Università della Svizzera italiana

Institute of Computing CI

High-Performance Computing **2022**

Student: Fabio Marchesi

Discussed with: Gioele De Pianto, Raffaele Morganti

Solution for Project 6

Due date: 07.12.2022, 23:59

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

For this task, we have a list of graphs, for each one of them we are given two comma-separated-value files, the first one contains the list of edges expressed as the pairs of nodes that they connect, the second one containing the positions of the nodes in the coordinate plane. For each graph, we need to build the adjacency matrix, visualize it and store it. We use **csvread()** to read the csv files, we pass **1** as the second parameter to specify that we want to exclude the header of the file. After that, we use accumarray. This function allows us to create a matrix *adj* where $adj_{i,j}$ is equal to one if there is an edge between node i and node j, otherwise $adj_{i,j}$ is equal to zero. Using **true** as the sixth parameter, we are specifying that we want the resulting matrix to be sparse. Being the graph undirected we want this matrix to be symmetric, in order to do that we can just sum to *adj* its transpose matrix. In the end, we visualize the graph using *gplotg* and we store *adj* and *positions* in a file **.mat**.

```
for c = 1:nc
    edges = csvread(adjs{c},1);
    positions = csvread(pts{c}, 1);
    % 2. Construct the adjaceny matrix (NxN). There are multiple ways to do so
    adj = accumarray(edges, 1, [], [], [], true);
    adj = adj + transpose(adj);
    % 3. Visualize the resulting graphs
    gplotg(adj, positions)
    pause;
    % 4. Save the resulting graphs
    save(flnames{c}, "adj", "positions")
end
```

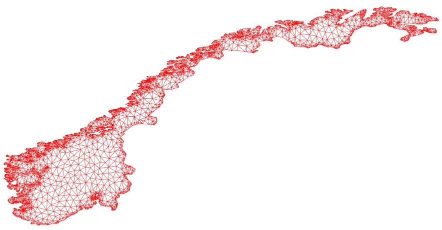

Figure 1: Visualization of Norway's map

Figure 2: Visualization of Vietnam's map

2. Task: Implement various graph partitioning algorithms [25 points]

We implemented the two requested algorithms:

- **Spectral Graph Bisection**: Given the adjacency matrix adj we compute its Laplacian matrix $L(adj)$, find the eigenvector associated to the second-smallest eigenvalue of $L(adj)$ (all its eigenvalues are non-negative and the smallest is equal to zero), and use the median of its entries to decide in which partition each point should go. In particular, if the entry number i of the selected eigenvector is less than the median of its entries then we insert point i into the first partition, otherwise we insert it in the second one.

- **Inertial Graph Bisection**: This algorithm is different from the first one because it also considers the positions of the points in the plane. Given the adjacency matrix adj and the coordinates of the points *positions* we compute the center of mass of the points $(\bar{x}, \bar{y})$. We compute the matrix $\begin{bmatrix} \mathrm{S}_{yy} & \mathrm{S}_{xy} \\ \mathrm{S}_{xy} & \mathrm{S}_{xx} \end{bmatrix}$ where $S_{yy} = \Sigma_1^N (y_i - \bar{y})^2$, $S_{xx} = \Sigma_1^N (x_i - \bar{x})^2$ and $S_{xy} = \Sigma_1^N (x_i - \bar{x})(y_i - \bar{y})$. Then compute the partition around the line L obtained as $\{(\bar{x}, \bar{y}) + \alpha u | \alpha \in R\}$, where u is the eigenvector associated with the smallest eigenvalue of M.

Table 1: Bisection results

Mesh	Coordinate	Metis 5.0.2	Spectral	Inertial
mesh1e1	18	19	18	20
mesh2e1	37	34	35	47
mesh3e1	19	19	22	19
mesh3em5	19	19	23	19
airfoil1	94	97	132	93
netz4504_dual	25	19	23	27
stufe	16	18	16	16
3elt	172	118	117	257
barth4	206	97	127	208
ukerbe1	32	32	36	28
crack	353	205	233	384

Generally speaking, the **Metis** implementation is the one with the best results, but also the **Spectral Bisection Method** works really well, outperforming by a minimal margin the **Metis** implementation in few instances. The **Coordinate Method** works better than the **Inertial Bisection Method**, but both of them perform worse than the other two methods, this is because they are limited by the fact that they use a straight line to partition the points.

