ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

HPC Lab for CSE, Spring Semester 2023
Lecturer: Dr. R. Käppeli, Prof. O. Schenk
Assistants: Tim Holt, Malik Lechekhab, Dimosthenis Pasadakis, Xiaohe Niu, Niall Alexander Siegenheim, Julien Schmidt, Jitin Jami

Project 5 – Large Scale Graph Partitioning
Due date: 8 May 2023, 11:59pm (midnight)

Graph Partitioning – Load Balancing on Parallel Architectures

New discoveries in applications often require the solution of discretized partial differential equations, nonlinear optimizations, eigenvalue computations, and management of massive data sets, such as in accelerator design, PDE-constrained optimization, or groundwater flow modeling. In such simulations, continuous infinite-dimensional mathematical models are approximated with finite-dimensional models. To obtain the required accuracy and to resolve the underlying physics, the finite-dimensional models are often large, thus requiring fast parallel computers and/or advanced algorithmic solutions. A typical application might require a sequence of numerical optimization problems to be solved. Furthermore eigenvalues and eigenvectors have to be computed, and/or nonlinear and linear systems of equations have to be solved. Porting such an application to a parallel computer would require the computational work to be equally distributed on the processors. In an adaptive simulation framework, the work distribution can also change during the computation. The work and data need to be redistributed among the compute cores to enable it to be solved quickly. Several computational tasks need to be scheduled to maximize the utilization of the processors and to reduce the idle time processors spend waiting for data or for synchronization.

A typical model in computational science and engineering is expressed using the language of continuous mathematics, such as PDEs and linear algebra, but techniques from discrete or combinatorial mathematics also play an important role in solving these models efficiently. Several discrete combinatorial problems and data structures, such as graph and hypergraph partitioning, supernodes and elimination trees, vertex and edge reordering, vertex and edge coloring, and bipartite graph matching, arise in these contexts. As an example, parallel graph partitioning tools can be used to ease the task of distributing the computational workload across the processors.

Over the past decade, parallel computers have been used with great success in many scientific simulations. While differing in their numerical methods and details of implementation, most applications successfully parallelized to date are static applications. Their data structures and memory usage do not change during the course of the computation. Their interprocessor communication patterns are predictable and nonvarying, and their processor workloads are predictable and roughly constant throughout the simulation. However, increasing use of dynamic simulation techniques is creating new challenges for developers of parallel software. For example, adaptive finite element methods refine localized regions of the mesh and adjust the order of the approximation on individual elements to obtain a desired accuracy in the numerical solution. As a result, memory must be allocated dynamically to allow creation of new elements or degrees of freedom. Communication patterns can vary as refinement creates new element neighbors. In addition, localized refinement can cause severe processor load imbalance as elemental and processor work loads change throughout a simulation. Particle simulations and N-body simulations are other examples of dynamic applications. In particle simulations, scalable parallel performance depends on a good assignment of particles to processors; grouping physically close particles within a single processor reduces interprocessor communication. Data structures and communication patterns change as particles and surfaces move. Repartitioning of the particles or surfaces is needed to maintain geometric locality of objects within processors. This scientific question can be solved with parallel graph partitioning algorithms and parallel coloring tools, e.g., based on discrete graphs and hypergraphs techniques. See Figure 1 for applications of graph-theoretical tools in scientific computing.
High-performance graph partitioning libraries have been a very important part of scientific and engineering computing for years, and their importance continues to grow as microprocessor architectures become more complex and software libraries become better designed to integrate easily within applications. Despite the fact that there are various science

