Community detection-Report

Vaishnav K.V.

vaishnavk@iisc.ac.in

Q1. Spectral decomposition -one iteration

# Data preprocessing for Facebook data:

• I got the edge list matrix directly from the txt file using the NumPy-based function.

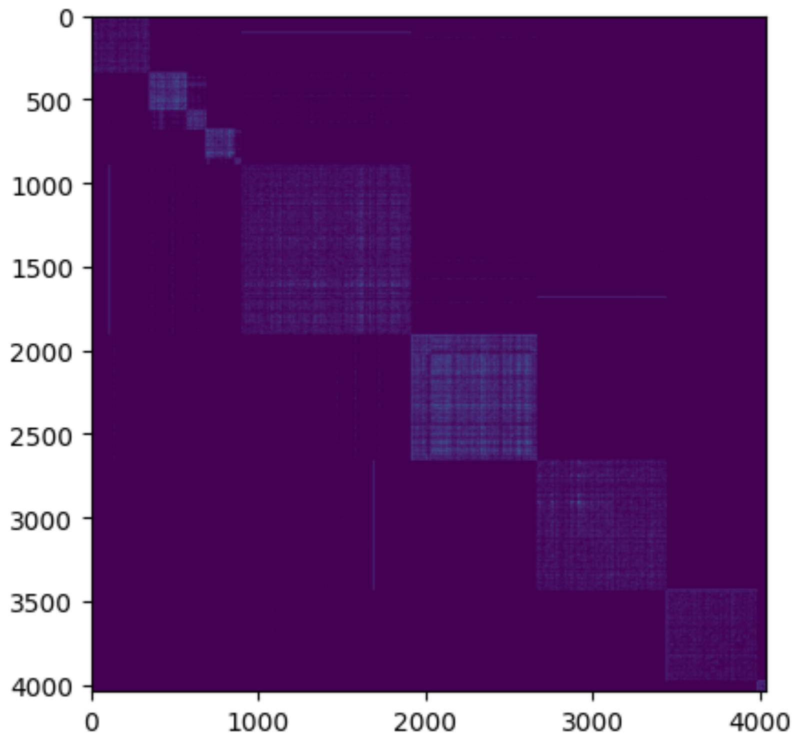

Fig:Adjacency matrix for Facebook data

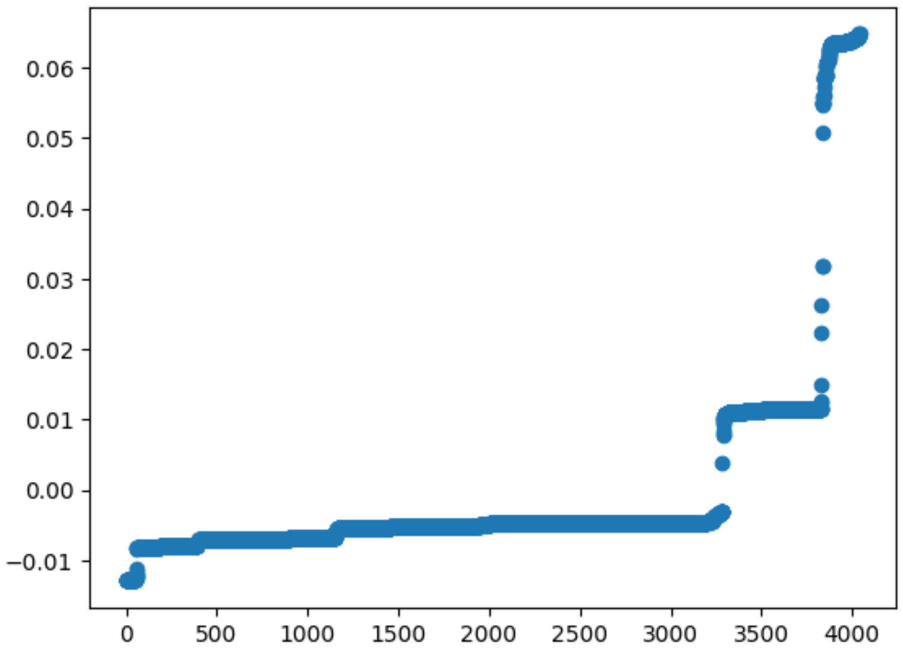

Fig. sorted Fiedler vector for the Facebook dataset.