3. Task: Recursively bisecting meshes [15 points]

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

Case	Spectral		Metis 5.0.2		Coordinate		Inertial	
Partitions	8	16	8	16	8	16	8	16
airfoil	398	633	336	584	516	819	670	1081
netz4504_dual	111	184	92	154	127	198	165	271
stufe	128	238	108	196	123	227	320	606
3elt	469	752	395	675	733	1168	814	1230
barth4	550	841	442	719	875	1306	977	1492
ukerbe1	467	695	132	233	225	374	339	499
crack	883	1419	819	1225	1343	1860	1351	1884

The idea behind recursive bisection is really simple. We have already seen different methods to split the nodes of a graph in two (ideally balanced) partitions. To divide a graph in more the two partitions (we will work with 8 and 16 partitions) we can apply the algorithms presented recursively. Indeed, we can start by splitting the original domain D into subdomains $D1$ and $D2$ and apply the same logic to $D1$ and $D2$ until we reached the desired number of partitions. Obviously, using more partitions will lead to an increment of the communication cost (indeed, the edge cut with 16 partitions is always greater than the one with 8 partitions) but the time required to execute the computation of each partition will be smaller because there are fewer nodes in each partition. In general, we can see how the results obtained with the Metis implementation are the best. The limitation imposed by the fact that the coordinate and the inertial method needs to choose a straight line to partition the current domain leads to bad results, indeed, in most cases the spectral outperforms them (like it did also for the 2-dimensional partitionings reported before).