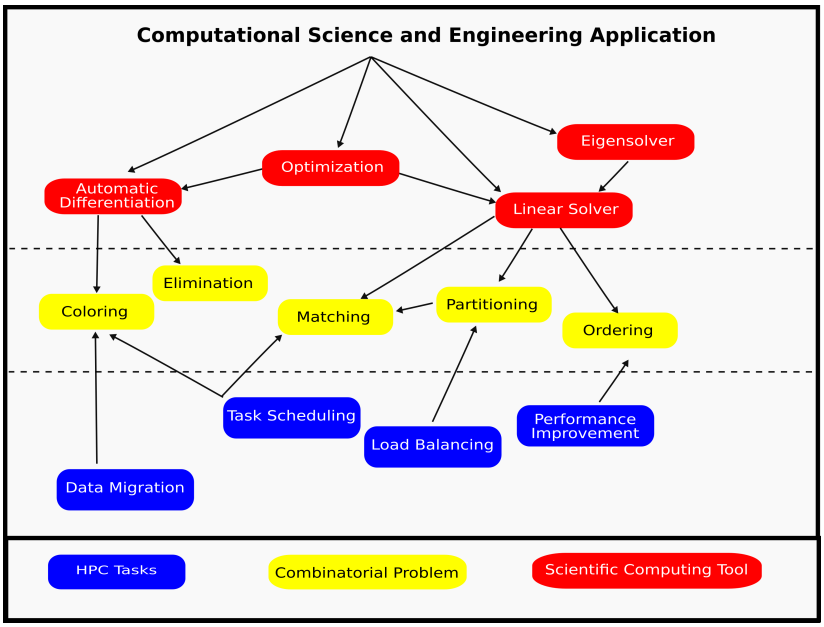


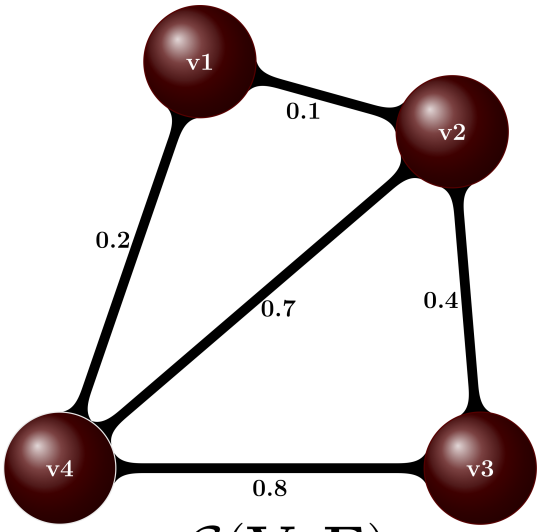
Figure 1: The solution of an application in computational science and engineering on a parallel computer requires scientific computing algorithms (the first row of the figure), and involves graph-theoretical research such as graph partitioning (the second row) and HPC tasks (the third row).

and engineering applications, the underlying algorithms typically have remarkable similarities, especially those algorithms that are most challenging to implement well in parallel. It is not too strong a statement to say that these graph partitioning libraries are essential to the broad success of scalable high-performance computing in computational science and engineering.

Graph Partitioning, Single-Core Optimization (100 points)

In computational science and high-performance computing, the graph partition problem is defined on data represented in the form of a graph $\mathcal{G} = (V, E)$ with V vertices and E edges, such that it is possible to partition G into smaller components with specific properties. For instance, a k-way partition divides the vertex set into k smaller components. In general, good solutions for the graph partitioning problem require that cuts are small and partitions have equal size. The problem arises, for instance, when assigning work to a parallel computer. In order to achieve efficiency, the workload (partition size) should be balanced evenly among the processors while communication between them (edge cut) should be kept to a minimum. Other important application domains of graph partitioning, and a detailed overview of advances in the field can be found at [2].
Recently, the graph partition problem has gained importance due to its application for clustering and detection of cliques in social, pathological, and biological networks. Since graph partitioning is an NP-hard problem, practical solutions are based on heuristics. There are two broad categories of methods, local and global. Well known local methods are the Kernighan–Lin algorithm, and Fiduccia–Mattheyses algorithms, which were the first effective 2-way cuts by local search strategies. Their major drawback is the arbitrary initial partitioning of the vertex set, which can affect the final solution quality. Global approaches rely on properties of the entire graph and not on an arbitrary initial partition. The most common example is spectral partitioning, where a partition is derived from the spectrum of the graph Laplacian matrix. In cases where the nodes of the graph are associated with a coordinate list, inertial bisection is also a frequent choice of partitioning method.

The **spectral method** was initially considered the standard to solve graph partitioning problems. It utilizes the spectral



$$\mathcal{G}(\mathbf{V}, \mathbf{E})$$

$$\mathbf{W} := \sum_{i \in A, j \in B} w_{ij} = \begin{bmatrix} 0 & 0.1 & 0 & 0.2 \\ 0.1 & 0 & 0.4 & 0.7 \\ 0 & 0.4 & 0 & 0.8 \\ 0.2 & 0.7 & 0.8 & 0 \end{bmatrix}, \quad \mathbf{D} := \sum_{j=1}^{n} w_{ij} = \begin{bmatrix} 0.3 & 0 & 0 & 0 \\ 0 & 1.2 & 0 & 0 \\ 0 & 0 & 1.2 & 0 \\ 0 & 0 & 0 & 1.7 \end{bmatrix}$$

$$\mathbf{L} := \mathbf{D} - \mathbf{W} = \begin{bmatrix} 0.3 & -0.1 & 0 & -0.2 \\ -0.1 & 1.2 & -0.4 & -0.7 \\ 0 & -0.4 & 1.2 & -0.8 \\ -0.2 & -0.7 & -0.8 & 1.7 \end{bmatrix}.$$

Figure 2: A weighted, undirected and connected graph $G(V, E)$, with 4 vertices and 5 edges, with its degree $\mathbf{D}$, adjacency $\mathbf{W}$, and graph Laplacian $\mathbf{L}$ matrices.

