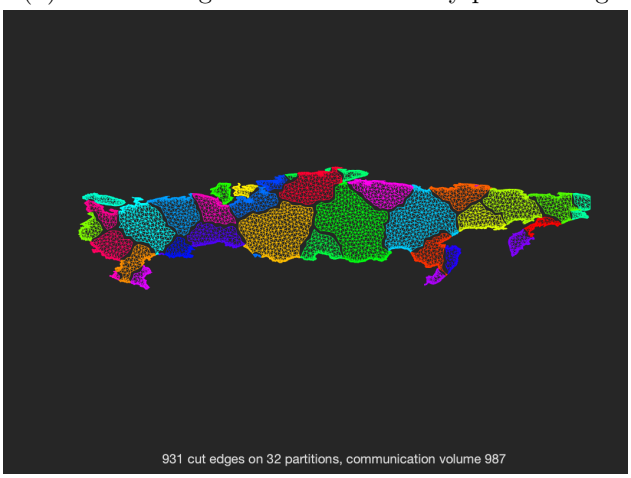
(c) Luxembourg with recursive bisection



(d) Luxembourg with direct multiway partitioning



(e) Russia with recursive bisection

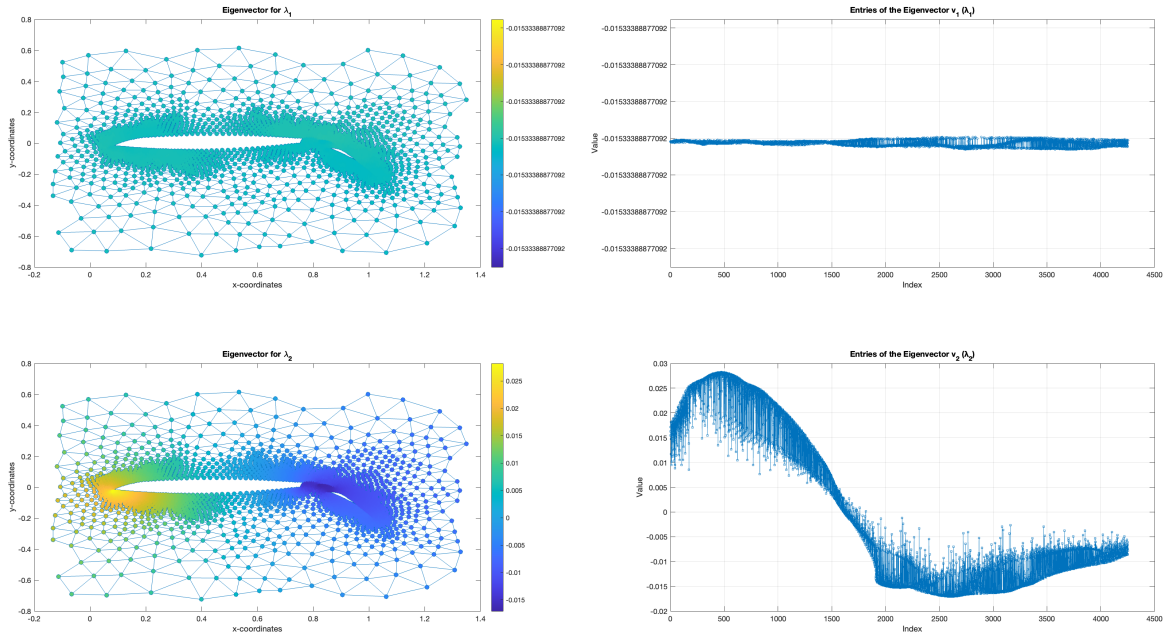


(f) Russia with direct multiway partitioning

Figure 3: Partitioning results for the graphs of USA, Luxemburg, and Russia for 32 partitions

5. Utilizing graph eigenvectors [25 points]

The entries of the eigenvector associated with the first (λ_1) and second (λ_2) smallest of the Laplacian L are plotted in Figure 4. The behaviour of the λ_1's eigenvector v_1 is expected to have only 1's as its elements due to the property of the L. In our case the entries are not 1 but the normalized vector which is $\frac{v_1}{\sqrt{n}}$, where n is the length of the eigenvector [1]. The Fiedler vector v_2 of the second smallest eigen value reflects the structure of the graph. Vertices are strongly connected will have similar eigenvector values, while vertices will show more distinct values, when they are further apart. When looking at plot for v_2 in the Subfigure 4b you can see that the vector shows a smooth transition from positive to negative indicating that they are transitioning from one partition to another. Subfigure 4a then shows this grouping using coloring of the vertices.



(a) The vertices colored according to their corresponding eigenvector element

(b) The entries of the eigenvector plotted against their indices.