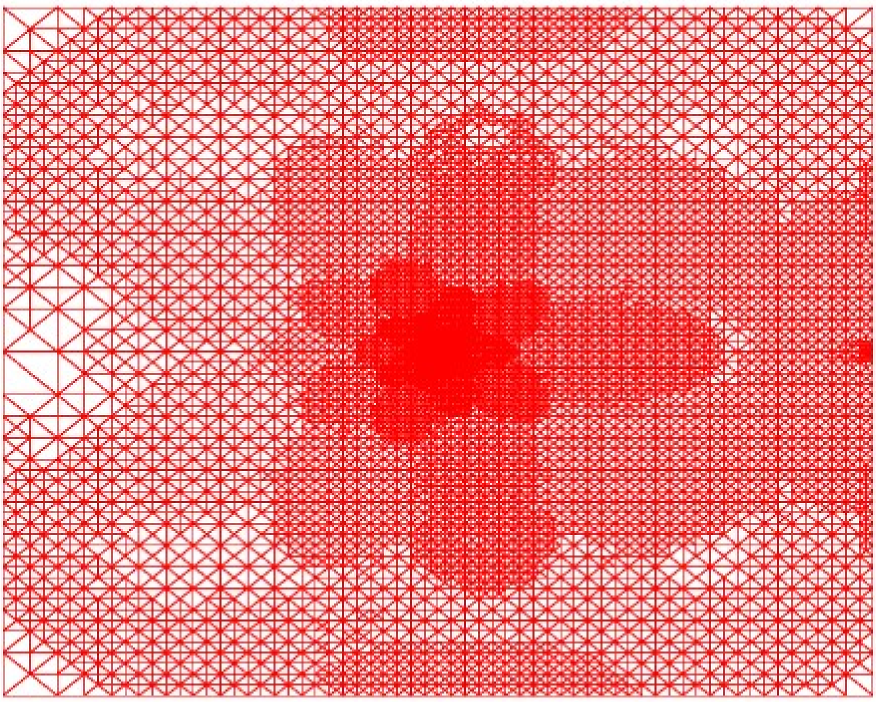

Figure 3: The finite element mesh "crack" with 10240 nodes and 30380 edges

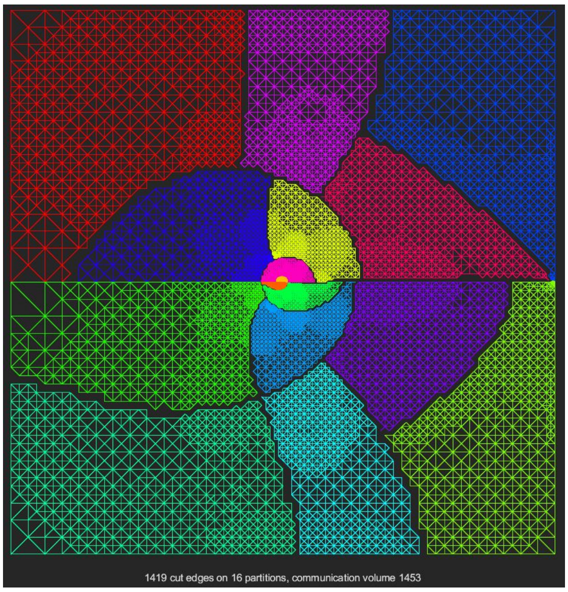

Figure 4: Recursive spectral bi-partitioning with 16 partitions

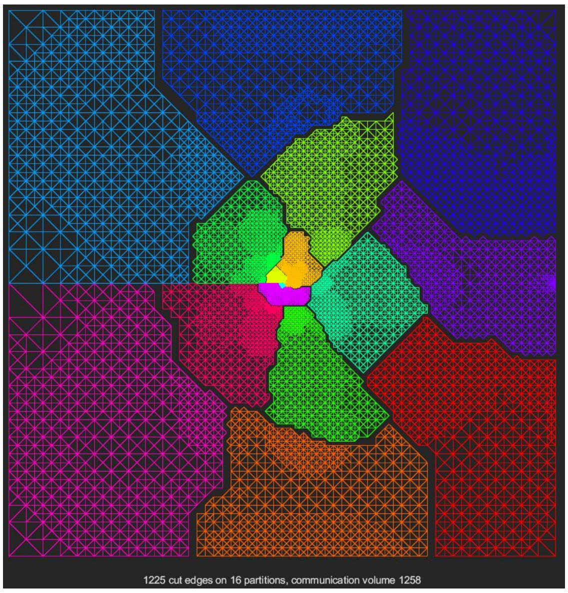

Figure 5: Recursive Metis bi-partitioning with 16 partitions

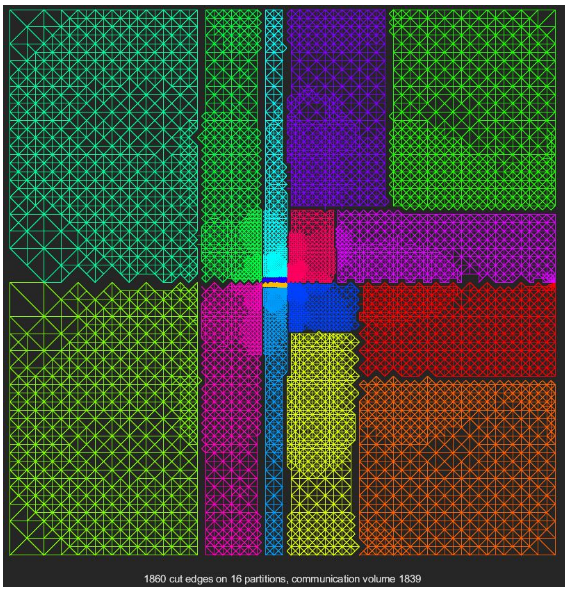

Figure 6: Coordinate bi-partitioning with 16 partitions

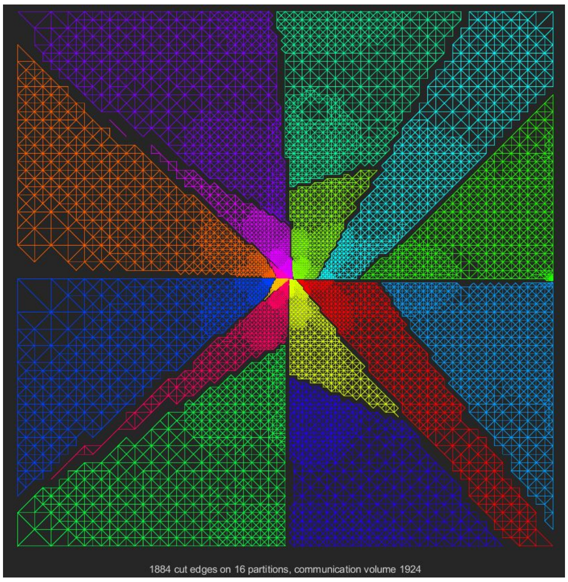

Figure 7: Recursive intertial bi-partitioning with 16 partitions

4. Task: 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. Using k-way partitioning, we "compress" the graph merging some nodes, obtaining a graph with k nodes. Once we have this smaller graph we can find k partitions on it and then the partition is projected back to the original graph. This algorithm tries to fix the fact that with recursive bisection method the first split into two partitions is crucial for the final result, and a suboptimal first split could lead to bad results, especially with many partitions.

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

Case	Recursive Bisection		Direct Multiway Partitioning	
Partitions	16	32	16	32
Luxembourg	186	289	184	309
usroads-48	600	952	572	913
Greece	309	513	303	471
Switzerland	684	1084	687	1023
Vietnam	278	441	233	416
Norway	275	465	285	445
Russia	614	1032	536	901

We can see how the performance on the partitioning with 16 partitions are quite similar, with the Direct Multiway partitioning having slightly better results in most of the cases. Like said before, when the number of partitions is higher the recursive bisection method starts to perform worse, this is because the result is too dependent from the first partition performed, and it's really difficult to take an optimal decision at a such early stage.

