Università della Svizzera italiana
Institute of Computing CI

High-Performance Computing, Fall 2025
Lecturer: Dr. A. Eftekhari
Assistants: V. Karpenko, L. Migliari, D. Vega

Project 6
Large Scale Graph Partitioning

Due date: See iCorsi submission

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` or `metismex.mexmaca64`) for Mac (Intel and Apple silicon respectively), a precompiled version for Linux (`metismex.mexa64`, Ubuntu, glibc 2.27) and a precompiled version for Windows (`metismex.mexw64`)[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 $\mathcal{G}_1 = (V_1, E_1)$ and $\mathcal{G}_2 = (V_2, E_2)$, that is $V_1 \cup V_2 = V$ and

[1] https://github.com/KarypisLab/METIS
[2] In case of installation problems we reccomend 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.

$V_1 \cap V_2 = \emptyset$, of near equal size, $|V_1| \approx |V_2|$, such that the number of edges between the parts,

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

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 $\mathbf{D} = \mathrm{diag}(d_1, \ldots, d_n)$. 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.

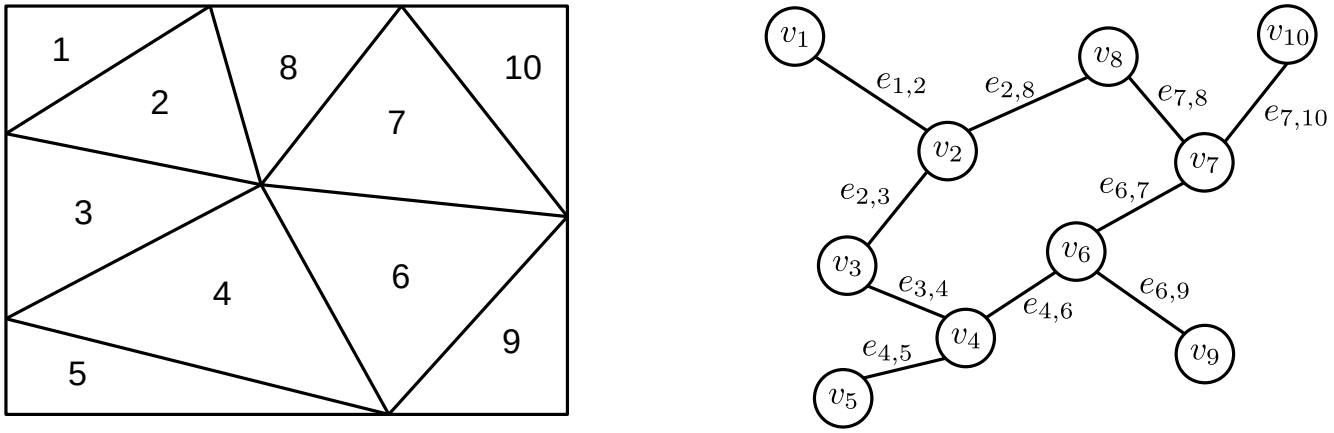
1.2 Graph partitioning: Simple algorithms

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



(a) (b)

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

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.
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

$$\begin{aligned}\sum_{i=1}^{n} d_i^2 &= \sum_{i=1}^{n} (x_i - \overline{x})^2 + (y_i - \overline{y})^2 - \left(u_1(x_i - \overline{x}) + u_2(y_i - \overline{y})\right)^2 \\ &= u_2^2 \sum_{i=1}^{n} (x_i - \overline{x})^2 + u_1^2 \sum_{i=1}^{n} (y_i - \overline{y})^2 + 2u_2u_1 \sum_{i=1}^{n} (x_i - \overline{x})(y_i - \overline{y}) \\ &= \mathbf{u}^T \begin{bmatrix} S_{yy} & S_{xy} \\ S_{xy} & S_{xx} \end{bmatrix} \mathbf{u} = \mathbf{u}^T M \mathbf{u},\end{aligned} \tag{5}$$

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.

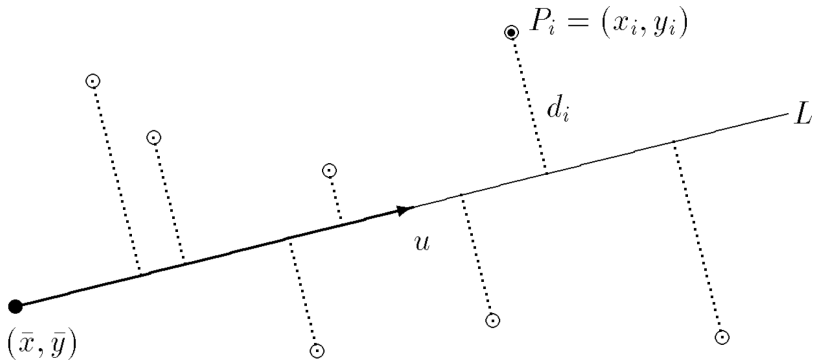


Figure 2: Illustration of inertial bisection in 2D (from Elsner [5]).

