ETH zürich

High-Performance Computing Lab for CSE **2023**

Student: Simon Aaron Menzi Discussed with: Lukas Bühler

Solution for Project 5

Due date: 8 May 2023 (midnight)

HPC Lab for CSE 2022 — Submission Instructions
(Please, notice that following instructions are mandatory: submissions that don't comply with, won't be considered)

- Assignments must be submitted to Moodle (i.e. in electronic format).
- 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.
- Provide both executable package and sources (e.g. C/C++ files, Matlab, Julia). If you are using libraries, please add them in the file. Sources must be organized in directories called:
  Project_number_lastname_firstname
  and the file must be called:
  project_number_lastname_firstname.zip
  project_number_lastname_firstname.pdf
- The TAs will grade your project by reviewing your project write-up, and looking at the implementation you attempted, and benchmarking your code's performance.
- You are allowed to discuss all questions with anyone you like; however: (i) your submission must list anyone you discussed problems with and (ii) you must write up your submission independently.

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

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. [1]
- 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`.

This Task was a warm up to the Julia language. The language seems to be a mix between python and C++ trying to get the best of both worlds. On a first glance, it seems to achieve that goal. Its fast, and has not a lot of boiler plate. What should be improved are the error messages, as they are not that useful for debugging. Following is the code that was used to solve this task

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

```julia
function read_csv_graph(path_file)
    # Steps
    #   1.  Load the .csv files
    #       see readdlm(...)
    data = readdlm(path_file *"-adj.csv", ',', UInt, skipstart=1)
    coords = readdlm(path_file*"-pts.csv", ',', Float64, skipstart=1)

    #   2.  Construct the adjacency matrix A
    A = create_adjacency_matrix(data)

    #   3.  Visualize and save the result
    #       use drawGraph(A, coords)

    #   4.  Return the matrix A and the coordinates
    #       return(A, coords)
    return(A, coords)
end
```

Following are the plots of Norway and Vietnam, generated in a for loop and this function:

Norway

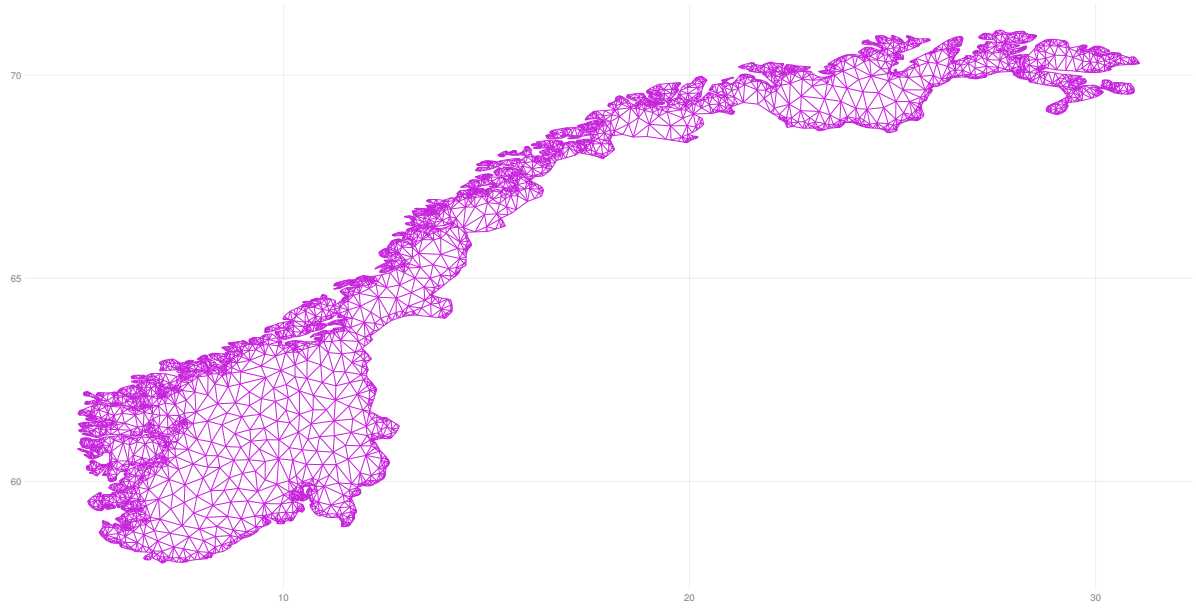

Figure 1

Vietnam

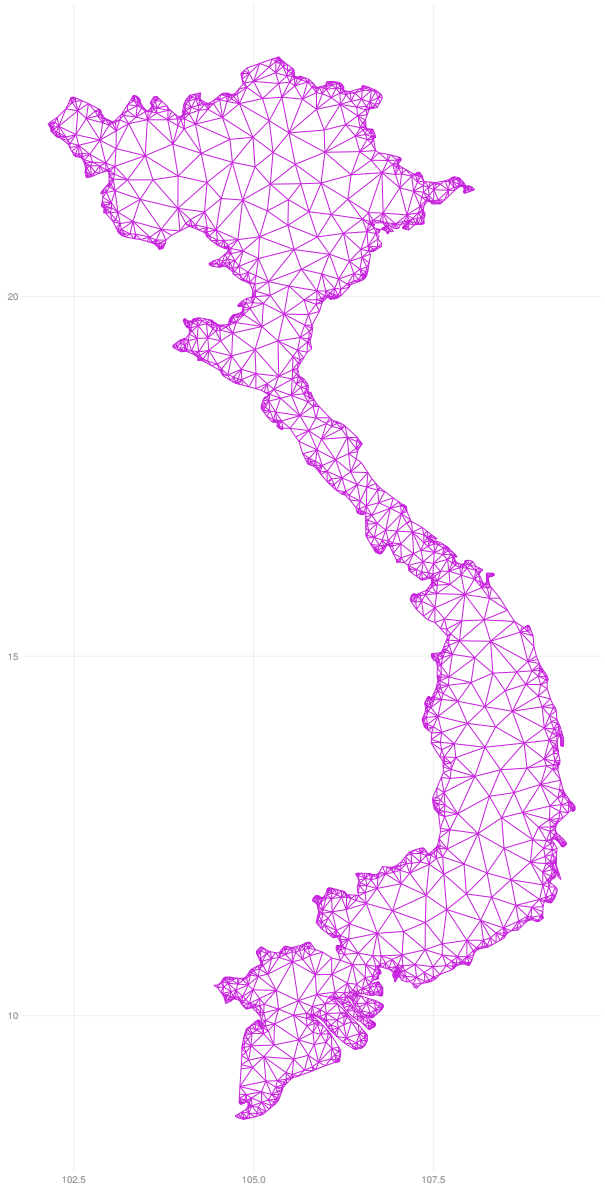

Figure 2

2. Task: 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.

This code was used to implement the spectral bisection method:

```
function degree_matrix(A)
    # n = size(A, 1)
    # D = zeros(Float64, n, n)
    # for i in 1:n
    #     D[i, i] = sum(A[i, :])
    # end
    D = diagm((vec(sum(A, dims=2))))
    return D
end


function spectral_part(A)
    n = size(A)[1]

    if n > 4*10^4
        @warn "graph is large. Computing eigen values may take too long."
    end

    # #   1.  Construct the Laplacian matrix.
    D = degree_matrix(A)
    L = D - A

    # # #   2.  Compute its eigendecomposition.
    val, vec = eigs(F, nev=2, which=:SM, ritzvec=true)

    # # #   3.  Label the vertices with the entries of the Fiedler vector.
    fiedler_vec = vec[:, 2]
    fiedler_val = val[2]

    #   4.  Partition them around their median value, or 0.
    #   The threshold of the partition is 0 as I want to have a roughly balanced partit
    p = ones(Int, n)
    for (i, val) in enumerate(fiedler_vec)
        if val > 0
            p[i] = 2
        end
    end

    p = Int.(p)
    #   5.  Return the indicator vector
    return p
end
```

As the degree-matrix will be used in other functions, it was taken out of the spectral-bisection-function.

The inertial bisection method was implemented with the following code:

```
function inertial_part(A, coords)
    @assert size(A)[1] == size(coords)[1] "Dimension mismatch: number of rows in A must


    #   1.  Compute the center of mass.
    n = size(coords)[1]
    x_bar = sum(coords[:,1]) / n
    y_bar = sum(coords[:,2]) / n
    x = coords[:,1]
    y = coords[:,2]
    center = [x_bar, y_bar]

    #   2.  Construct the matrix M. (see pdf of the assignment)
    S_xx = sum((x .- x_bar).^2)
    S_yy = sum((y .- y_bar).^2)
    # S_xy = dot(x, y) + x_bar * y_bar
    #      - sum(x .* y_bar)
    #      - sum(y .* x_bar)
    S_xy = sum((x .- x_bar) .* (y .- y_bar))

    M = [[S_xx S_xy]
         [S_xy S_yy]]



    #   3.  Compute the eigenvector associated with the smallest eigenvalue of M.
    # eigv = eigvecs(M)[:, 1]
    val, vec = eigs(M, nev=1, which=:SM, ritzvec=true)
    i = sortperm(val)
    eigv = vec[:,i]
    eigv = eigv / norm(eigv)
    eigv = [eigv[2], -eigv[1]]


    #   4.  Partition the nodes around line L
    #       (you may use the function partition(coords, eigv))
    p = ones(Int, n)
    V1, V2 = partition(coords, eigv)
    p[V2] .= 2

    #   5.  Return the indicator vector
    return p

end
```

Here, the provided recipe was a perfect step by step guide, except of the part where one needs to flip the eigenvector values. This was a major stepping stone.