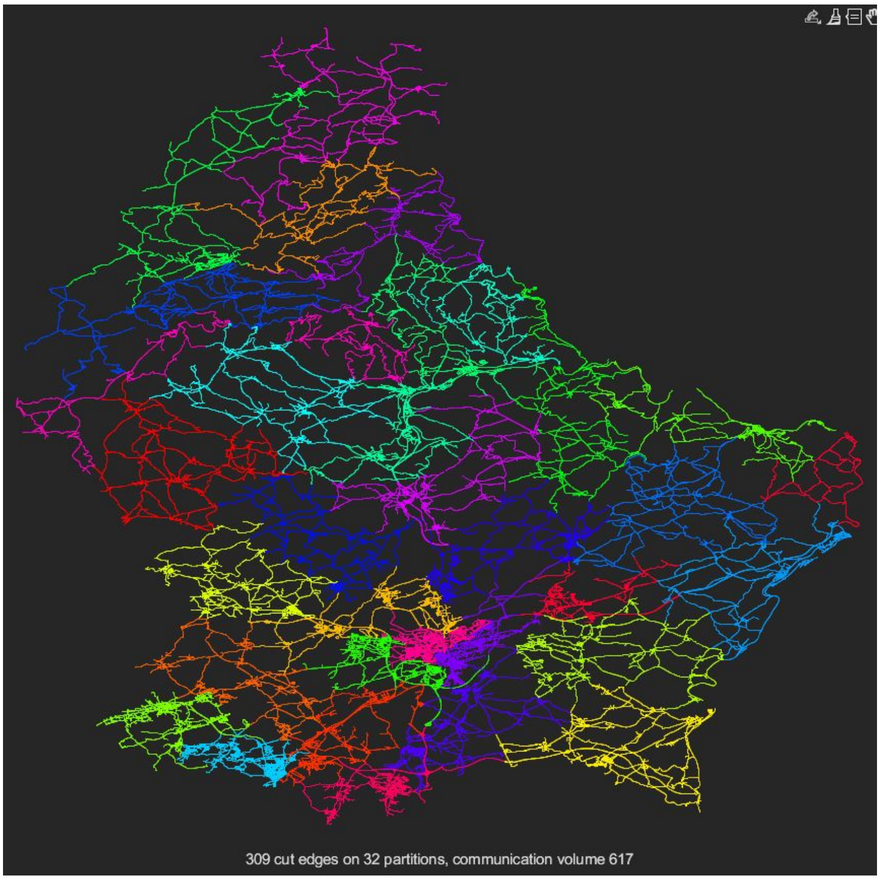

Figure 8: Visualizing the direct k-way partitioning of the graph containing the roads of Luxembourg.

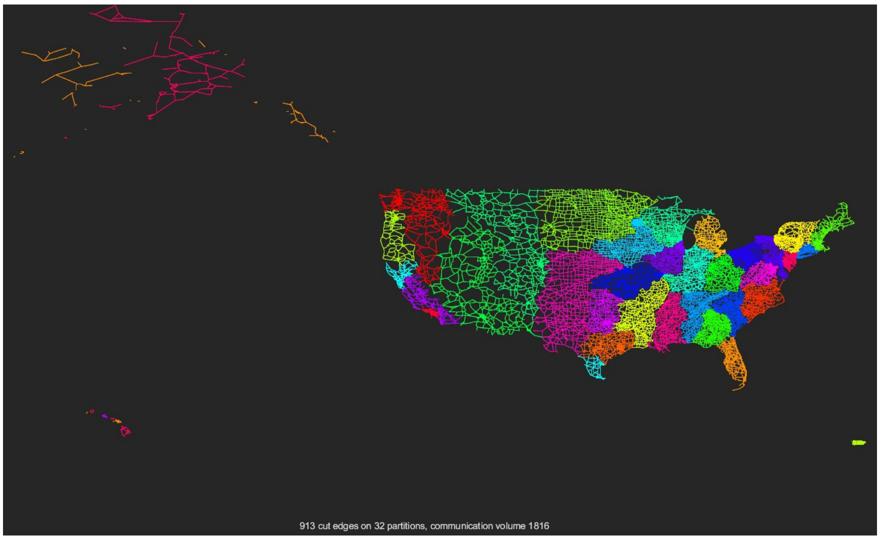

Figure 9: Visualizing the direct k-way partitioning of the graph containing the roads of the United States.