graph theorem of linear algebra, that enables the decomposition of a real symmetric matrix into eigenvalues, within an orthonormal basis of eigenvectors. For an undirected graph $\mathcal{G}(V, E)$, with vertex set $V = \{v_1, \cdots, v_n\}$ and two complementary subsets V_1, V_2, such that $V_1 \cup V_2 = V$ and $V_1 \cap V_2 = \emptyset$, we consider an indicator vector $\mathbf{x} \in \mathbb{R}^n$ such that

$$x_i = \begin{cases} 1, & \text{if } i \in V_1, \\ 0, & \text{if } i \in V_2. \end{cases}$$

Some of the n nodes of the graph are connected via m edges, each of which has a positive weight associated with it, thus

$$w_{ij} = \begin{cases} > 0, & \text{if } v_i \text{ connected to } v_j, \\ = 0 & \text{otherwise.} \end{cases}$$

A bisection is computed using the eigenvector associated with the second smallest eigenvalue of the graph Laplacian matrix $\mathbf{L} \in \mathbb{R}^{n \times n}$, which encodes the degree $d_i = \sum_{j=1}^{n} w_{ij}$, i.e.; the number of connected edges, of each vertex along its diagonal, and the negative weights $-w_{ij}$. For an undirected and connected graph $G(V, E)$, as illustrated in Figure 2, with n vertices and m edges, the graph Laplacian can be realized in terms of the adjacency matrix $\mathbf{W} \in \mathbb{R}^{n \times n}$ and the degree matrix $\mathbf{D} \in \mathbb{R}^{n \times n}$ as follows. The graph Laplacian is a symmetric positive semidefinite matrix, with its smallest eigenvalue being $\lambda^{(1)} = 0$, and the associated eigenvector $\mathbf{v}^{(1)} = c\mathbf{1}$ being the constant one. The eigenvector $\mathbf{v}^{(2)}$ 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 $\mathbf{v}^{(2)}$. Thresholding

the values of $\mathbf{v}^{(2)}$ around 0 results in two roughly balanced (equal sized) partitions, with minimum edgecut, while thresholding around the median value of $\mathbf{v}^{(2)}$ produces two strictly balanced partitions. The procedure to compute a bisection using spectral partitioning is summarized in Algorithm 1. For a much more detailed description of the method we refer to [5].

Algorithm 1 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.
5 Choose $V_1 := \{v_i \in V | u_i < thres\}, V_2 := \{\{v_i \in V | u_i \geq thres\}$.
6 **return** V_1, V_2 ▷ bisection of $\mathcal{G}$
7 **end function**

Inertial bisection is a very different approach, relying on the geometric layout of the graph, i.e. geometric coordinates attached to the vertices of the graph. The main idea of the method is to find a hyperlane running though the "center of gravity of the points." In 2 dimensions (2D) such a line L is chosen such that the sum of squares of the distance of the nodes to the line is minimized. It is defined by a point $\overline{P} = (\overline{x}, \overline{y})$ and a vector $\mathbf{u} = (u_1, u_2)$ with $\|\mathbf{u}\|_2 = \sqrt{u_1^2 + u_2^2}$ such that $L = \{\overline{P} + \alpha\mathbf{u} \,|\, \alpha \in \mathbb{R}\}$ (Figure 3). We choose

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

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

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

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 (2) 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 [5] and references therein for a thorough overview of the inertial method.
The last task of the inertial partitioning routine (line 4), i.e partitioning nodes associated with geometric coordinates around a direction normal to the partitioning plane, is already implemented in the toolbox provided to you in function `inertialPart` available in the file `Partitioning.jl`. The eigenvalue computations, both for spectral and inertial bisection should be performed with the function `eigs` from the `Arpack` package. Visit Arpack.jl for more information.

Partitioning metrics
In what follows we will use the number of cut edges between partitions to determine the quality of the partitioning result. The size of an edge separator, or edgecut, which partitions the graph into a vertex subset and its complement

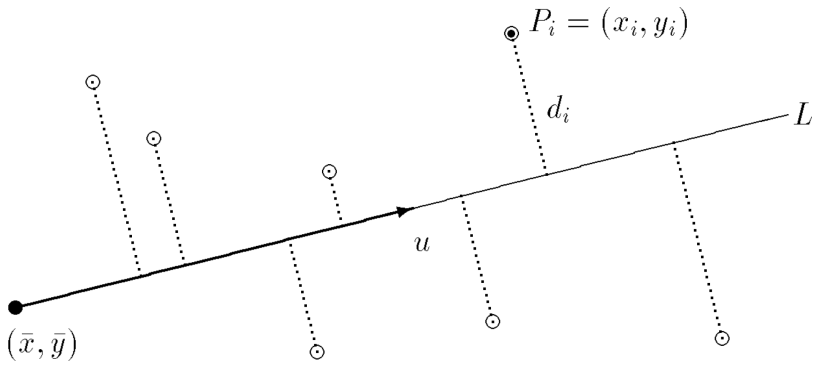


Figure 3: Illustration of inertial bisection in 2D [5].