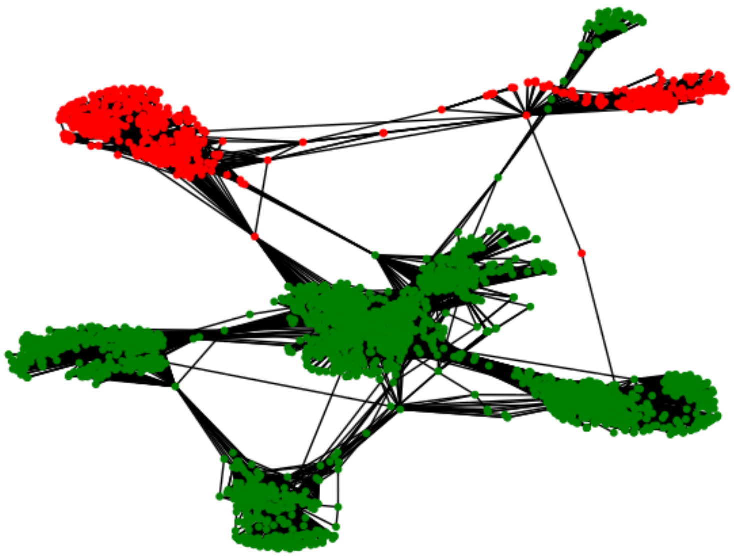

Fig.Partitioned(after one iter) graph plotted using networkx for Facebook dataset.

Data preprocessing for bitcoin data:

- Taken the two columns and neglected the weight value. Since the min node value is 1, reduce it to 0 as the starting node.