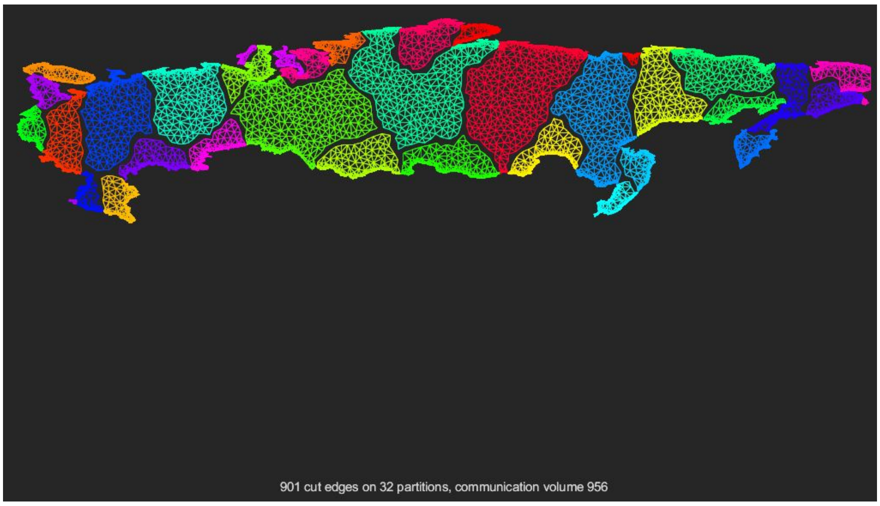

Figure 10: Visualizing the direct k-way partitioning of the graph representing Russia.

5. Task: Utilizing graph eigenvectors [25 points]

Provide the following illustrative results. Use the incomplete script `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.".

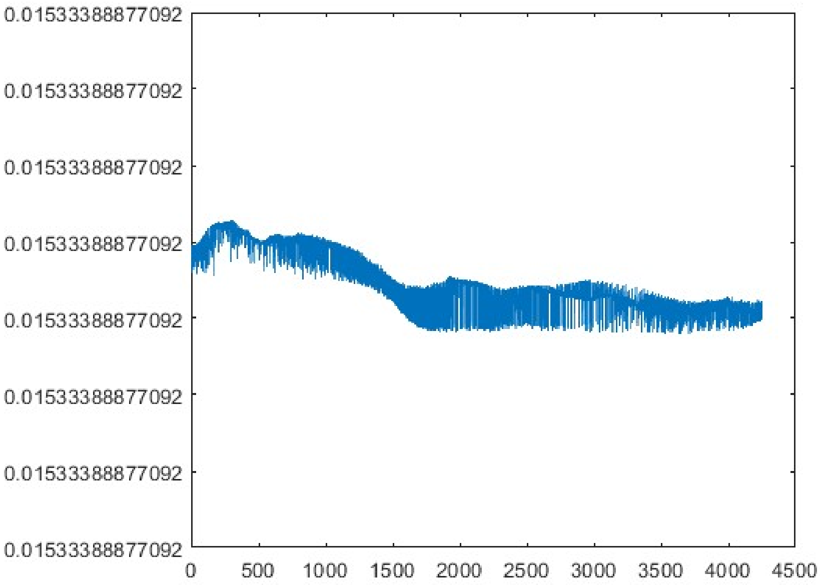

Figure 11: Eigenvector associated to the smallest eigenvalue of the Laplacian matrix.

We can notice how all the entries of the eigenvector x_1 associated to the smallest eigenvalue λ_1 are equal, let's understand why. A matrix A has an eigenvalue λ if and only if there exists a nonzero vector x such that $Ax = \lambda x$.This is equivalent to the existence of a nonzero vector x such that $(A - \lambda I)x = 0$. In our case, A isn't a generic matrix, but it's a Laplacian matrix, this means that the sum of the entries in each row is equal to 0. So we expect to have the eigenvalue λ_1 which is equal to 0 and in this case we need to find a vector x for which $Ax = 0$, and having all the rows sum equals to 0 this is true if all the entries of x are equals. (If the graph isn't connected, then we have a number of zero eigenvalues equals to the number of components).