Algorithm 2 Inertial bisection.

Require: $\mathcal{G}(V, E), P_i = (x_i, y_i)$, $i = 1, \ldots, n$ the coordinates of the nodes
Ensure: A bisection of $\mathcal{G}$

```
function InertialBisection(graph G(V, E), P_i)
    Calculate the center of mass of the points                    ▷ acc. to (1)
    Compute the eigenvector associated with the smallest eigenvalue of M   ▷ M acc. to (2)
    Partition the nodes of the graph around line L.
    return V_1, V_2                                               ▷ bisection of G
end function
```

$V_1 \cup V_2 = V$ is defined as follows:

$$\text{cut}(V_1, V_2) = \sum_{i \in V_1, j \in V_2} w_{ij}. \tag{3}$$

The calculation of cut edges is implemented in the function `countEdgeCut()` in the file `Partitioning.jl`. The cardinality of each subset (i.e the number of nodes in each partition) is given by counting the recurrence of the indicator vector $\mathbf{x}$ entries

Good scalability of parallel distributed memory solvers requires equally sized (balanced) cardinalities, so that the waiting time of the individual threads is minimized. For a k-way partition the load imbalance b_k is defined as the ratio of the highest partition cardinality over the average partition cardinality,

$$b^k = \max_{\left|V_i^k\right|} \left|V_i^k\right| / \left|\tilde{V}^k\right|, \tag{4}$$

where $\tilde{V}$ is the average partition weight. The load imbalance $b^k = 1 + b_r^k \geq 1$, where $b_r^k \geq 0$ characterizes the deviation from obtaining an evenly balanced partitioning. The optimal value for b_r^k is therefore zero, and it implies that the k partitions contain exactly the same number of nodes. In most applications it suffices to ensure that b_r^k is bounded from above by a desired threshold.

The purpose of this assignment is

1. construct graphs from lists describing node connectivity
2. to implement various graph partitioning algorithms in Julia and to test these methods on a variety of graphs;
3. to use a standard software package for graph partitioning (such as METIS) and to use modern computational science software engineering tools such as GitHub to interface external software libraries with Julia;

4. to evaluate the quality of the various graph partitioning algorithm, and visualize the partitioning results.

Software tools

We will use one external partioning software tool for this HPC miniproject. METIS[1] [7], probably the most widely used graph partitioning software, was developed by Karypis and Kumar and is probably the most widely used graph partitioning software. It consists of a set of serial programs for partitioning graphs, partitioning finite element meshes, and producing fill reducing orderings for sparse matrices. It is based on local techniques that iteratively improve starting solutions by swapping nodes between the partitions that lie in the neighborhood of the cut. Kernighan and Lin [8] were the first to propose a local search method in this fashion. Their partition refinement strategy, which swaps pairs of vertices that result in the largest decrease in the size of the edgecut, was later accelerated by Fiduccia and Mattheyses [6]. METIS is based on this method, and subsequent improvements of it. Finally, 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 Julia and METIS through the Julia wrapper `Metis.jl` which allow us to use the functions `METIS_PartGraphRecursive` and `METIS_PartGraphKway` with the same arguments and argument order described in the Metis manual.

The assignment

You can find the graph partitioning toolbox `Tools` on the Moodle webpage. This toolbox contains Julia code for the creation of several graph and graph partitioning methods, e.g., coordinate bisection, and has routines to generate recursive multiway partitions and visualize the partitioning results.

1. **Log in to Euler and set up the environment:**

   The following commands log in to Euler and set up the Julia environment:

```
[user@local]$ ssh -Y euler.ethz.ch
[user@eu-login]$ module load gcc/6.3.0 julia eth_proxy
[user@eu-login]$ export JULIA_DEPOT_PATH="$HOME/.julia:/cluster/project/sam/karoger-
    shared/julia:$JULIA_DEPOT_PATH"
[user@eu-login]$ srun --time=4:00:00 --mem-per-cpu=8G --x11 --pty bash
srun: job xxxxxxxx queued and waiting for resources
srun: job xxxxxxxx has been allocated resources
[user@eu-xx-xxx-x]$ julia
               _
   _       _ _(_)_     |  Documentation: https://docs.julialang.org
  (_)     | (_) (_)    |
   _ _   _| |_  __ _    |  Type "?" for help, "]?" for Pkg help.
  | | | | | | |/ _` |   |
  | | |_| | | | (_| |   |  Version 1.8.5 (2023-01-08)
 _/ |\__'_|_|_|\__'_|   |  Official https://julialang.org/ release
|__/                    |

julia> 1 + 1
julia> ans
2
```

   The first command enables X11 forwarding, the second loads the necessary modules, the third sets the Julia depot path (where we have already installed the necessary packages), the fourth submits an interactive job to the queue (1 CPU, 8G of memory and X11 forwarding enabled for 4 hours) and the final command

[1] https://github.com/KarypisLab/METIS

launches a julia interactive session (also called "REPL" for Read-Eval-Print-Loop). For the duration of the project, we recommend adding the second and third commands to your `.bashrc` configuration script (located in `$HOME/.bashrc`).