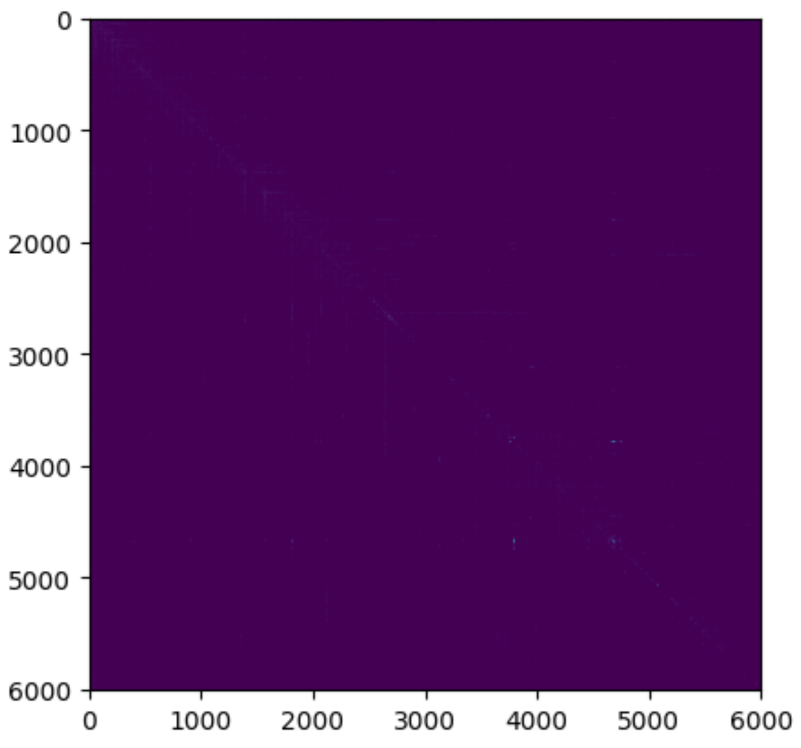

Fig: Adjacency matrix for bitcoin dataset

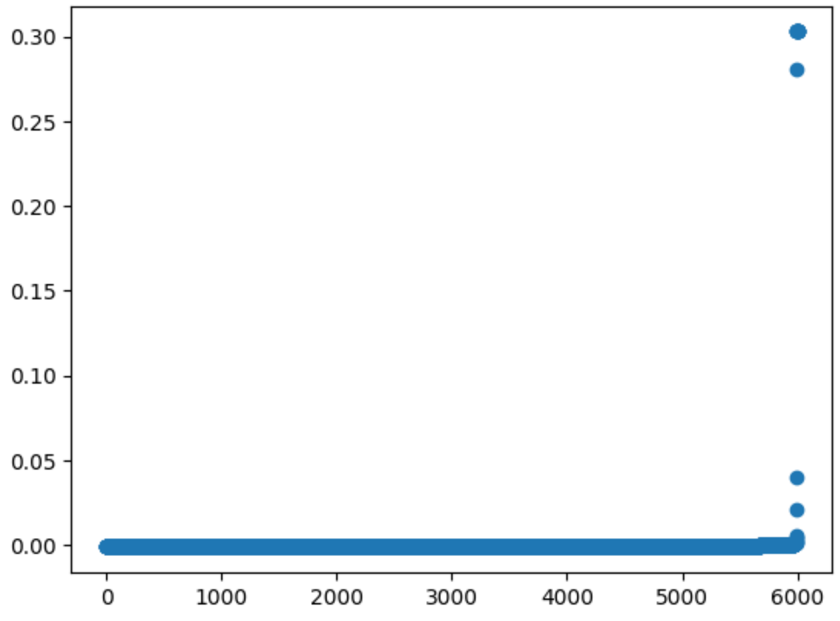

Fig: Sorted Fidler vector for the bitcoin dataset.

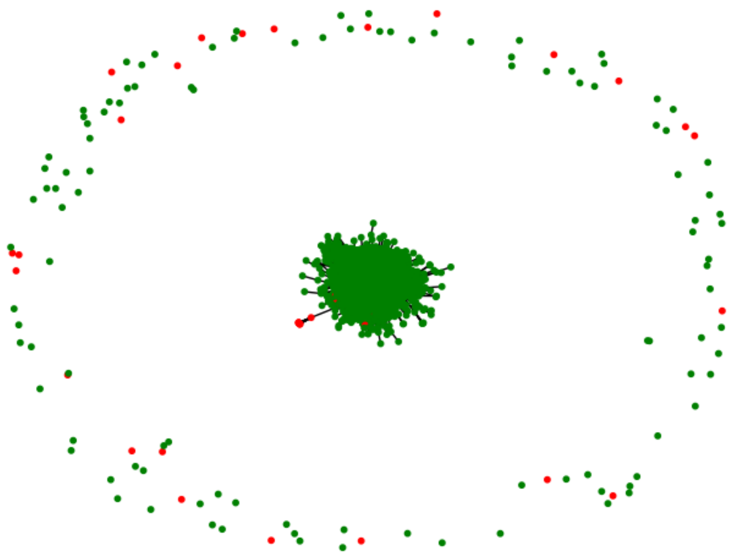

Fig: partitioned graph(after one iter) plotted using networkx for bitcoin dataset

Implementation :

1. Given the edge list.
2. Get A.here I have used a local function to do this task.
3. L=D-A, now get perform the Eigen decomposition of the L matrix. We knew that L is a symmetric matrix, so I used np. eigh function, which utilises the symmetric property to speed up the computation process.
4. Second small Eigenvalue and Fidler vector are found.
5. taking the sign(Fidler) will give the community it belongs to.
6. Sorted Fidler vector element-wise, to get the sorted_fidlr vector.
7. using matplotlib scatter plot sorted, Fidler is plotted.
8. Using networkx's plotting functions, partitioned graphs are drawn.

Note:

The plotting lines are commented inside the spectralDecomp_OneIter() function.

Q2. Extending spectral decomposition idea for multi clusters.

The approach: Use the spectralDecomp_OneIter function recursively on the result clusters.

Stopping criterion: The recursion will continue until all the elements are of the same sign. Then it will backtrack.

Implementation:

It was difficult for me to implement this idea. It took a long time to figure it out.

Now we have a method to find two communities in the graph. We can apply the same alg to the two communities individually. Here initially, nodes are unassigned, to indicate that I have given a -1 value during initialisation. Within the functions, local functions are defined as a helper functions. To all such, a recursive approach can get us results. But the problem here is we need to keep track of the node numbers while moving from one function to another(in-depth). For this, "transform" takes global to local and "inverse_transform" takes local to global. This recursive tree should evolve till all the signs of vectors are the same; it can be negative and positive. In short, we keep partitioning into two each time, until all the signs are the same in a vector. **As an edge case**, some nodes were unallocated because of inter-community edges. To remove this condition, a technique is deployed. We will classify those nodes into nearby clusters based on their number of occurrences. This is like a hack based on heuristics. But in terms of results, it is giving good results. It takes a long to write the details, the above is just a simplified version.