Table 1: Bisection results

Mesh	Coordinate	Metis 5.0.2	Spectral	Inertial
mesh1e1	18	17	17	19
mesh2e1	37	34	35	47
netz4504_dual	25	20	24	30
stufe	16	17	16	16

3. Task: Recursively bisecting meshes [20 points]

The recursive bisection 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".

The bisections with the coordinate method just run through the x and y planes. The Inertial method seems to partition the vertices more equally but still not optimal.

Metis tries to get equal partitions by going from the center, where there are the most vertices to edges. The Spectral Method is visually going in circles. Over all, the metis method seems to be the most efficient in terms of cuts, the spectral method being a close second. Both coordinate and inertial methods are way worse, which is expected.

Crack with 16 partitions

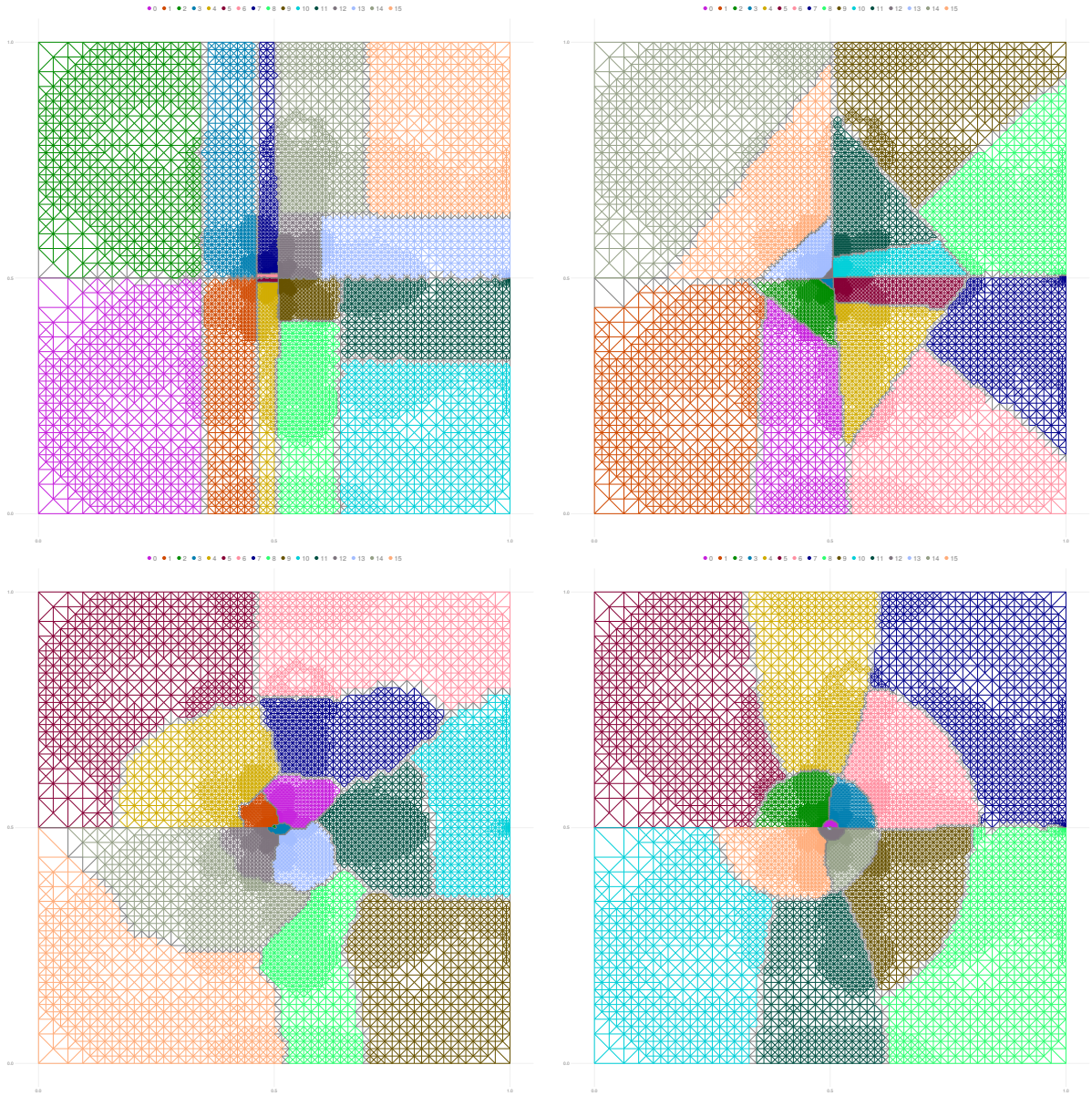

Figure 3: Top left: Coordinate, Top right: Inertial, Bottom left: Metis, Bottom right: spectral

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

Case	Spectral 8	Metis 5.0.2	Coordinate	Inertial
mesh3e1 - 8	74	74	75	83
mesh3e1 - 16	114	112	118	128
airfoil1 - 8	327	324	516	578
airfoil1 - 16	578	566	819	902
3elt - 8	372	402	733	880
3elt - 16	671	645	1168	1342
barth4 - 8	505	409	875	888
barth4 - 16	758	716	1306	1347
crack - 8	804	736	1344	1061
crack - 16	1303	1238	1861	1618

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

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.