In order to use `Metis.jl` and the other packages, you can enter the package manager interactive session from the Julia REPL by pressing the key `]`. There, you can activate and instantiate the project environment:

```
julia> ]
(@v1.8) pkg> activate "/path/to/project/.../External"
```

Here `/path/to/project/.../External` should be replaced with the path where you unpacked the `project5-code.tgz` archive.

Alternatively, you can also install Julia and the packages on your own machine. First, you need to install Julia. You can either use pre-compiled binaries (recommended) or compile directly from the source. Please follow the instructions available at https://julialang.org/downloads/. Once Julia is installed, launch an interactive Julia REPL session and either activate and instantiate the project environment

```
julia> ]
(@v1.8) pkg> activate "/path/to/project/.../External"
(@v1.8) pkg> instantiate
```

or manually install the packages into your `MyOwnProject` with the Julia package manager

```
julia> ]
(@v1.8) pkg> activate "MyOwnProject"
(@v1.8) pkg> add delimitedFiles MAT Arpack LinearAlgebra Metis Random SparseArrays
    Statistics Graphs SGtSNEpi GLMakie Colors CairoMakie PrettyTables
```

Once the environment set up, you may run the `main.jl` script in the REPL

```
julia> include("main.jl")
```

This will print several tables (filled with dummy values) and create a PDF `airfoil1.pdf` of the graph in `airfoil1.mat`. You can display the PDF with

```
[user@eu-login]$ display airfoil1.pdf
```

For more information about thee Julia programming language, packages management and environment, please refer to the Julia documentation.

2. **Construct adjacency matrices from connectivity data [10 points]:**

Your 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 `Meshes/Countries/` 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 4.

Open the Julia file `Tools/read_csv_file.jl` and complete the missing sections of the code. Your goal is to:

- Read the .csv files in Julia.
- 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. [2]

[2] If there is an edge missing and the matrix is non-symmetric, apply the transformation $\mathbf{W}_{\text{sym}} = \frac{\mathbf{W}+\mathbf{W}^T}{2}$.

- Visualize the graphs of Norway and Vietnam using the function `draw_graph(A, coords)`.
- Save the sparse adjacency matrices and the corresponding coordinates in a new folder, e.g. `/Meshes/Countries/mat`.

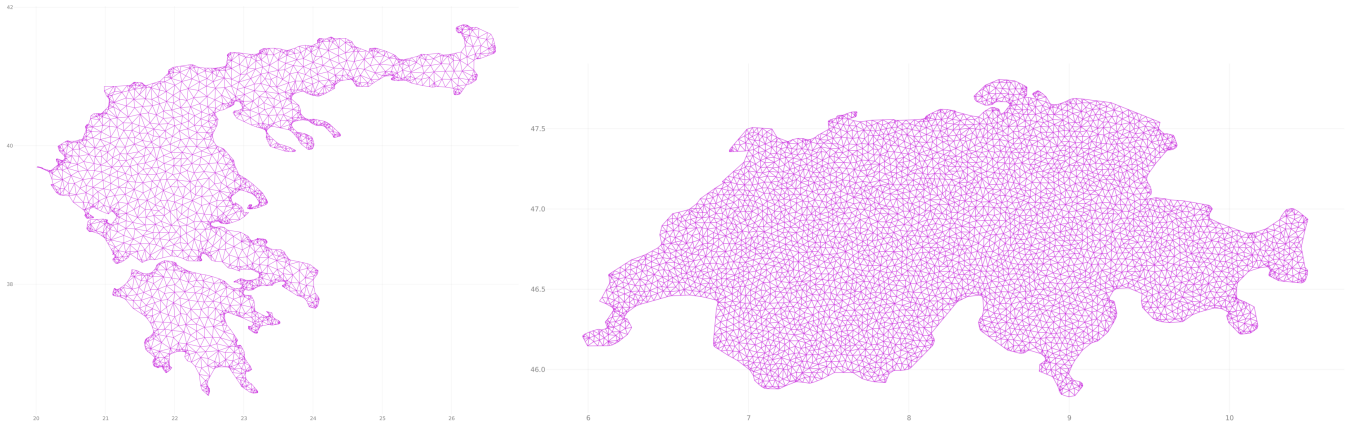


Figure 4: Graphs corresponding to the maps of various countries. Left: Greece with 3117 nodes and 3902 edges. Right: Switzerland with 4468 nodes and 15230 edges.

