Università della Svizzera italiana | Faculty of Informatics | Institute of Computing CI

Numerical Computing **2022**

Student: Albert Cerfeda Discussed with: Alessandro Gobbetti

Solution for Project 3

Due date: Wednesday, November 9, 2022, 11:59 PM

Numerical Computing 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 iCorsi (i.e. in electronic format).
- Provide both executable package and sources (e.g. C/C++ files, 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.

Contents

1. The assignment

1.1. Implement various graph partitioning algorithms [50 points]

For an unweighted graph $G = (V, E)$ its adjacency matrix $A^{n \times n}$ where $n = |V|$ is defined as follows:

$$A_{ij} = \begin{cases} 1 & \text{if } (ij) \in E \\ 0 & \text{otherwise} \end{cases},$$

such that the Adjacency matrix contains a 1 when there is an edge between two vertices and 0 otherwise. For undirected graphs, the adjacency matrix is symmetrical.

$$\mathbf{A} = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix} \text{ Adjacency matrix for a fully connected undirected graph where } |V| = 3$$

Spectral Graph Bisection

The Spectral Graph bisection involves partitioning the **Fielder vector** (i.e the eigenvector associated with the second smallest eigenvalue) of the graph **Laplacian matrix** $L \in \mathbb{R}^{n \times n}$ around some value m.
The **graph Laplacian** is a symmetric positive semi-definite matrix that expresses various graph properties. It is computed through the adjacency matrix A and weight matrix D.
Given a weight matrix D:

$$\mathbf{D} := \sum_{j=1}^{n} A_{ij} = \begin{bmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 2 \end{bmatrix} \text{ such that } D_{jj} \text{ contains the total degree of vertex } V_j$$

we define the **graph Laplacian matrix** as follows:

$$\mathbf{L} := \mathbf{D} - \mathbf{A} = \begin{bmatrix} 2 & -1 & -1 \\ -1 & 2 & -1 \\ -1 & -1 & 2 \end{bmatrix}$$

The threshold value m around which we define the two partitions can be chosen in two ways:

1. **median value** of the chosen eigenvector
   The two partitions will have homogeneous size.
2. $m = 0$
   Guarantees a lower amount of edge cuts between nodes.

```
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

    D = Diagonal(vec(sum(A,dims=1)))
    L = Matrix(D-A);

    e = eigen(L);
    w = e.vectors[:,sortperm(e.values)[2]];

    # m = median(w);
    m = 0
    return map(x->x < m ? 1 : 2, w);
end
```

Notice how the chosen threshold value is 0. This yields fewer edge cuts between the nodes of the two partitions.

Inertial Graph Bisection

Inertial bisection partitions the vertices based on their geometrical location, expressed with a coordinate tuple (x_i, y_i).
We trace a line l across the space and partition the vertices based on whether they are on one side of the line or the other.
The sum of distances of the vertices from the line l is defined as follows:

$$\begin{aligned}\sum_{i=1}^{n} d_i^2 &= \mathbf{u}^T \begin{bmatrix} S_{xx} & S_{xy} \\ S_{xy} & S_{yy} \end{bmatrix} \mathbf{u} \\ &= \mathbf{u}^T \mathbf{M} \mathbf{u}\end{aligned}$$

Matrix M expresses the distance of the vertices from the **center of mass**, defined as follows:

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

In order to minimize the sum of distances as much as possible we choose u to be a vector orthogonal to the smallest eigenvector for matrix M.

```
function inertial_part(A, coords)
    tuples = eachrow(coords)

    cm = reduce((acc,coord)->(acc[1]+coord[1], acc[2]+coord[2]),tuples, init=(0,0));
    cm = (cm[1]/length(tuples),cm[2]/length(tuples))

    sxx = reduce((sxx,coord)->sxx+(coord[1]-cm[1])^2,tuples,init=0)
    syy = reduce((syy,coord)->syy+(coord[2]-cm[2])^2,tuples,init=0)
    sxy = reduce((sxy,coord)->sxy+(coord[1]-cm[1])*(coord[2]-cm[2]),tuples,init=0)
    M = [ sxx sxy ; sxy syy]

    e, v = eigs(M,nev=1,which=:SR)
    v = v[:,1] # Smallest eigenvector
    w = [ v[2],-v[1] ];

    p = partition(coords,w)
    return map(x->xp[1] ? 1 : 2, collect(1:size(A)[1]))

end
```