The results show, that the k-way partition Metis method outperforms the recursive method. This is expected, as the direct approach is minimizing the cuts globally, whereas the recursive method is optimizing locally and increases its range with each iteration. In smaller graphs, this performance difference, performance in this context meaning less cuts, is not to big, but the greater the graph, the bigger the gap between the methods get. The plots roughly visualize this behavior and difference in performance.

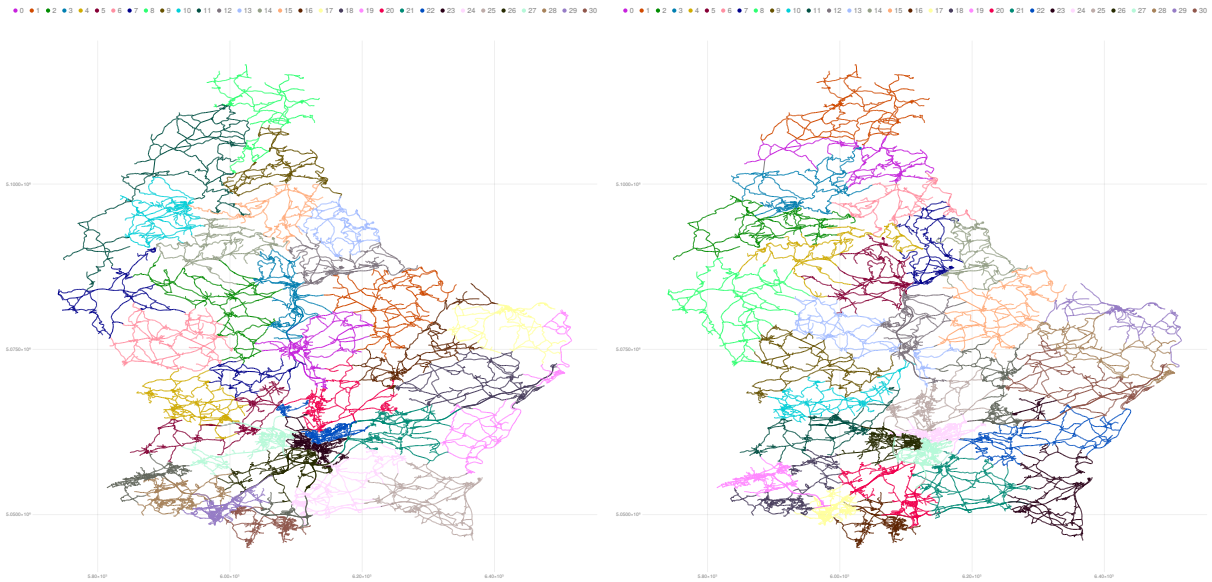

Figure 4: Partitions for 'luxembourg' mesh: left: METIS k-way, right: METIS recursive (virgin islands omitted for visibility)

