Community Detection in Networks

Name: Chaitanya Agrawal
Roll No.: 2016027

BTP report submitted in partial fulfillment of the requirements
for the Degree of B.Tech. in Computer Science & Engineering

on 15 November 2019

BTP Track: Research

BTP Advisors
Dr. Syamantak Das
Dr. Tanmoy Chakraborty

Indraprastha Institute of Information Technology
New Delhi

Student's Declaration

I hereby declare that the work presented in the report entitled **"Community Detection in Networks"** submitted by me for the partial fulfillment of the requirements for the degree of *Bachelor of Technology* in *Computer Science & Engineering* at Indraprastha Institute of Information Technology, Delhi, is an authentic record of my work carried out under guidance of **Dr. Syamantak Das and Dr. Tanmoy Chakraborty**. Due acknowledgements have been given in the report to all material used. This work has not been submitted anywhere else for the reward of any other degree.

Place & Date: Delhi, 5 June 2020
Chaitanya Agrawal

Certificate

This is to certify that the above statement made by the candidate is correct to the best of my knowledge.

Place & Date: Delhi, 5 June 2020
Dr. Syamantak Das and Dr. Tanmoy Chakraborty

Abstract

We are trying to devise a community detection algorithm for networks through modularity maximization. In this semester we further studied and understood the modularity function. We have explored semi-supervised active algorithms to devise an algorithm for this task.

Keywords: Community Detection, Modularity, Semi-Supervised Algorithms, Active Learning

Acknowledgments

I thank Dr. Syamantak and Dr. Tanmoy for letting me work on this very exciting idea and their tremendous guidance, support and wisdom.

Contents

Chapter 1

Introduction

Community Detection, or clustering, is a fundamental problem in a wide range of disciplines. The study of biological, social, citation networks has led to the observation of community structure in those networks. The nodes in these communities show important similarities and their classification into communities reveals many interesting properties.

Finding such communities is, however, a very hard task because ‘Communities’ is a very abstract concept [1]. The most accurate definition of a ‘Community’ can be a grouping of nodes such that the number of edges within the groups is more than the number of edges joining these groups, which is still very weak and abstract. Figure 1 shows such a network where the number of inter-community edges is much smaller than the number of intra-community edges.

Finding a clustering in a graph that minimizes the number of inter-community edges is an NP-complete problem [2]. Various attributes in networks make this problem incredibly hard. Communities can be of varying sizes with respect to both the number of edges and the number of nodes. The number of communities in a single network is not fixed. The ratios of inter-commu-

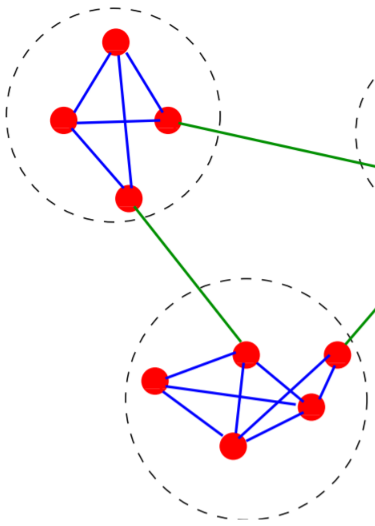

Figure 1: The nodes within circles form individual communities. The community structure seems very clear as the number of inter-community edges are minimal. [1]

nity and intra-community edges can vary widely. Nodes can even belong to more than 2 different communities, i.e. communities can be overlapping. Networks can have millions of nodes and edges which makes time-complexity a very important part of the problem.

We have been studying this problem for 3 semesters now. In the first semester, we studied the vast literature in community detection [1, 13, 14], identified the challenges of the problem and found how we can contribute. Our first task was to choose our definition of a community before devising an algorithm for community detection. The Modularity function [9, 13] is a very popular objective function used for the same. This function has been extensively studied and there have been a lot of algorithms [1, 10] that try find community structures in graphs using Modularity maximization.

In the second semester we sought to understand more about Modularity in particular. We observed that Modularity maximization is NP-Complete [11] and even an Inapproximable problem [10]. We sought to circumvent around this constraint by exploring Fixed Parameter Tractable Algorithms for Modularity.

In this semester, we tried to devise some Semi-Supervised Active Algorithms for Modularity Maximization. There has been promising work of this type when the given data is in a Euclidean space [19], or we are given a similarity relation among vertices which follows a hidden probability distribution depending on whether 2 vertices are in the same community or not [33]. There has been some work of this form for Community Detection in Networks [5–8]. However, none talk about Modularity Maximization or give any performance guarantees.