Figure 4: Entries of the eigenvector associated with the first (λ_1) and second (λ_2) smallest eigenvector.

As the next step we project each solution on the coordinate system space for the graph mesh3e1, barth4, 3elt and crack The result of this can be seen in Figure 5. Looking at the shape of the values of the fiedler vector we can see that the for 3elt and barth4, they are concentrated in a vertical transition. While for the other two, they have more of a planar structure, creating a clear horizontal separation. Furthermore all graphs appear to have a smooth transition, seen by the transition of color from dark blue to red.

As the final step show the graph with its partitions and edge cut by not using the coordinate space, but by using the eigenvectors values of the graph Laplacian. In Figure 6 one can see the partitioning results in two ways. On the top using the eigenvector values and on the bottom the Euclidean coordinates. For the barth4 mesh for example in Subfigure 6b we are able to visualize the relation between nodes a lot more effectively using the eigenvector values as our coordinates, in comparison to the cartesian version, where the groups are barely visible.

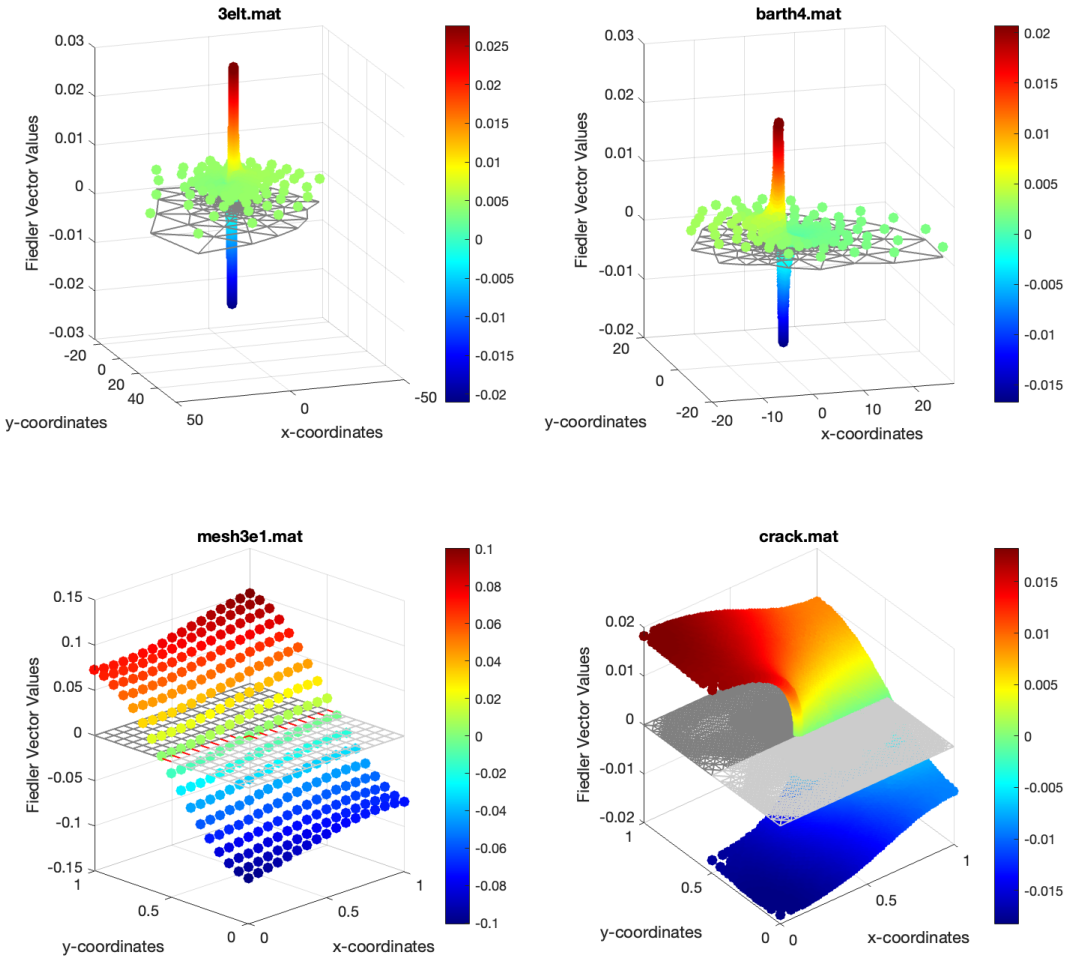
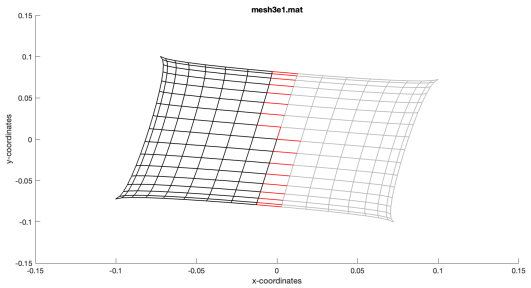
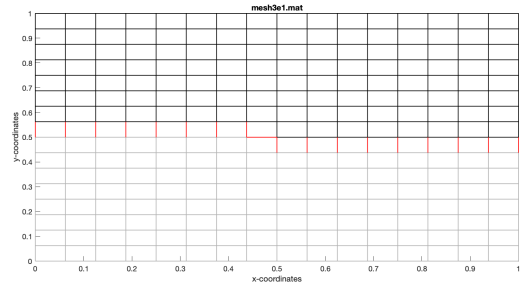
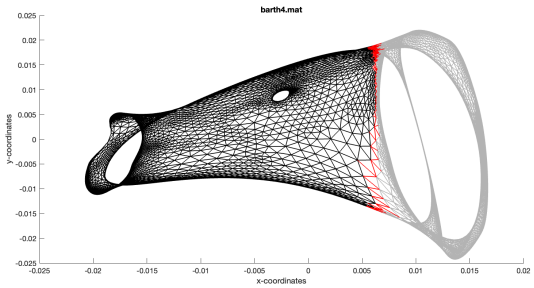