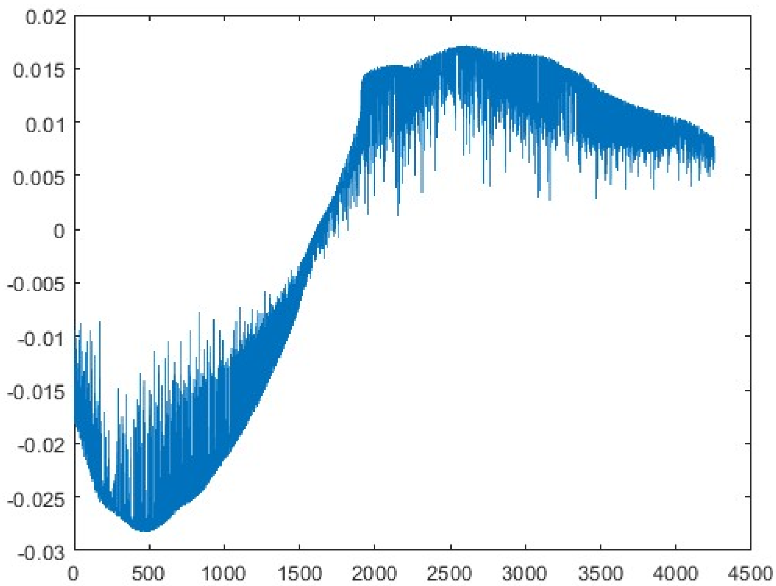

Figure 12: Eigenvector associated to the second-smallest eigenvalue of the Laplacian matrix.

This eigenvector x_2 is the one that minimizes the distance between the nodes in the same partition, and can be used to split the graph in two partitions simply deciding to put in the first partition all the nodes i for which $x_{2,i} < 0$ (or less than the median of x_2, we can see that also in this case the two conditions are pretty similar). In this case, the nodes in the graph are numbered in a "spatial" order, this means that consecutive nodes are near in the graph. For this reason, we can see that applying the partitioning condition expressed before the first part of the nodes will end up in partition one and all the others in partition two, which logically makes sense.

2. Plot the entries of the eigenvector associated with the second smallest eigenvalue λ_2 of the Graph Laplacian matrix $\mathbf{L}$.