Running the bisective benchmark yields the following results:

Table 1: `bench_bisection.jl` results

Mesh	Coordinate	Metis v.5.1.0	Spectral	Inertial
grid(12, 100)	12.0	12.0	12.0	12.0
grid(100, 12)	12.0	14.0	12.0	12.0
grid(100, 12, -/4)	22.0	14.0	12.0	12.0
gridt(50)	73.0	80.0	66.0	72.0
gridt(40)	59.0	62.0	52.0	59.0
smallmesh	25.0	13.0	12.0	30.0
tapir	55.0	24.0	18.0	49.0
eppstein	42.0	40.0	42.0	45.0

We notice how `Coordinate` and `Metis` are the less accurate. The randomness of M*etis* has been taken in account of and after running the benchmark multiple times, the average results for M*etis* are in the likeliness of the ones reported in the table.
S*pectral* although being the most accurate is also the slowest as computing the eigenvectors and eigenvalues of large matrices such as matrix M is computationally very expensive.

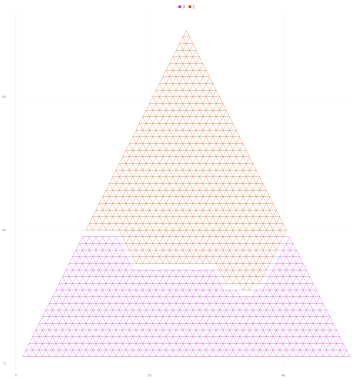

Figure 1: `Metis` algorithm. **80 edge cuts**

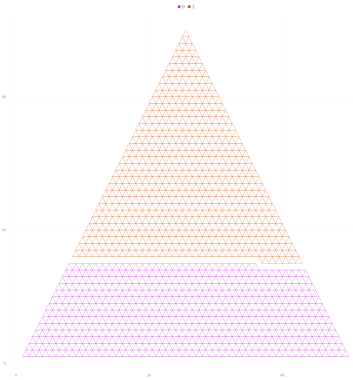

Figure 2: `Inertial` algorithm. **73 edge cuts**

Partition comparison for mesh `tapir`

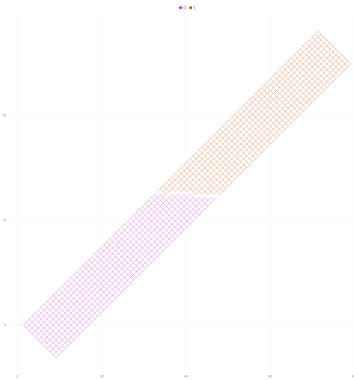

Figure 3: `Coordinate` algorithm. **22 edge cuts**

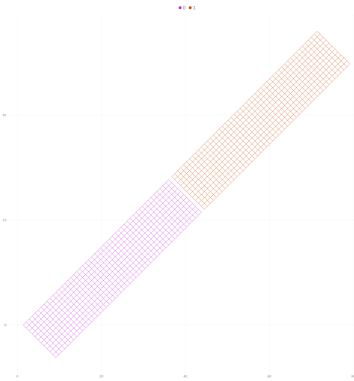

Figure 4: `Spectral` algorithm. **12 edge cuts**

Partition comparison for mesh `smallmesh`

1.2. Recursively bisecting meshes [20 points]

By running the partitioning algorithms recursively, we split each graph/subgraph G into two more subgraphs $G^{'}$ and $G^{''}$. Using recursion can pay to our advantage if we intend to parallelize and scale our partition computation.
We therefore split the graphs into 2^n subgraphs where n is the number of recursion levels.
