

Graph Partitioning Project

September 4, 2023

Gerardo Maruotti s317642

Simone Varriale s315962

Description of the graph

The graph is described by $G=(V,E)$

- V is the number of nodes of the graph
- E is the number of edges of the graph

The graph is weighted:

- $W_1: V \rightarrow \mathbb{R}$ which maps vertices to real-valued weights
- $W_2: E \rightarrow \mathbb{R}$ which maps edges to real-valued weights

The implementation of the graph is without coordinates so it is not embedded in space and it is also undirected.

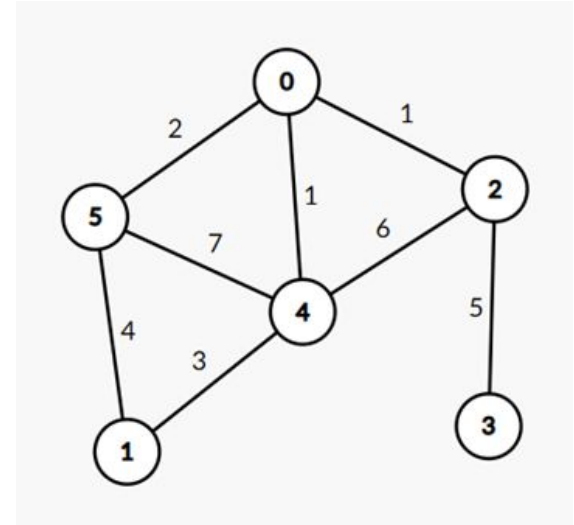


Figure 1: Example of weighted graph

Our goal

Implement algorithms that performs p -way partitioning which is the division of graph G into p sub-graphs in which their vertices do not overlap.

Two specific properties need to be satisfied:

- The sum of the weights of the nodes in each subgraph is balanced
- The sum of the weights of the edges crossing between subsets is minimized

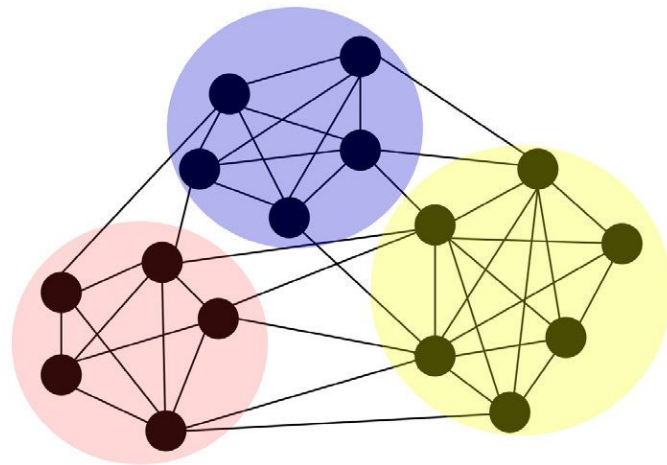


Figure 2: Example of p -way partitioning

Recursive Spectral Bisection

Recursive Spectral Bisection was proposed by Pothen et al. for the first time in 1989 in order to implement an efficient way to compute the ordering of a matrix in a parallel way.

It uses an adjacency matrix and the Laplacian matrix from which it compute the eigenvectors used to perform partitioning.

It is called spectral algorithm due to three reasons:

- The spectral methods employs global information to compute separators
- It makes a continuous choice in order to belong to one partition instead of having a discrete choice
- The dominant computation is an eigenvector computation through Lanczos or similar algorithm.



RSB pseudocode

MatDegree \rightarrow G.getMatDeg()

MatAdj \rightarrow G.getMatAdj()

L \rightarrow MatDegree-MatAdj

Fiedler Vec \rightarrow eigenSolver(L).col(1)

SortedIdxs \rightarrow sortIndices(FiedlerVec)

Partition size \rightarrow G.numofnodes()/p

for each partition

 Partitions[i][sortedIDxs[j]] = true;

Compute partitions weight

Cut size \rightarrow calculateCutSize(partition)

Considerations:

- The algorithm might struggle with graphs that have irregular structures or complex connectivity patterns
- It often produces well-balanced partitions with a relatively small number of cut edges due to the global informations it knows



Multilevel

Multilevel has been introduced to address the problem of big graph due to high computational time.

The main phases are:

- Coarsening: procedure to create a hierarchy of smaller graphs
- Uncoarsening: procedure to reconstruct the original graph

Basically, it is an approximation of the original graph in order speed up the computation of the partition.

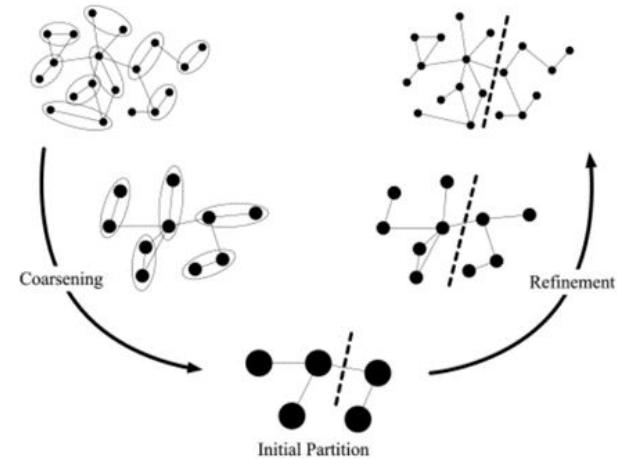


Figure 3: Example of multilevel

Multilevel: how to create smaller graph?