3. **Implement various graph partitioning algorithms [30 points]:**

   - Open the Julia file `bench_bisection.jl` and familiarize yourself with the code.
   - Implement **spectral graph bisection** based on the entries of the Fiedler eigenvector. Use the incomplete Julia file `part_spectral.jl` for your solution. Justify the threshold you chose.
   - 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 Julia file `part_inertial.jl` for your solution.
   - Report the bisection edgecut for all toy meshes that are loaded in the script `bench_bisection.jl`. Use Table 1 to report these results. Comment on your results.

Table 1: Bisection results

Mesh	Coordinate	Metis 5.0.2	Spectral	Inertial
mesh1e1	18			
mesh2e1	37			
netz4504_dual				
stufe				

4. **Recursively bisecting meshes [20 points]:**

   We will now partition 2D graphs emerging from structural engineering matrices of NASA, available in the SuiteSparse Matrix Collection (SSMC) [3]. A robust algorithm that allows us to create a 2^l partition, with l being an integer, is presented in Algorithm 3. 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 $\mathcal{G}_p$. In practize, only strongly balanced bisection routines are utilized within this algorithm [1].

Algorithm 3 Recursive bisection.

Require: $\mathcal{G}(V, E)$,
Ensure: A 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+k')
    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

This algorithm is implemented in the file `recursive_bisection.jl` of the toolbox. Utilize the script `bench_recusrive` to recursively bisect the finite element meshes loaded 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 2 and comment about these results. Finally, visualize the results for $p = 16$ for the case "crack". An example for spectral recursive partitioning is illustrated in Figure 5.

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

Case	Spectral	Metis 5.1.0	Coordinate	Inertial
mesh3e1				
airfoil1				
3elt				
barth4				
crack				

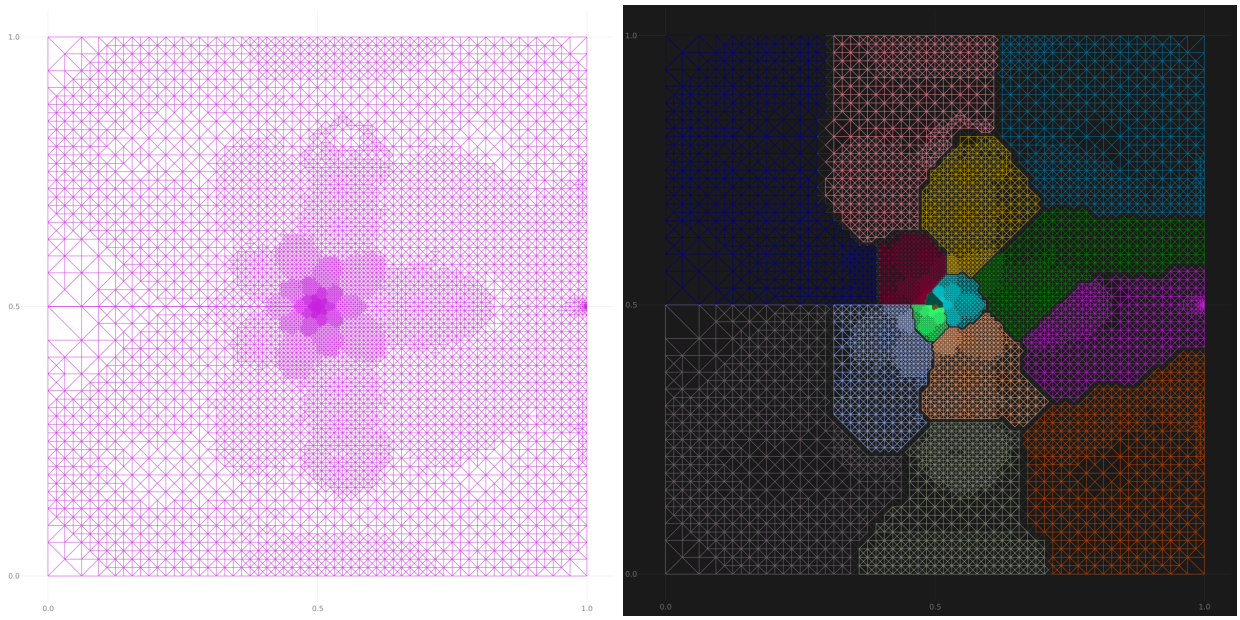


Figure 5: Left: The finite element mesh "crack" with 10240 nodes and 30380 edges. Right: 16-way recursive bisection of the mesh using Metis 5.1.0. Number of cut edges: 1186.