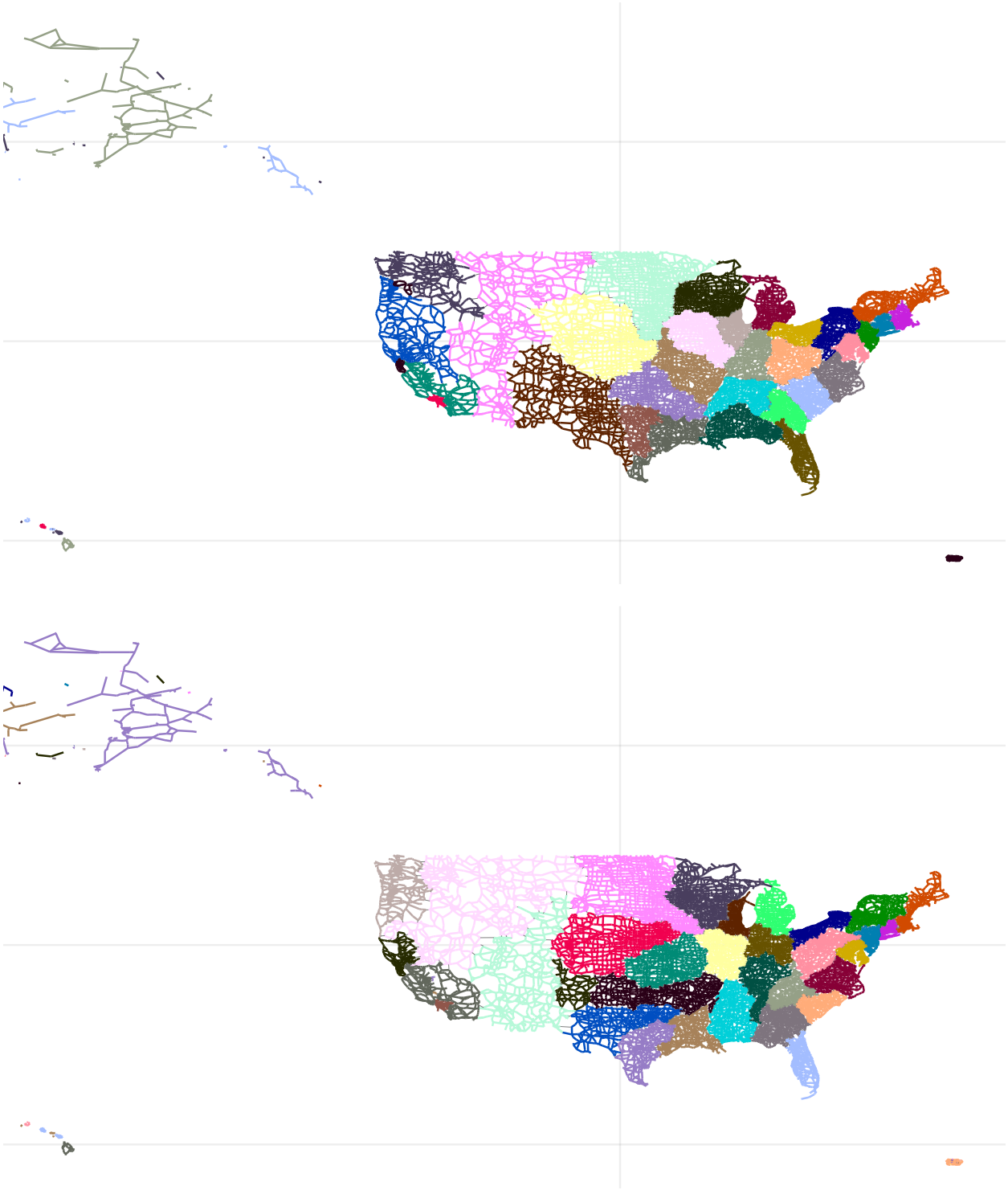

Figure 5: Partitions for 'usroads' mesh: Top: METIS k-way, bottom: METIS recursive (virgin islands omitted for visibility)

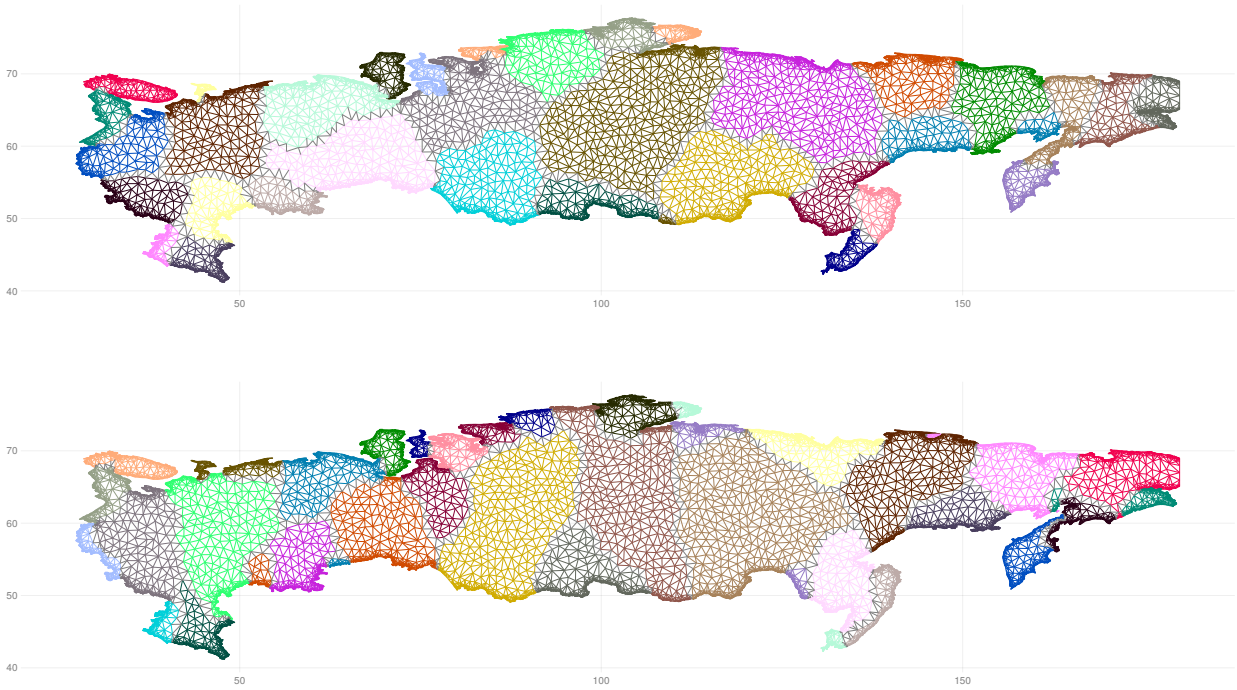

Figure 6: Partitions for 'RU-40527' mesh: Top: METIS k-way, bottom: METIS recursive