Q3. Plots for Recursive Spectral decomposition

Facebook dataset:

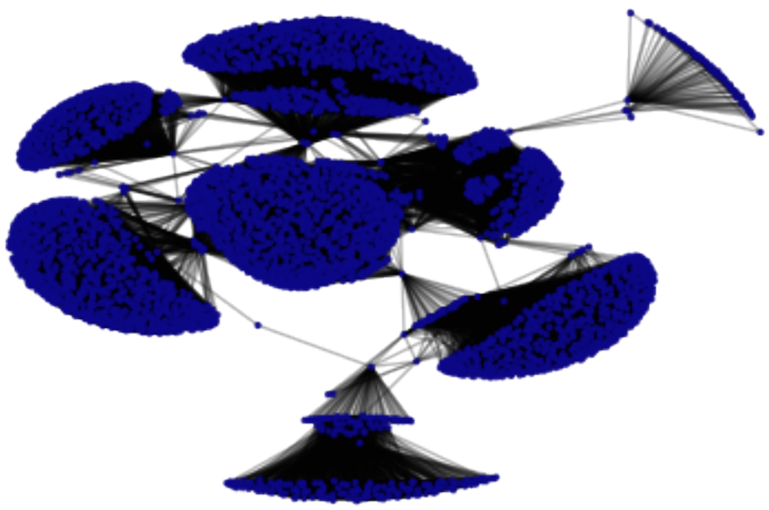

Fig: Partitioned(after recursively) graph plotted for Facebook data

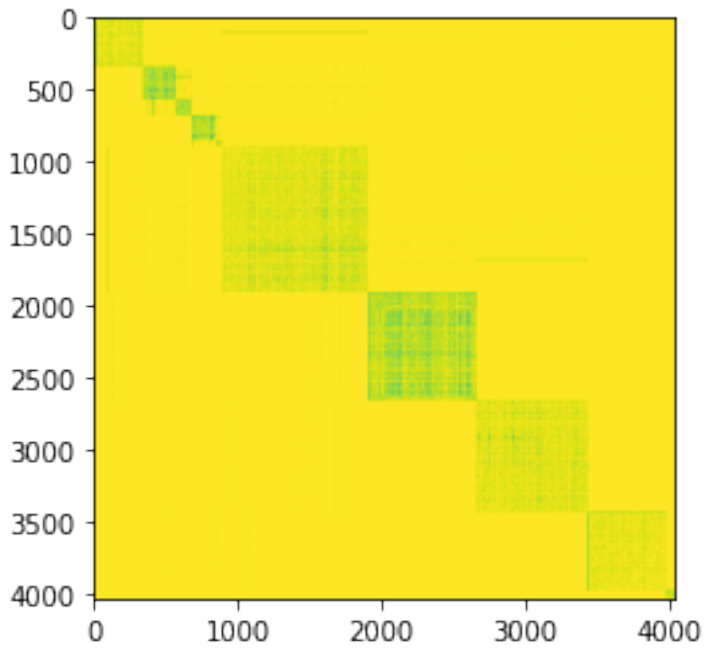

Fig: clustered adjacent matrix for Facebook dataset

Bitcoin Dataset:

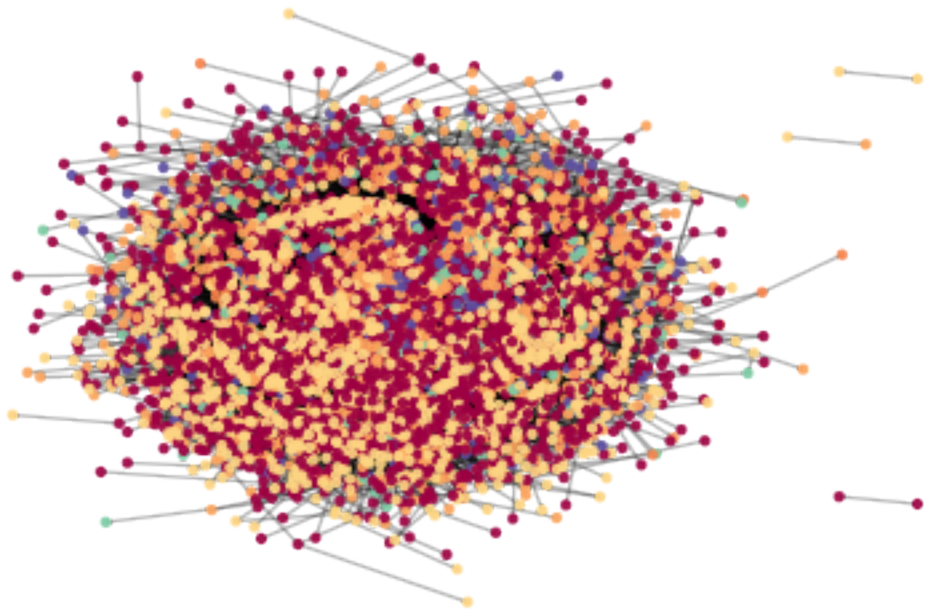

Fig: Partitioned(after recursively) graph plotted for Bitcoin data

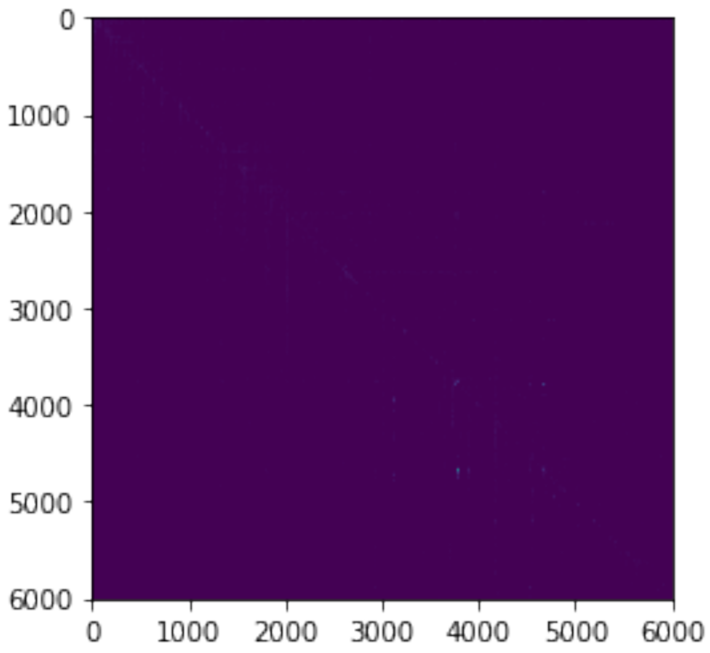

Fig: clustered adjacent matrix for Bitcoin dataset

Q4.Louvain algorithm:

Implementation:

Here we iterate over vertices in the graph. For each vertex, iterate over the communities (but in the implementation, we are iterating only over neighbouring vertices). For the current vertex, there will be its parent community. We will check the differential modularity gain (delta Q) when the vertex is moved from the parent community to the iterating community. After iterating over all the communities, we will choose the target community that yields the maximum Delta Q. Now, we will move this vertex to the founded optimal community(in a greedy sense). We will perform this iteratively until no more change occurs.

In summary, we have founded communities in the graph that yields high modularity value.

Note:

- The direct formula for delta Q used in the script is obtained via simplification of the original Delta Q expression.

Challenges faced:

- The problem I faced while implementing this part was choosing the right data structure. Initially, the implementation was completely different, but it took a significant amount of time during the run.
- As per the algorithm that is given in the slide, one update was performed only after iterating over the entire vertex space, and over the entire community space. But such an implementation, when tried, takes a significant amount of time to execute.
- Even though it is an O(n*log(n)) alg, the choice of data structure will have a significant effect on it.

Plots :

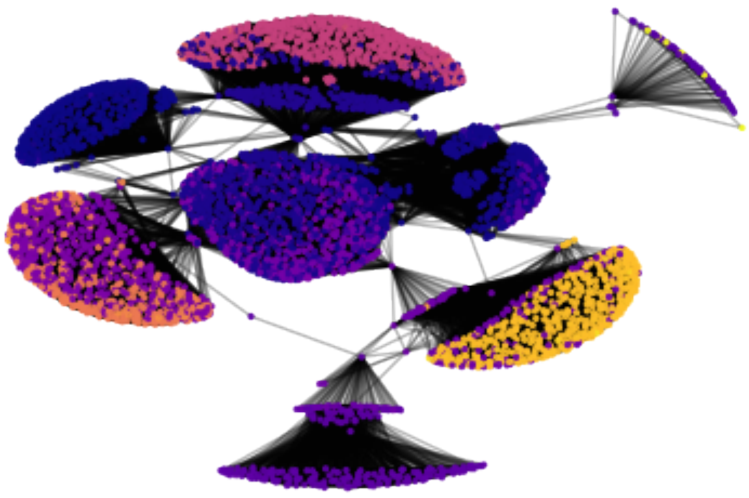

Fig: plot obtained for Louvain algorithm with the Facebook dataset.