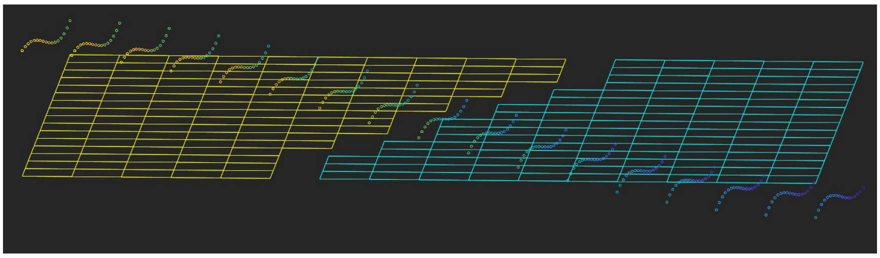

Figure 13: Graph: mesh3e1

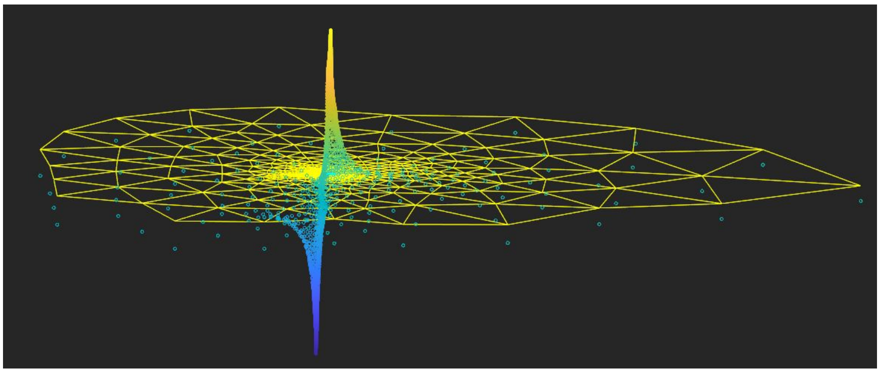

Figure 14: Graph: barth4

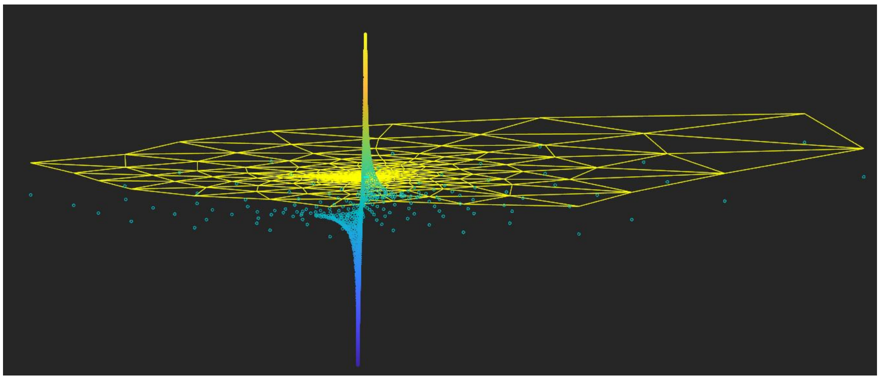

Figure 15: Graph: 3elt

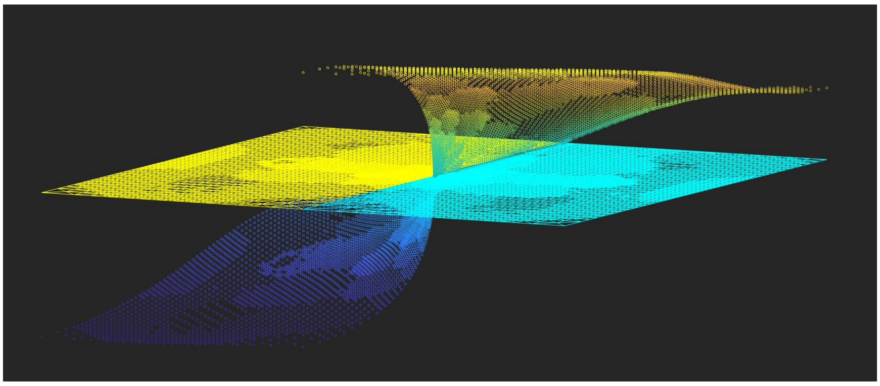

Figure 16: Graph: crack

In each of these 3-dimensional plots we have a 2-d visualization of the graph, where the nodes of the same partition have the same color. In the 3-dimensional space are plotted the entries of the Fiedler vector for each point. What we notice is that all the entries that are less than the median of them (which is basically zero) are under the 2-dimensional partition graph and all the corresponding nodes have the same color, the same thing could be said for all the nodes over the 2-dimensional plot. So, this shows how the entries of the Fiedler vector can be used to bi-partition a graph.

3. We will now see a spectral graph drawing method, which constructs the layout utilizing the eigenvectors of the graph Laplacian matrix $\mathbf{L}$. To produce the **spectral bi-partitioning** results, we use eigenvectors to supply coordinates. We 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 $\mathbf{L}$.