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 [10]. This necessitated the development of robust 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 down to a small number of vertices, a k-way partitioning of this much 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 [7].

   We will compare the quality of the cut resulting from the application of recursive bipartitioning and direct multiway partitioning, as implemented in Metis 5.1.0. Our test cases will be the graphs presented in Figure 6.

These graphs emerge from the road networks of Luxemburg and the US, with the graph edges and nodes representing road segments and intersections, respectively. 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.

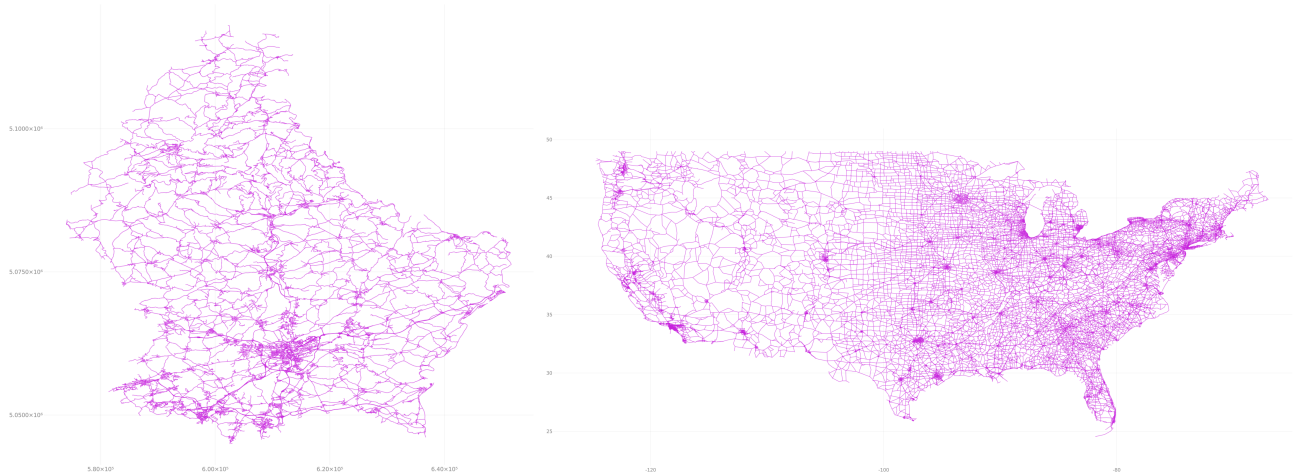


Figure 6: Left: The road network of Luxemburg with 114599 nodes and 119666 edges. Right: A segment of the US road network with 126146 nodes and 161950 edges.

Use the incomplete file `bench_metis.jl` for your implementation. Compare the cut obtained from Metis 5.1.0 after applying recursive bisection and direct multiway partitioning for the graphs in question. Consult the Metis manual 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 5.1.0.

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

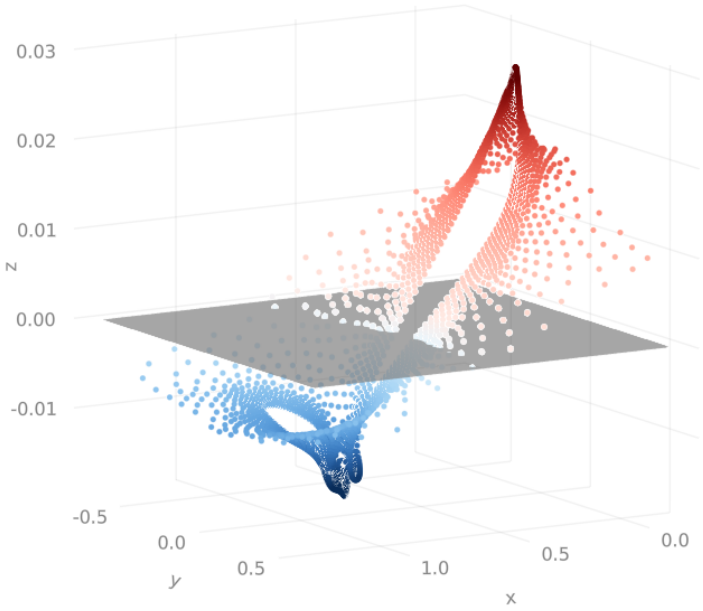


Figure 7: Partitioning the Airfoil graph based on the values of the Fiedler eigenvector. The two partitions are depicted in blue and red. The z-axis represents the value of the entries of the eigenvector.