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].

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 3 Spectral bisection

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

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
```

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.

```
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.
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.

[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 $\mathbf{W}_{\text{sym}} = \frac{\mathbf{W}+\mathbf{W}^T}{2}$.

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				

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_bisectio` 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. These graphs emerge from the road networks of Luxemburg and the US, with edges representing road segments and node intersections. Graph partitioning is crucial

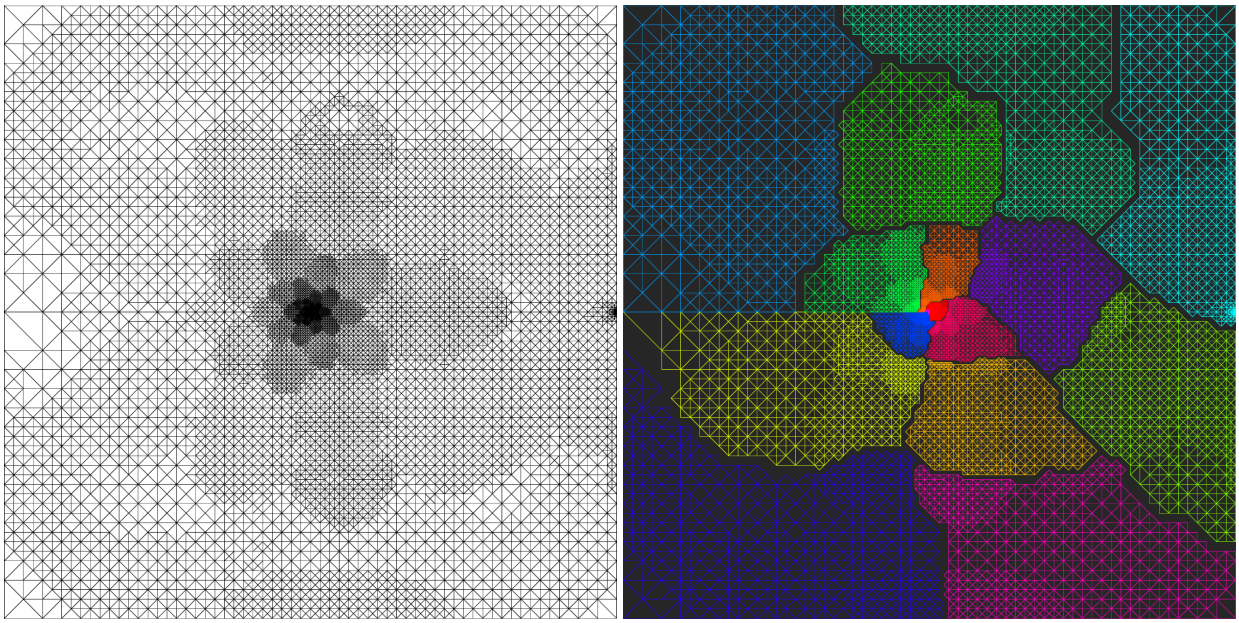


Figure 4: Left: The finite element mesh "crack" with 10240 nodes and 30380 edges. Right: 16-way recursive bisection of the mesh using Metis 5.0.2. Number of cut edges: 1290.

