Numerical Computing **2024**

Student: Lorenzo Galli **Discussed with:**

Solution for Project 3 **Due date:** Wednesday, 13 November 2024, 11:59 PM

---

1. Implementing various graph partitioning algorithms [50 points]

Graph partitioning is the process of partitioning a graph into two subgraphs in a way that the subgraphs have an approximately equal number of vertices, while also minimizing the number of edges between the two subgraphs. This computation can be done by using various algorithms.

The script Bench_bisection.m compares different graph bisection algorithms applied on different grids. The algorithm used are the Spectral bisection, the Coordinate bisection, the Inertial bisection, the Multilevel method and the Partition points by a plane method.

Before starting, let's quickly spend few words to have an idea about the theory behind those algorithms:

The Spectral Bisection method uses the eigenvalues and eigenvectors of the Laplacian matrix of the graph to partition nodes. It works by finding the second smallest eigenvector (Fiedler vector) and partitioning nodes based on the signs of this vector.

The Coordinate Bisection method uses the geometric position of data points for partitioning. In both 2D or 3D, this method can split based on x, y, z to form two groups.

The Inertial Bisection method calculates the center of mass and from it it constructs the inertia matrix. After calculating the eigenvalues and eigenvectors, it performs a projection of the nodes respect to the principal axis, and then we can take a cut point to divide them into two groups based on this projection.

The "multilevel" approach is used by the tool METIS and it first simplifies the graph by combining nodes into fewer, larger groups. Then, it divides this simpler graph into two parts. Finally, it improves the partition by refining it back to the original graph.

Finally, The Partition points by a plane approach is an useful function for dividing a set of points based on their relative position to a given plane.

In this section, we are asked to implement the Spectral graph bisection based on the entries of the Fiedler eigenvector, using the file bisection_spectral.m. In simple words, we are going to create the Laplacian matrix L associated to our graph, then we find the second smallest eigenvalue and its eigenvector, in order to split the graph into two subgraphs, based on the value of the components of the vector compared to the median of the fiedler vector (we could also use 0 as threshold).

The snippet is the following:

```
% 1. Construct the Laplacian matrix L = D - A.
G = graph(A,'omitselfloops');
L = laplacian(G);

% 2. Calculate the second smallest eigenvalue of L.
[V, ~] = eigs(L, 2, 'SM');

% 3. Label the vertices with the components of the Fiedler vector.
fiedler_vector = V(:,2);

% 4. Partition them around their median value, or 0 (sign).
median_value = median(fiedler_vector);
part1 = find(fiedler_vector < median_value);
part2 = find(fiedler_vector >= median_value);
```

Now, let's implement the Inertial graph bisection. This method, for a 2D environment, basically constructs a line such that half the nodes are on one side of the line, and half are on the other. Again, starting from the file bisection_inertial.m.