The rest of the report is organized as follows: In Chapter 2 we formally define Modularity and discuss the Modularity Maximization problem in detail. We list a few popular existing algorithms and list Hardness results for the problem. In Chapter 3, we explore Semi-Supervised Active Algorithms for Modularity Maximization. In Chapter 4, we discuss other attempts we have made for devising algorithms for Modularity Maximization. Finally, we conclude in Chapter 5 along with listing a few directions we can take this project into in the future.

Chapter 2

Modularity

Modularity is measured for a given undirected Graph G and a clustering C, where C is a disjoint partition of $V(G)$ and each $c \in C$ is considered as a separate community. It is based on the idea that a random graph is not expected to have a community structure and hence, the quality of a clustering C can be measured by comparing the density of edges among the subgraphs induced by the members of C in the actual graph to the density of edges among the subgraphs induced by the members of C in a random graph.

For Modularity, this random graph is computed such that the degree sequence remains intact. In this random graph, there is an edge between vertices v_i and v_j with probability $d(i)d(j)/(2m-1)^2$, where $d(i)$ is the degree of v_i. One can visualize this as follows. In order to form an edge between v_i and v_j, one needs to join 2 half-edges, or stubs. The probability to pick a stub incident with v_i is $d(i)/2m-1$. Thus, the probability that v_i and v_j are connected is $d(i)d(j)/(2m-1)^2$ and further the expected number of edges between v_i and v_j are $2m * d(i)d(j)/(2m-1)^2$. For ease of computation, this is approximated to $d(i)d(j)/2m$.

Thus, Modularity (Q) can be computed as follows

$$Q(G,C) = \frac{1}{2m}\sum_{ij}\left[A_{ij} - \frac{d(i)d(j)}{2m}\right]\delta^{C}_{i,j}, \tag{2.1}$$

where, A_{ij} is the adjacency matrix of G, $m = |E(G)|$ and $\delta^{C}_{i,j}$ is the Kronecker delta function which yields 1 when v_i, v_j belong to the same cluster in C and 0 otherwise.

The delta function ensures that the only contributions to the sum come from vertex pairs that belong to the same cluster. Hence, we can rewrite Modularity as follows

$$Q(G,C) = \sum_{c \in C}\left[\frac{l_c}{m} - \left(\frac{d_c}{2m}\right)^2\right], \tag{2.2}$$

where, $l_c = |E(G[c])|$ and d_c is the sum of degrees of the vertices in c.

The modularity of the entire graph taken as a single community is zero, as the 2 terms in equation 2.2 are 1 and -1 and hence, cancel each other. Modularity is always smaller than 1 and can be negative as well. For instance, let $C = \{\{v_1\}, \{v_2\}, ..., \{v_n\}\}$, where in this case, the first term is zero while the second term is negative. So, $Q(G, C) \in [-1, 1]$. However, we can say that $\max_C Q(G, C) \geq 0$ as we can always take the entire graph as a single community to get Modularity equal to zero.

Modularity is even defined similarly graphs having self-loops and multiple/weighted edges.

Other properties of Modularity

Theorem 1 [27]: For any integer $k > 0$ and any graph G,

$$\max_{C:|C| \leq k} Q(G, C) > \max_C Q(G, C)(1 - 1/k).$$

This tells that when we find the maximum Modularity while binding the number of clusters by a constant k, it is a $(1 - 1/k)$ factor approximation of maximum Modularity where the number of clusters is unbounded.

Theorem 2 [11]: Suppose that G is connected graph with no isolated vertices. If C' is a clustering of G, such that $C' = \operatorname{argmax}_C Q(G, C)$, for every $c \in C'$, $G[c]$ is a connected graph.

2.1 Maximizing Modularity

It can be seen from equation 2.2, that a higher number of internal edges in a cluster increases the value of Modularity. So, large values of Modularity generally indicate good partitions (It is not always the case that large values indicate good partitions. This is because of a phenomenon called Resolution Limit [21]. But we have not gone into detail about this.). It is by far the most used and the most popular objective function used for community detection as it was one of the first attempts to define and quantify the quality of a graph partitioning [1].