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

Partitions-Method	Luxemburg	usroads-48	Greece	Switzerland	Vietnam	Norway	Russia
16-KWAY	172	529	282	693	244	255	514
16-RECURSIVE	175	628	298	699	400	290	577
32-KWAY	308	926	468	1052	248	487	909
32-RECURSIVE	319	972	525	1089	453	486	1019

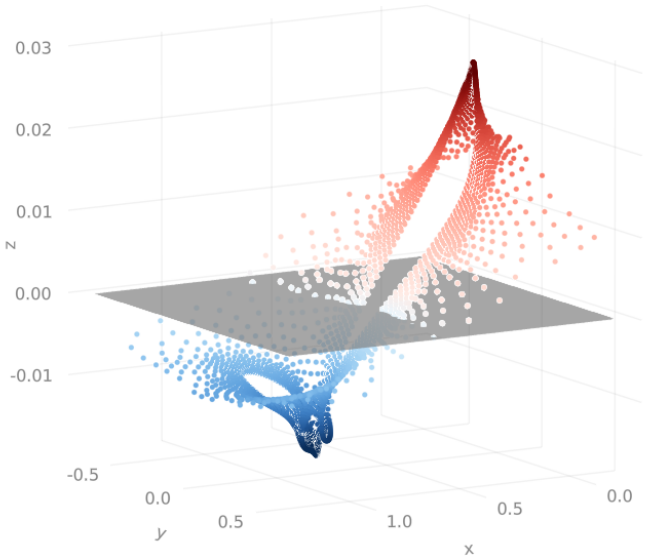

Figure 7: Partitioning the Airfoil graph based on the values of the Fiedler eigenvector. The two partitions are depicted in green and purple. The z-axis represents the value of the entries of the eigenvector.

5. Task: Utilizing graph eigenvectors [30 points]

Provide the following illustrative results. Use the incomplete script `plot_fiedler.jl` 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?

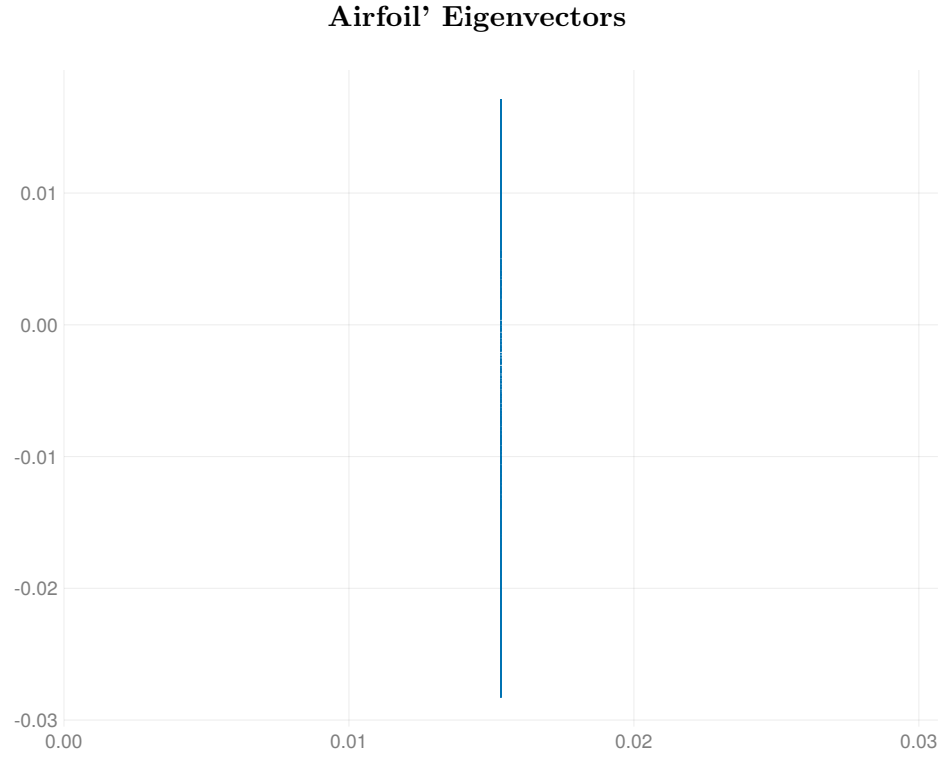

Figure 8: Airfoil' two smallest Eigenvectors plotted. As expected, the first Eigenvector is constant.

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 7, for the graph "airfoil1". Comment on this plot.