Matching \rightarrow HEM(G)

for each match:

Coarse \rightarrow matching nodes, weights, adjacency

G1.setNode(G1.lastNodeID, weight, coarse)

Process unmatched nodes \rightarrow Coarse with same parent node

end for

for each match

For each node in G1:

Set G1 edges based on the adjacency of G node

Return new graph

- Heavy Edge Matching uses the weighted edges of the graph to decide which nodes to merge.
- The "heavier" edges, typically those with higher weights, are then selected to form pairs or matchings while ensuring that nodes are not repeated in multiple matches.
- This process aims to maximize the total weight of matched edges, often resulting in more balanced and optimized partitions when applied in graph partitioning algorithms.



Multilevel RSB

```
fv=fiedler(G) → MatDegree → G.getMatDeg(), MatAdj→G.getMatAdj()
Median→computeMedian(fv)
L→MatDegree-MatAdj
for each node of G
    if(fv[i]≤median)
        partition[i]=false
    else
        partition[i]=true
Return partition

if(!stoppingCondition)
    G1 = coarsening(G)
    fv1=Fiedler(G1)
    fv= interpolate(fv1,L,sizeNodes)
    fv=RQI(fv,L ,sizeNodes)
else
    fv=eigenSolver(L).eigenvectors.real.col(1)
Return fv
```

The most important part of the function is the Fiedler function where we can find the important part of the algorithm:

- Coarsening
- Interpolation
- Rayleigh Quotient Iteration

Interpolation and RQI

After the computation of the first fiedler vector on the smallest graph, the algorithm have to approximate the FV in order to make it represent the original graph. This is done in the following way:

- Interpolation: it expands the fiedler Vector in order to make it bigger and come back to the original form. Precisely, given a Fiedler Vector from a contracted Graph G' with $n=|V1|$ the interpolation construct an expansion of this vector that can be used as an approximation of the original graph G . There is a mapping of $V1$ to V , from that we obtain a new fiedler vector with the same values and the remaining nodes are set by averaging the elements of their neighbors set during the phase of injection.
- For the refinement phase, RQI is used to help Lanczos algorithm to produce a good approximation of the Fiedler vector. RQI solves a linear system in each iteration, then through the solution a new fiedler vector is computed and the iteration are repeated until the convergence is reached.



Parallel versions

Threads and mutex synchronization were used.

Parallel parts in RSB:

- Laplacian computation
- Fiedler vector median calculation
- Partitions making

Parallel parts in Multilevel RSB:

- Laplacian computation
- Interpolation

Other computations were unchanged, as they relied on the Eigen library.

Parallel Fiedler:

MatDegree \rightarrow *G.getMatDeg()*

MatAdj \rightarrow *G.getMatAdj()*

Compute Chunk size = *sizeNodes/numThreads*

Create Thread for each row of L

L[row] \rightarrow MatDegree[row] - MatAdj[row]

Join Threads

if(!stoppingCondition)

G1 = coarsening(G)

fv1 = Parallel Fiedler(G1)

fv = Parallel interpolate(fv1, L, sizeNodes)

fv = RQI(fv, L, sizeNodes)

else

fv = eigenSolver(L).eigenvectors.real.col(1)

Return fv

Parallel Interpolation:

Compute first part of FV

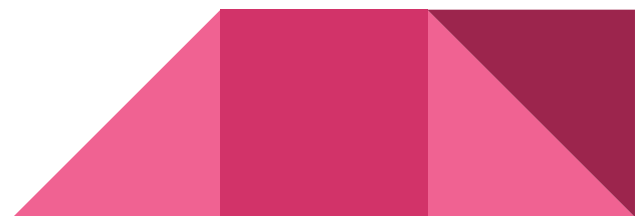
Compute Chunk size = *sizeNodes/numThreads*

Create Thread for each missing row of FV

Compute sum of the neighbors

Assign average

Return fv




Benchmark file

Main structure:

- First row: number of nodes
- Second row: number of edges
- Following rows: (id,weight) of the nodes
- Following rows: (id, id, weight) of the edges


Since our computing facility was limited the analysis has been performed on graph that had from 50 to 500 nodes.

Name of the file (with number of nodes and edges) and total weight of nodes and edges:

- graph_50_128.txt, 300, 671
 - graph_100_256.txt, 587, 1320
 - graph_250_640.txt, 1419, 3400
 - graph_500_1024.txt, 2740, 5578
- 

Results and conclusion

From the analysis RSB tends to excel in balance, particularly for smaller graphs and fewer partitions, while MLRSB offers superior speed, making it highly efficient for larger graphs. Cut size optimization often tilts in favor of MLRSB, despite higher memory consumption. It is noticeable that memory allocated by the multilevel version is bigger due to the saved coarsened graph in memory. This is a pattern that would be present in all the analysis made on the different graph.

- **graph_50_128.txt:** Not good performance. The normal version of the algorithm perform better than the multilevel versions in terms of balance factor and cut size, probably due to the approximation made during the coarsening procedure. Non acceptable results in terms of balance for 4 and 8 partitions.
 - **graph_100_256.txt:** Better performance than before, but with an increase of execution time.. Balance factor for 2 partitions is really good for RSB algorithm that in its sequential version performs really well, but in terms of cut size is the multilevel one which minimizes it the most and this applies for 2,4 and 8 partitions.
 - **graph_250_640.txt:** Very good performance in terms of balance factor. MLRSB is 50 times faster than normal RSB with also better results in terms of balance factor, Cut size instead is worse here.
 - **graph_500_1024.txt:** MLRSB is still 16 times faster than the normal RSB. In this case the partitions produced are a little bit more unbalanced, but still with an optimal results. Cut size again is still less for MLRSB but its results are a little bit worse with 4 and 8 partition while RSB remains basically consistent with the results.
- 

Thanks for your attention!