2.1.1 Popular Existing Algorithms

As mentioned above there have been a lot of algorithms that try to find Community structures in graphs by optimizing Modularity. As we will discuss later in section 3.2.1, maximizing Modularity is NP-Complete [11]. However, there are several algorithms that are able to achieve fairly good approximations. We summarize a few below categorized by the underlying strategy used by the algorithm.

Greedy Techniques

The greedy method of Newmann [22] was the first algorithm devised to maximize Modularity. It is an agglomerative hierarchical clustering method where groups of vertices at each iteration are joined to form larger communities such that Modularity increases after merging. One starts with n clusters, each containing a single vertex. The edges of the graph are not added yet. We always choose an edge that is outside the already formed communities, and after adding this edge, we merge the 2 communities it connects. If there are edges that haven't been added within this new community, we add them. This is repeated till there are no edges left to add. The number of clusters goes down by one after every iteration and hence there are n-1 iterations in total. After the termination of the algorithm, there are n-1 clusterings formed, one at the end of each iteration, and the one with the maximum Modularity is chosen. This algorithm runs in $O((n+m)n)$ or $O(n^2)$ time. There have since been a lot of improvements proposed to this algorithm that either improve complexity or the maximum Modularity as returned by the algorithm [1]. Despite improvements and refinements, the accuracy of this algorithm is still not that good as compared with other techniques.

Another type of greedy method was proposed in [30] and is popularly known as Louvain. It consists of 2 phases which are repeated. Same as the previous algorithm, initially, every node is a community only consisting of themselves. Every edge in the graph is assigned a weight of 1. In the first phase, every node, in a certain order, is considered for merging with its neighbouring community. The merge happens with the community that yields the largest positive change in Modularity. This procedure repeats iteratively and stops when there is no possible increase in Modularity. In the second phase, a community network is built based on the communities discovered in the first phase where the communities built in the previous phase are the nodes of the new network and there is an edge between 2 nodes if there are edges between the corresponding communities. This new edge is assigned a weight equal to the total weight of the edges between the corresponding communities. These 2 phases keep repeating iteratively until there is no positive change in modularity. This algorithm performs much better and faster than the Newmann algorithm. However, the results of the algorithm are impacted by the order in which the nodes are considered in the first phase for merging, which impacts its utility [31].

Simulated Annealing

Simulated Annealing [23] is a probabilistic procedure used for optimization of any function F. It performs an exploration of the space of possible states. Transitions from 1 state to other occur with probability 1 if F increases, otherwise with probability $\exp(\beta \Delta F)$, where ΔF is the decrease in F and β is an index of stochastic noise which increases after every iteration.

It was first employed for Modularity maximization in [24]. Transitions are either local or global. Local transitions involving the shifting of single vertices from one cluster to another, and Global transitions involving the merging and splitting of clusters. This method works particularly well for cases where we only have to form 2 clusters and one only considers local moves in that case. It can potentially come very close to the true Modularity maximum, but it is very slow and can only be used for small graphs of upto 10^4 vertices.

2.1.2 Computational Hardness

None of the algorithms shown above have any sort of theoretical guarantees regarding the maximum Modularity they achieve. Besides there is a huge tradeoff between accuracy and complexity with the more accurate algorithms being considerably more inefficient. Maximizing Modularity is very difficult in general and has been shown to be NP-Complete and even Inapproximable.

NP-Completeness

[11] proved that the decision version of Modularity maximization,

MODULARITY: *Given a Graph G and a number $K \in \mathbf{R}$, is there a clustering C of G, for which $Q(G, C) \geq K$?*

is strongly NP-Complete. They show this by reducing the 3-PARTITION problem, which is strongly NP-Complete [25], to the MODULARITY problem. They also show that even if one sets a bound on the number of clusters, the decision version of this modified problem,

k-MODULARITY: *Given a Graph G and a number $K \in \mathbf{R}$, is there a clustering C of G into $\geq k$ clusters, for which $Q(G, C) \geq K$?*

is NP-Complete. They show this by reducing the MINIMUM BISECTION FOR CUBIC GRAPHS problem, which is strongly NP-Complete [26], to the $k-$MODULARITY problem.

Inapproximability

Modularity has been shown to be APX-Hard [27] as well. [10] have also proved that there doesn't exist an approximation algorithm with any factor $\rho > 0$ for the Modularity clustering. However, [10] have also proposed the first additive approximation algorithm that finds community structure with modularity atleast $\max_C Q(G, C) - 2(1 - \kappa)$ for $\kappa = 0.766$. It is based on rounding a semidefinite program, similar to that in [28] for the MaxAgree problem.