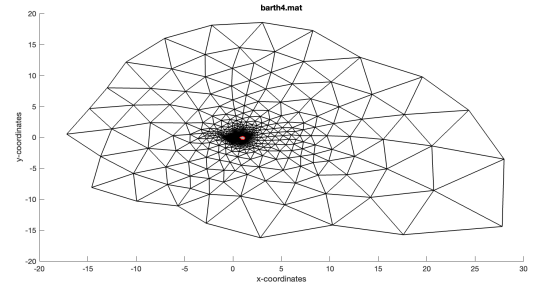
Figure 5: Entries of the eigenvector associated with the second smallest eigenvalue λ_2 of the Graph Laplacian matrix L. The two partitions are depicted in dark gray and light gray, while the cut edges is red respectively. The z-axis represents the value of the entries of the eigenvector.



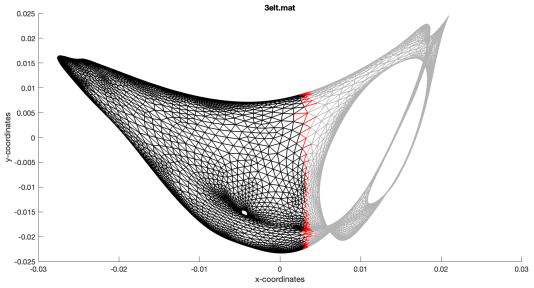


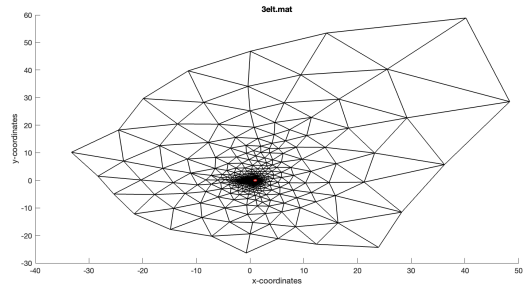
(a) mesh3e1



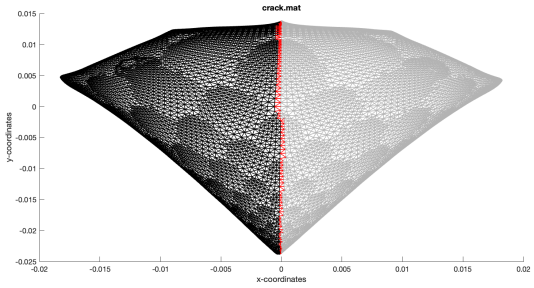


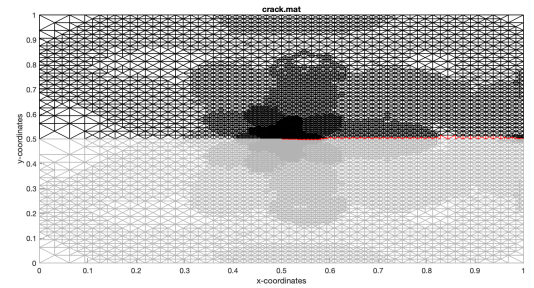
(b) barth4





(c) 3elt





(d) crack

Figure 6: Spectral bi-partitioning results using the eigenvectors to supply coordinates (top) and using cartesian coordinates (bottom).

References

[1] *The Smallest Eigenvalues of a Graph Laplacian*. URL: `http://blog.shriphani.com/2015/04/06/the-smallest-eigenvalues-of-a-graph-laplacian/` (visited on 12/16/2024).