One fundamental difference though is that the two partitions that each recursion level yields **influence all the next recursions**. For $n = 3$ and $n = 4$ we split the graph into 4 and 8 subgraphs respectively.
By running the benchmark we notice how the `Spectral` really stands out for being the affected the most performance-wise from computing the eigenvectors and eigenvalues of very large graphs. Generally, the recursive approach results in greater edge cuts. `Inertial` yields low edge cuts and is very fast as it always computes eigenvectors of matrix $M \in \mathbb{R}^{2\times 2}$.

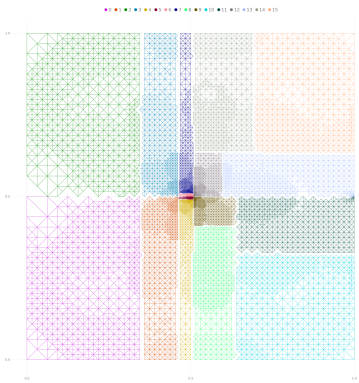

Figure 5: `Coordinate` algorithm.**1861 edge cuts**

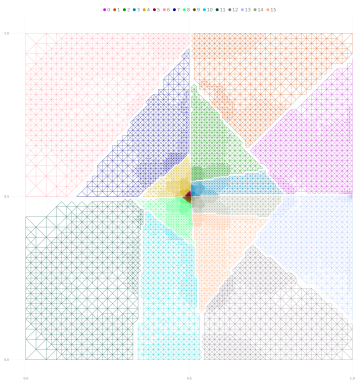

Figure 6: `Inertial` algorithm. **1618 edge cuts**

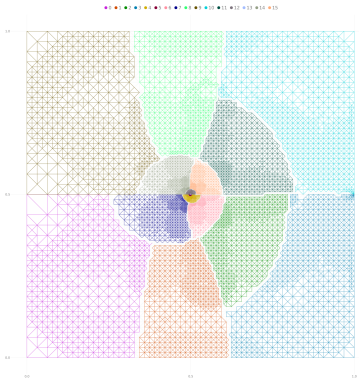

Figure 7: `Spectral` algorithm. **1303 edge cuts**

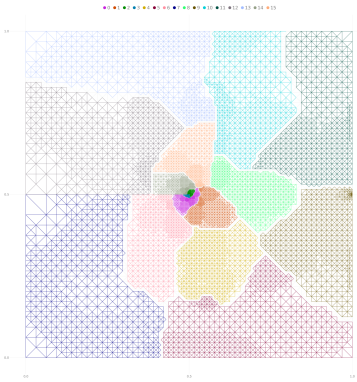

Figure 8: `Metis` algorithm. **1276 edge cuts**

Partition comparison for mesh `crack` with $n = 4$ (16 partitioned subgraphs)

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

Mesh	Spectral 8 parts	Spectral 16 parts	Metis 8 parts	Metis 16 parts	Coordinate 8 parts	Coordinate 16 parts	Inertial 8 parts	Inertial 16 parts
airfoil1	327.0	578.0	311.0	580.0	516.0	819.0	578.0	904.0
netz4504_dual	105.0	174.0	100.0	154.0	127.0	198.0	122.0	202.0
stufe	124.0	216.0	112.0	197.0	123.0	228.0	135.0	268.0
3elt	372.0	671.0	418.0	651.0	733.0	1168.0	880.0	1342.0
barth4	505.0	758.0	491.0	773.0	875.0	1306.0	892.0	1350.0
ukerbe1	118.0	224.0	125.0	241.0	225.0	374.0	280.0	469.0
crack	804.0	1303.0	759.0	1276.0	1344.0	1861.0	1061.0	1618.0

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

Running the