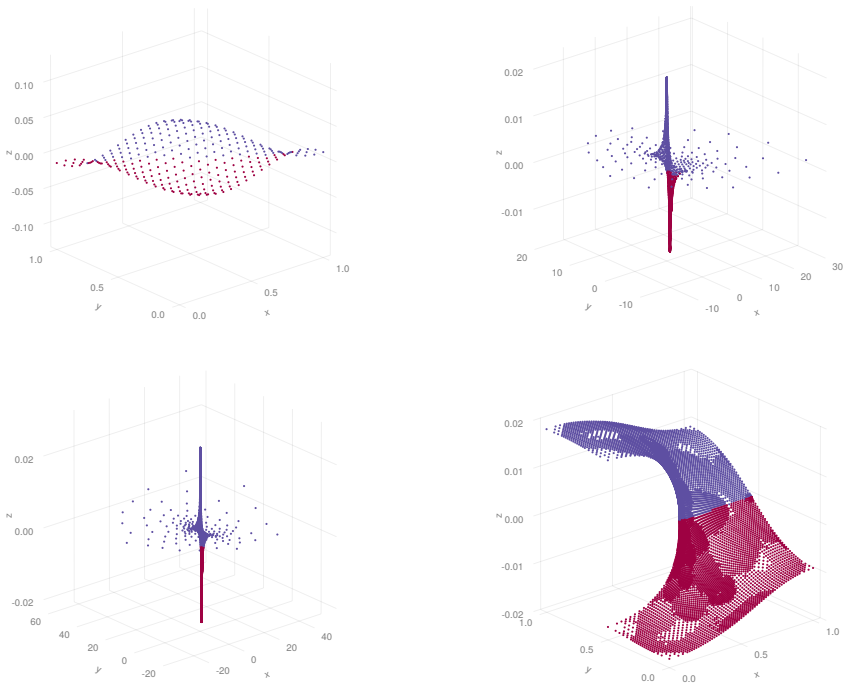

Figure 9: Top left: Mesh3e1, Top right: Barth4, Bottom left: 3elt, Bottom right: Crack

The results show, that the cut is done such that there roughly equal "strong" vertices in

terms of Eigenvector on both sides, as well as the roughly the same amount of vertices. The maximal iteration count for the Eigendecomposition has been set to 10000 due to limitations of Euler and time constraints.

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, which is the ultimate usefulness of the resulting layout [1]. 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 14 illustrates these 2 ways of visualizing the partitions of the "airfoil1" graph. Describe the visual impact of the threshold you chose for spectral bisection.

A threshold of 0 has been chosen for the spectral method, which has an astounding visual impact, as sometimes, the cut can not even be seen. Still, the method seems to split the vertices equally and minimizes cuts. In Eigenvector space, one sees that for symmetrical graphs, the resulting Eigenvectors also show that symmetry. Furthermore, wholes show itself in the Eigenvector space if the cut was made in the graph and not through the graph. Inaccuracies can be explained as the maximal iterations for the Eigendecomposition has been set to 10000 due to constraints on Euler and time.