The snippet is the following:

```
% 1. Calculate the center of mass (baricentro).
center_of_mass = mean(xy);

% 2. Construct the matrix M.
Mxx = 0;
Mxy = 0;
Myy = 0;
for i = 1 : length(xy)
    Mxx = Mxx + (xy(i,1) - center_of_mass(1))^2;
    Mxy = Mxy + (xy(i,1) - center_of_mass(1)) * (xy(i,2) - center_of_mass(2));
    Myy = Myy + (xy(i,2) - center_of_mass(2))^2;
end
M = [Mxx, Mxy; Mxy, Myy];

% 3. Calculate the smallest eigenvector of M.
[eigenvector, eigenvalue] = eigs(M, 1, 'SA');

% 4. Find the line L on which the center of mass lies.
a = eigenvector(1);
b = eigenvector(2);
L = [-b, a];

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

After implementing the two code snippets, I can report the bisection edgecut for all toy meshes that are either generated or loaded in the script "Bench_bisection.m".

Mesh	Coordinate	Metis 5.0.2	Spectral	Inertial
grid5rec(12,100)	12	14	12	12
grid5rec(100,12)	12	12	12	12
grid5recRotate(100,12,-45)	22	12	12	12
gridt(50)	72	80	82	72
grid9(40)	118	128	140	118
Smallmesh	25	12	14	30
Tapir	55	24	58	49
Eppstein	42	41	47	45

Table 1: Bisection results

2. Recursively bisecting meshes [20 points]

For this section I updated the script Bench_rec_bisection.m in order to recursively bisect the finite element meshes loaded within the script in 8 and 16 subgraphs.

I used the previous built partitioning algorithms, as requested. The results of the recursive partition can be seen in the following table:

Case	P	Spectral	Metis 5.0.2	Coordinate	Inertial
mesh3e1	8	58	59	63	59
	16	58	59	63	59
bodyy4	8	1093	937	1065	1363
	16	1839	1614	1951	2212
de-2010	8	742	524	929	1080
	16	1340	893	1796	1996
biplane-9	8	510	447	548	647
	16	899	805	974	1092
L-9	8	705	607	631	828
	16	1122	988	1028	1378

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

It's interesting to notice that for the mesh3e1 case, the graph structure may not contain enough edges to create a significant difference in the partitioning between p = 8 and p = 16, since the number of cut edges remains the same with every partitioning algorithm. In fact, when the mesh is pretty small, dividing it into more partitions might not significantly impact the number of edges that cross between partitions.

Furthermore, partitioning algorithms like Spectral, Metis, Coordinate, and Inertial bisection rely on heuristics and approximations. For a small mesh, the algorithms might find a stable partitioning solution that doesn't change as the number of partitions increases from 8 to 16.

Finally, to conclude this section, in the new page there is the visualization of the results for the values p = 8 and p = 16 for the case "de-2010":

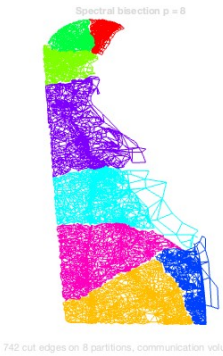

Figure 1: Rec. Spectral Bis., p = 8

Figure 2: Rec. Spectral Bis., p = 16

Figure 3: Rec. Metis Bis., p = 8

Figure 4: Rec. Metis Bis., p = 16

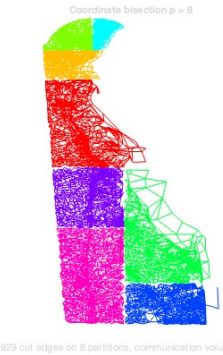

Figure 5: Rec. Coordinate Bis., p = 8

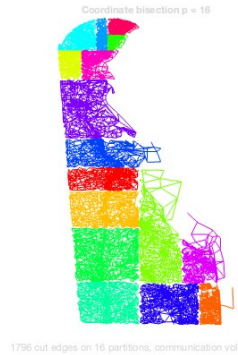

Figure 6: Rec. Coordinate Bis., p = 16

Figure 7: Rec. Inertial Bis., p = 8

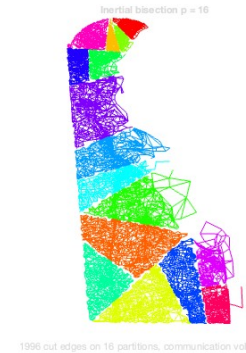

Figure 8: Rec. Inertial Bis., p = 16

3. Comparing recursive bisection to direct k-way partitioning [15 points]

As we have just seen, Recursive bisection partitions a graph into multiple subgraphs by repeatedly dividing it into two parts. Even if it can be computationally efficient, this approach has some limitations.

Each partitioning step splits the graph into two parts, and all the successive bisection are based on the quality of previous cuts. Not optimal choices in the early stages could lead to imbalanced or not optimal partitions in later stages.