6. **Utilizing graph eigenvectors [30 points]:**

   Provide the following illustrative results. Use the incomplete script `plot_fiedler.jl` for your implementation.

   a) 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?

   b) 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 7, for the graph "airfoil1". Comment on this plot.
   Hint: You might have to use the function `scatter` to get the desired result.

   c) 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, which is the ultimate usefulness of the resulting layout [9]. A successful layout should transmit 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 8

illustrates these 2 ways of visualizing the partitions of the ”airfoil1” graph. Describe the visual impact of the threshold you chose for spectral bisection.

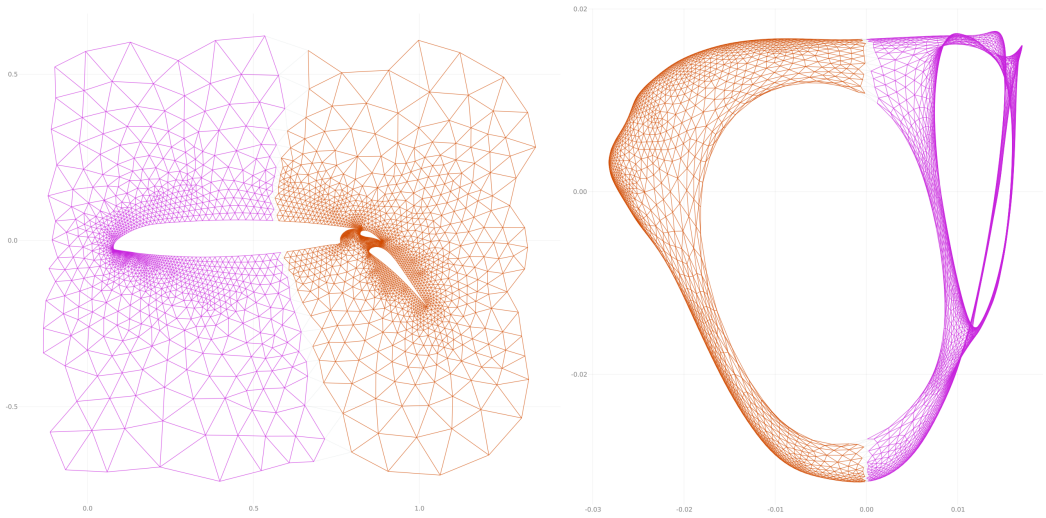


Figure 8: Visualizing the spectral bipartitioning of the graph ”airfoil1” with 4253 nodes, 12289 edges and a threshold set to zero. Left: Spatial coordinates. Right: Spectral coordinates.

Additional notes and submission details

We only accept submissions using our Latex template and Julia code. Submit **all the source code files** (together with your environment files `Manifest.toml` and `Project.toml`) in an archive file (tar, zip, etc.) and summarize your results and the observations for all exercises by writing a detailed Latex report. Use the Latex template from the webpage and upload the Latex summary as a PDF to Moodle .

- Your submission should be a zip or tar archive, formatted like project_number_lastname_firstname.zip / .tgz. It must contain:
  - all the source codes of your solutions.
  - Makefiles. If you have modified them, make sure they still build the sources correctly. We will use them to grade your submission.
  - project_number_lastname_firstname.pdf, your write-up (report) with your name.
- Submit your archive file through Moodle .

References

[1] C.E. Bichot and P. Siarry. *Graph Partitioning*. ISTE. Wiley, 2013.

[2] Aydin Buluc, Henning Meyerhenke, Ilya Safro, Peter Sanders, and Christian Schulz. *Recent Advances in Graph Partitioning*, pages 117–158. Springer International Publishing, Cham, 2016.

[3] Timothy A. Davis and Yifan Hu. The university of florida sparse matrix collection. *ACM Trans. Math. Softw.*, 38(1):1:1–1:25, December 2011.

[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.

[5] U. Elsner. Graph Partitioning: A Survey. Technical report, Technische Universität Chemnitz, Germany, 97-27, 1997.

[6] C. M. Fiduccia and R. M. Mattheyses. A linear-time heuristic for improving network partitions. In *Proceedings of the 19th Design Automation Conference*, DAC '82, pages 175–181, Piscataway, NJ, USA, June 1982. IEEE Press.

[7] George Karypis and Vipin Kumar. Parallel multilevel k-way partitioning scheme for irregular graphs. In *Proceedings of the 1996 ACM/IEEE Conference on Supercomputing*, Supercomputing 96, page 35es, USA, 1996. IEEE Computer Society.

[8] B. W. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *The Bell System Technical Journal*, 49(2):291–307, Feb 1970.

[9] Y. Koren. Drawing graphs by eigenvectors: Theory and practice. *Comput. Math. Appl.*, 49(1112):18671888, June 2005.

[10] Horst D. Simon and Shang-Hua Teng. How good is recursive bisection? *SIAM J. Sci. Comput.*, 18(5):14361445, September 1997.