Mesh3e1 Spectral Bisection

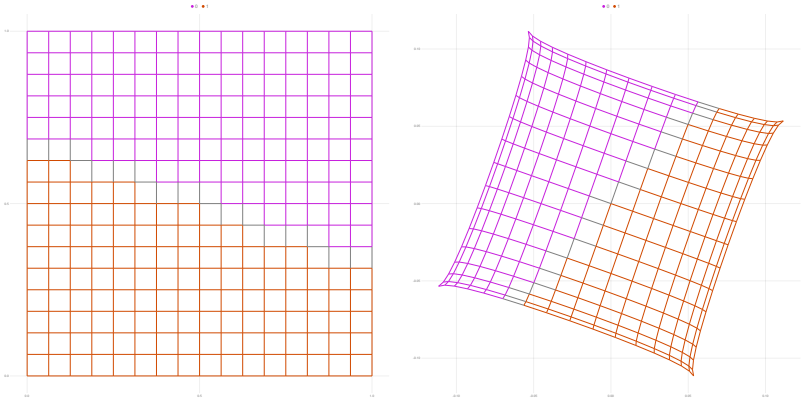

Figure 10: Left: Euclidian Space, Right: Eigenvector Space

Barth4 Spectral Bisection

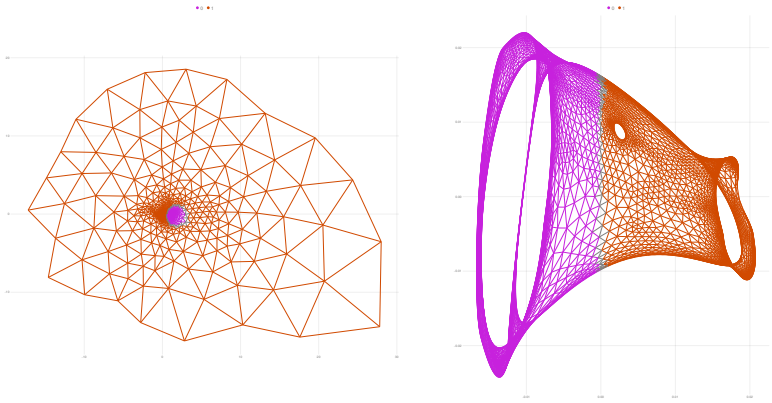

Figure 11: Left: Euclidian Space, Right: Eigenvector Space

3elt Spectral Bisection

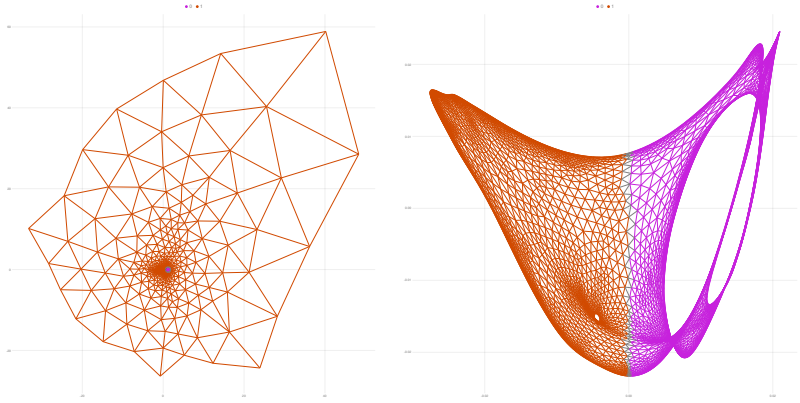

Figure 12: Left: Euclidian Space, Right: Eigenvector Space

Crack Spectral Bisection

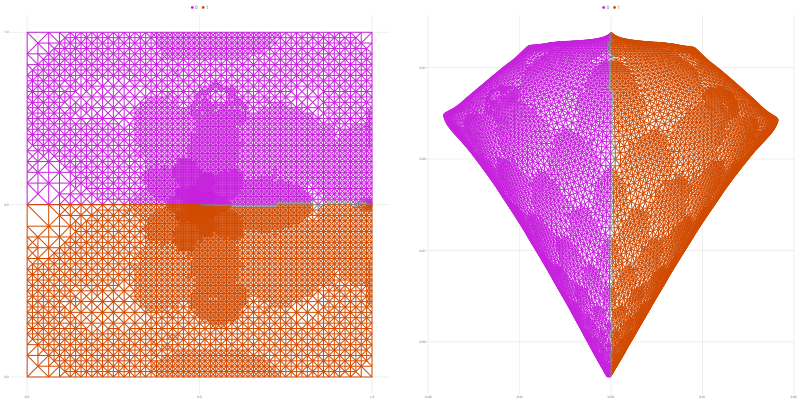

Figure 13: Left: Euclidian Space, Right: Eigenvector Space

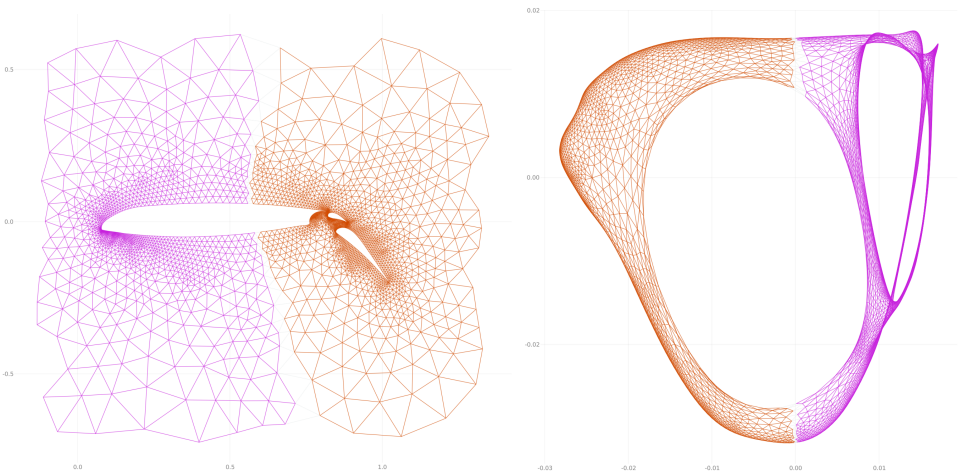

Figure 14: Visualizing the bipartitioning of the graph "airfoil1" with 4253 nodes, 12289 edges and a threshould set to zero. Left: Spatial coordinates. Right: Spectral coordinates.

References

[1] Y. Koren. Drawing graphs by eigenvectors: Theory and practice. *Comput. Math. Appl.*, 49(11–12):1867–1888, June 2005.

As last comment: As this project was to introduce us with graph partioning algorithms, I think it did a very well job. Also, it showed the capabilities of the Julia programming language and its strong points, being fast, high performing code with little boiler plate code.