Furthermore, each partitioning decision is made based on local information that are available at that particular split. Since this method doesn't account for the entire structure of the graph globally but for the local split previously made, this limited scope can lead to partitions that are not optimal when compared with the full graph context. For this reasons some partitions might be imbalanced in terms of the number of nodes or edge cuts.

Summing up, in the Recursive bisection, we start with the entire graph and repeatedly dividing it into two parts, until the desired number of partitions is reached. So, basically, it's a top-down approach: starting with the entire graph (high-level problem) and reducing it by successively splitting the graph.

METIS also employs a multilevel k-way partitioning algorithm. The multilevel k-way partitioning method follows a sort of bottom-up approach. First of all, the graph is coarsened down, meaning it's reduced until it becomes small enough to be handled easily ("base case" graph).

Then, the k-way partitioning is applied to this simplified version of the graph, before the partitions are projected back up to the original graph level through successive uncoarsening stages.

This approach is "bottom-up" in the sense that it starts with a simplified version of the graph and then builds up toward the original graph structure.

For this last section, after completing the implementation of the multiway partitioning in the file Bench_metis.m, we want to compare the cut obtained from Metis 5.0.2 after applying the recursive bisection and the direct multiway partitioning for the graphs "helicopter" and "skirt".

Personally, what I'm expecting is that the two algorithms might perform similarly for smaller graphs and fewer partitions (for example helicopter with p = 16), while I think that the direct partitioning might perform better for larger graphs and larger p values.

The results are summarized in the following table:

Partitions	**Helicopter**	**Skirt**
16 - recursive bisection	337	3315
16-way direct partition	336	3314
32 - recursive bisection	553	5989
32-way direct partition	499	6239

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

The results are actually very interesting. As I predicted, for p = 16 the difference between the two algorithms is minimal for both graphs. I thought we would have noticed more difference increasing the dimension of the graph (skirt is bigger than helicopter), but in reality there is only 1 edge cut of difference. This suggests that for relatively small partition counts, both methods might perform similarly.

Surprisingly, for p = 32 things are different than I was expecting. For helicopter with p = 32 the Direct partitioning is better than Recursive partitioning with 54 fewer edge cuts. However, for skirt with p = 32 the Recursive partitioning is better than Direct partitioning with 250 fewer edge cuts. Apparently, for this graph at p = 32, the Recursive partitioning might have a better behavior.

In conclusion, I think that we could have a general idea on the performances of the two algorithms, but in reality it really depends on the structure and the connections of the graphs. The Recursive partitioning could generate a lot of edge cuts if the structure is weird, since it doesn't care about the overall shape of the graph, but it performs based on the previous split. However, in this case, we have seen that the algorithm works properly.

Finally, let's visualize the partitioning results for both graphs for 32 partitions, using the "Rotate 3D" option from the MATLAB figure selection "Tools":

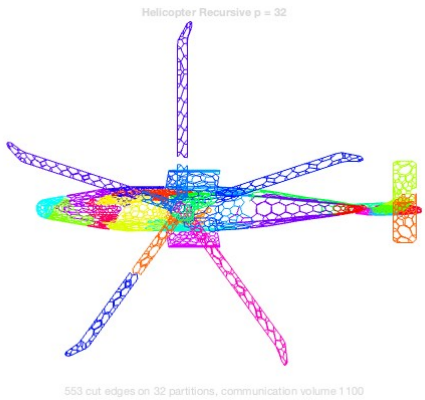

Figure 9: Helicopter recursive bisection 32

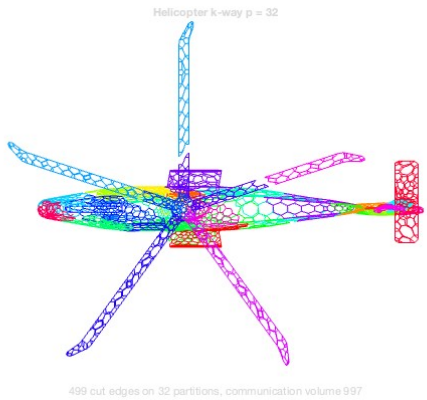

Figure 10: Helicopter 32-way direct bisection

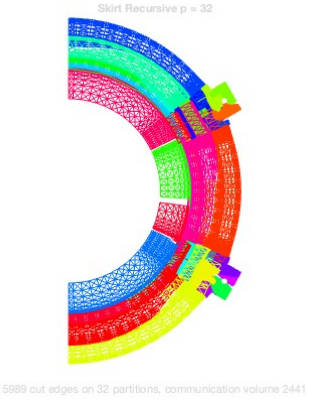

Figure 11: Skirt recursive bisection 32

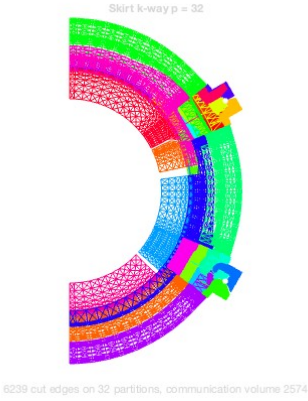

Figure 12: Skirt 32-way direct bisection

Sources:

Spectral partitioning

Inertial partitioning

Recursive partitioning

K-way direct partitioning