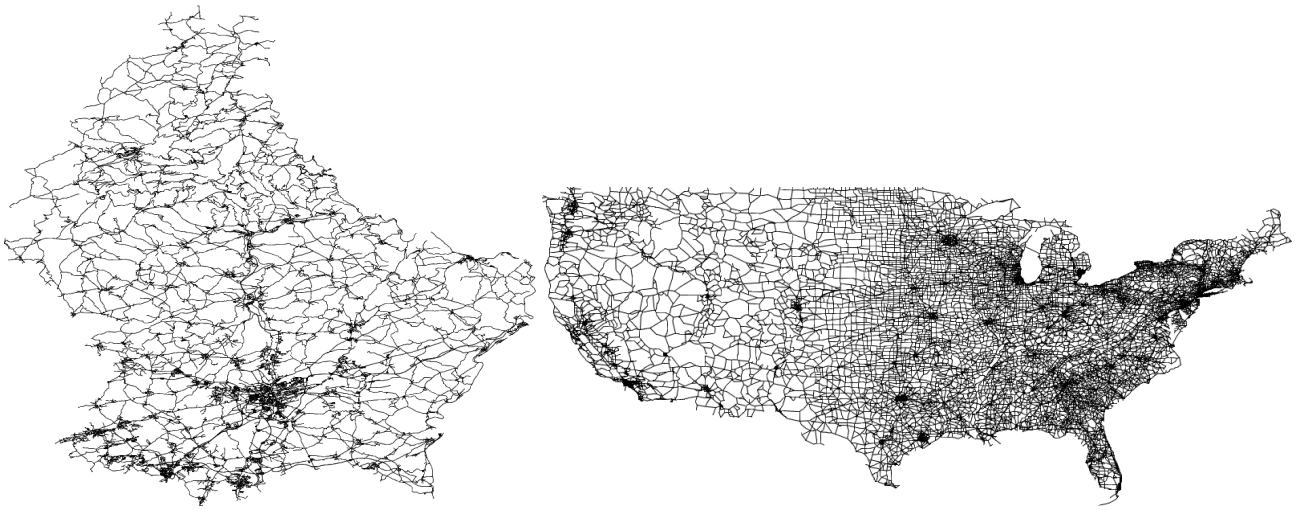


Figure 5: Left: The road network of Luxemburg with 114599 nodes and 119666 edges. Right: A segment of the US road network with 126146 nodes and 161950 edges.

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							

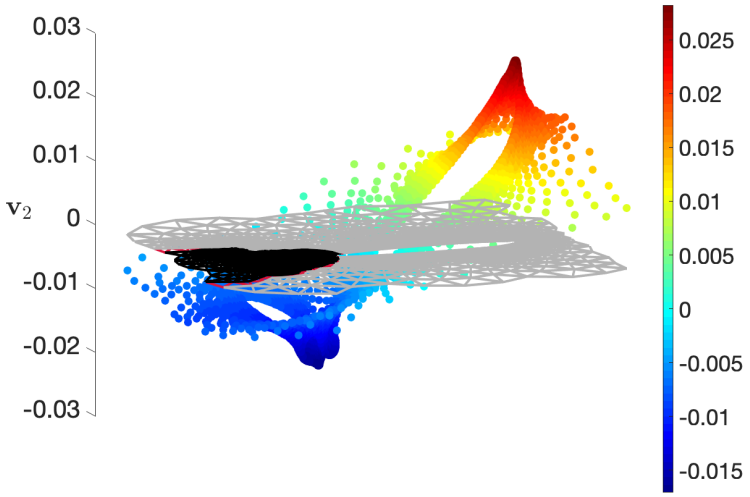


Figure 6: Partitioning the Airfoil graph based on the values of the Fiedler eigenvector. The two partitions are depicted in black and gray, while the cut edges in red respectively. The z-axis represents the value of the entries of the eigenvector.

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 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.

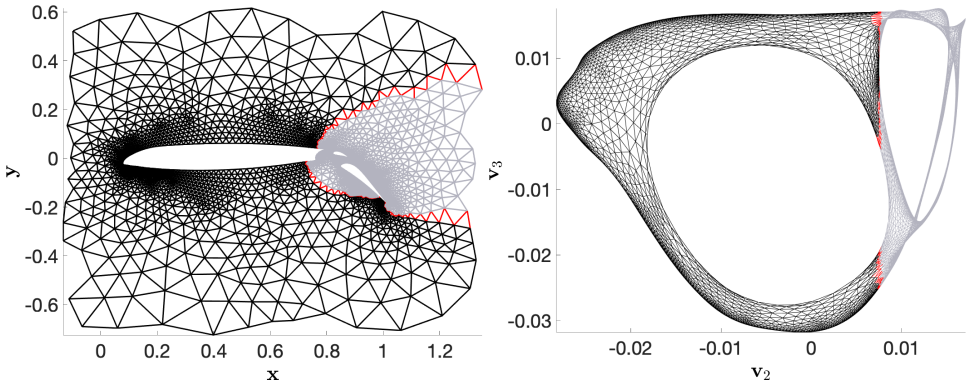


Figure 7: Visualizing the bipartitioning of the graph "airfoil1" with 4253 nodes and 12289 edges. Left: Spatial coordinates. Right: Spectral coordinates.

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.