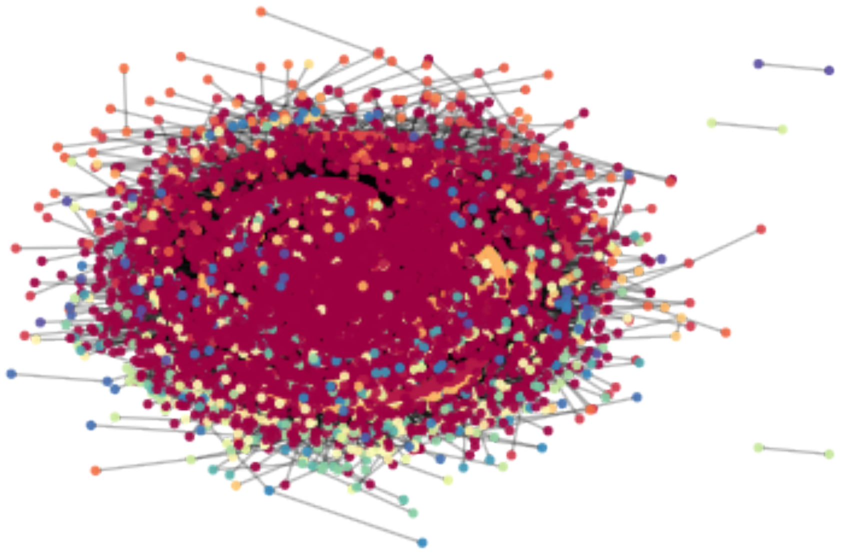

Fig: plot obtained for Louvain algorithm with the Bitcoin dataset.

Q5. How to pick the best?

Since the modularity metric returns the amount of structure in a network, in some sense, our objective should be to find clusters that can give high values of modularity gain. Note that It is a hard optimisation problem. The different algorithms try different algorithmic design paradigms. Eg. Louvain uses a greedy approach but notes that it doesn't guarantee that the founded structure is the global optima.

Q6.Time analysis.

- **Lauvin algorithm for both datasets takes 3 to 4 minutes** to execute, But the initial implementation took nearly 1 hour. Using efficient data structures and simplifying the expression for Q substantially decreased the time for execution.
- Note that, the previous implementation was feasible for small-size data, but as the size grows, it takes significant time.
- **Spectral decomposition is taking more time than expected.**
- **It can be increased or decreased by adjusting the parameter we set for selecting the 2nd minimum Eigen value and corresponding Eigenvector.**
- It takes 5 to 6 minutes if we put the threshold as 10e-8 while choosing 2nd min Eigenvalue.
- Such behaviour is expected because Eigenvalue/vector computation is an O(n^3) operation. Since we perform this step recursively, it takes O(2^n) time, this component comes because, in the worst case, all nodes can themselves be a community. Hence in total, O((2^n)*(n^3)) is the time complexity.

Q7. Which algorithm to choose?

I will go with the Louvain algorithm because of the following reasons.

1. less time complexity,O(n*log(n)),

Note:

- I feel that with the use of efficient data structures, the running time can be further reduced significantly.

2. Problem of recursion in spectral decomposition.

Note that recursion is a big eater of memory, each time we go one level in-depth, a new stack frame is needed, and all these are needed till we hit the base condition. So if we wish to perform these frequently in some embedded kind of devices, things can become problematic.

Take ways from the Community detection assignment:

1. While creating the dictionary for building up the environment, in Louvain alg, the initial implementation was behaving in a weird way. After debugging a lot, I can understand the way python handles the dependence between variables. Because of the mutual dependencies between variables, updating one variable afterwards sometimes update the primal variable. Using List and dictionaries within dependency can be problematic if we haven't used some techniques to isolate their connection.
2. Even though, in theory, Louvain alg takes big oh n*log(n), the implementation does matter. Initially, I used some data structures initially, taking significant time. After using dictionaries in a clever way, the time was significantly reduced.
3. Initially, the update was done only after iterating over the entire vertex(as per the slide).later to speed up, for each vertex, an update is performed.

Thank you