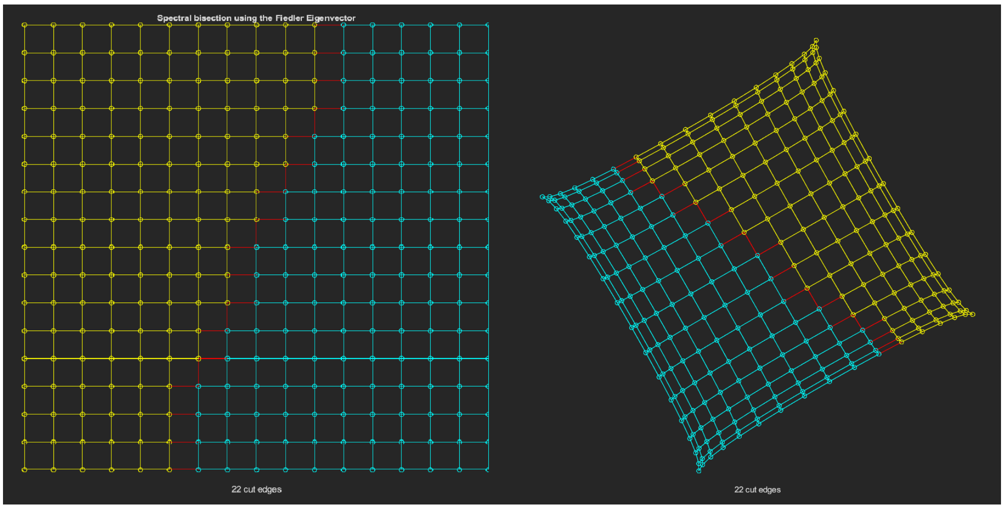

Figure 17: Spectral bisection and spectral visualization of the graph mesh3e1

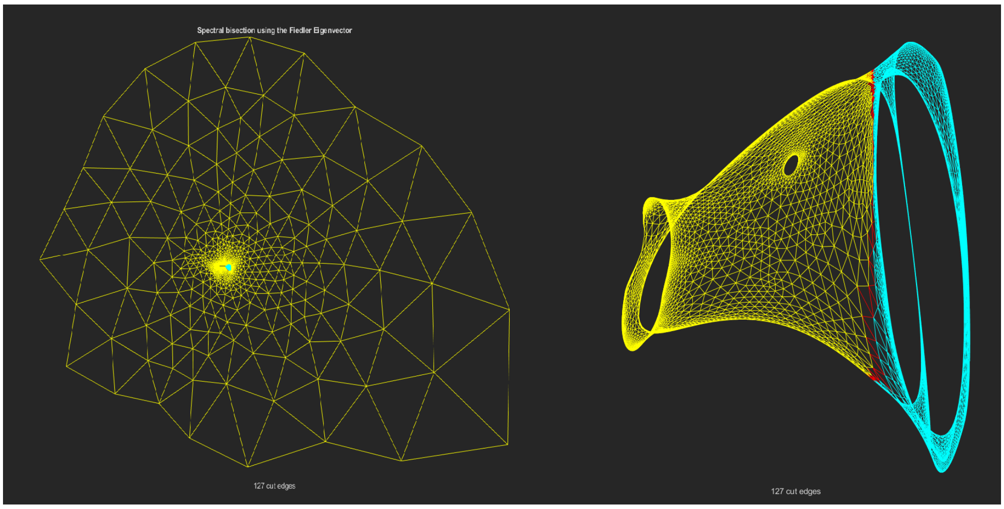

Figure 18: Spectral bisection and spectral visualization of the graph barth4

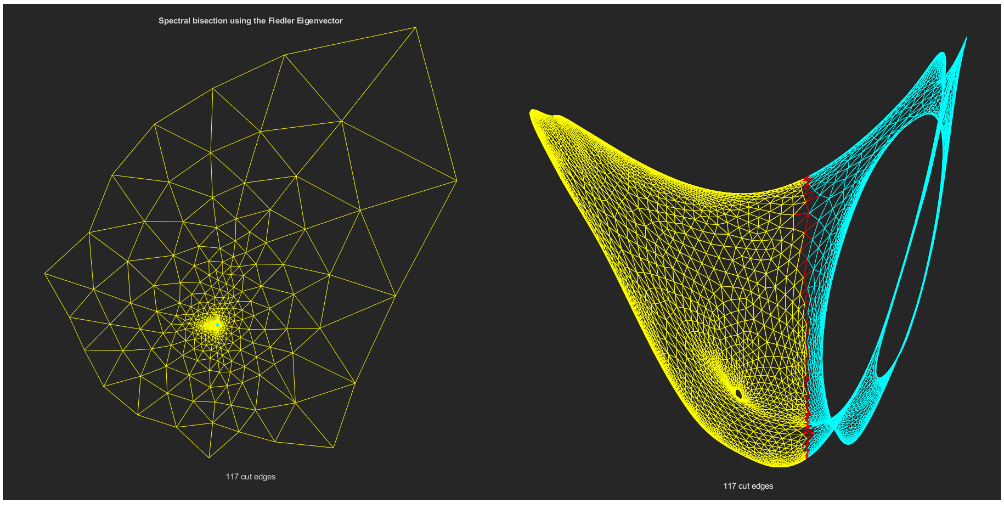

Figure 19: Spectral bisection and spectral visualization of the graph 3elt

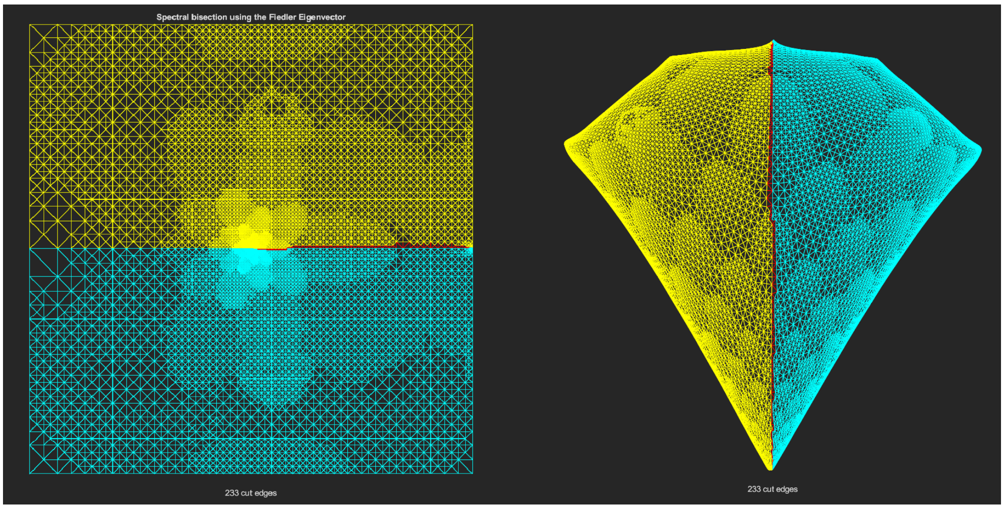

Figure 20: Spectral bisection and spectral visualization of the graph crack