Modularity for Trees

[32] prove that Modularity for Trees with arbitrary edge weights is NP-Complete by reducing it to the Subset-Sum problem. They also give a polynomial time algorithm that can solve Modularity for uniformly weighted trees in polynomial time. They show this by reducing Modularity for trees to the k-MSSV problem defined as follows:

k-MSSV: Given a tree T, find the find a set of k edges such that their removal minimizes the sum-of-squares of component volumes in the residual forest.

They further give a polynomial time dynamic programming based algorithm for k-MSSV that runs in $O(n^5)$ time. They use this algorithm to give a pseudo-polynomial time algorithm for Modularity on trees having arbitrary weights. This algorithm has complexity $O(n^5W^4)$ where W is the maximum edge weight in the tree. They further give a PTAS for Modularity for Trees with arbitrary edge weights which gives an $(1 - \epsilon)$-approximation in $O(n^{(1+1/\epsilon)})$ time.

Chapter 3

Semi-Supervised Active Algorithms

We have seen how it is very hard to maximize Modularity. In general, physicists, statisticians and computer scientists have been trying to develop algorithms for community detection for a long time. However, perfectly recovering each element's community by only observing the given graphical data is shown to be statistically impossible for many networks of interest [3, 4]. We feel that using some degree of supervision can help a lot in this regard.

Supervised information is not freely available and can often be very expensive to obtain. Apart from that, most information might not even be useful every time. Querying information, that is more important for the purpose of the algorithm, is much more useful in this scenario. This is called active learning. The primary objective of active learning is to minimize the number of queries used to achieve the underlying objective, which in this case is community detection.

It is not easy to use this information in clustering algorithms either. Since, the number of communities is not fixed in most networks, labelled information can only tell whether 2 nodes can belong to the same community or not. Most state of the art community detection algorithms use matrix forms which require a fixed number of communities as a parameter whereas the labelled information doesn't assume a fixed number of communities. Thus, labelled information is incompatible to use with most state of the art community detection algorithms. A lot of semi-supervised algorithms have been proposed in recent years that address the same [15–18]. In most cases, researchers have proposed completely different algorithms altogether, just to use the labelled information.

Thus, we explored using a controlled amount of supervision to obtain a better community structure. This class of algorithms falls under semi-supervised active (SSA) community detection. This is a very recent and growing area of research [5–8]. The primary challenge here is to find an effective way to minimize the number of queries used, or the query complexity. Hence, we have to (1), find a way to make queries such that they provide us the maximum information and (2), find an effective way to use this information and create a semi-supervised community detection algorithm for it. We intend to prove bounds on the number of queries required to get maximum Modularity values, or an α-approximation of maximum Modularity for some $\alpha > 0$.

Similar results have been proven in 2 recent works [12,19,33].

3.1 Query Model

One can make many different types of queries as well. The most popular methods of making queries in literature are:

- *Assignment Query*: Given a vertex, we make a query asking which community some vertex i belongs to.
- *Same-Cluster Query*: Given 2 vertices, we make a query asking whether these 2 vertices belong to the same community or not.

For our work, we focused on the Same-Cluster Query model as we felt that it would be easier for an oracle to answer queries of this form and hence, is more practical. The relationship between any 2 nodes is easier to identify. One should note that a single assignment query is essentially equivalent to making same-cluster queries to all the existing clusters. Thus, every single query gives us lesser information .

Upper Bound on Query Compexity

For a given number of vertices n and number of communities k, one can make $\Theta(nk)$ same-cluster queries to obtain the optimal community labelling. For every v, same-cluster queries to a single vertex of all the other existing clusters to obtain the optimal community labelling.

3.2 Related Work

The following works have tackled the problem of community detection using same-cluster queries along with proving bounds on the query complexity of their algorithms. However, these works are not for the problem we consider. Instead of network data, they work on some other form of data. To the best of our knowledge, no such work has been done for community detection in networks yet.

[12] assumes that the given data has no underlying attribute, structure or inter-relatedness and the answers to the queries they make are wrong with some probability p. They say that queries can be made in 2 ways: (1) Adaptively: ask a few queries, compute the results, ask

new queries; (2) Non-adaptively: ask all the queries at once in the beginning. The authors have given algorithms for both these settings and have proved query complexity and time complexity bounds with lower bounds on accuracy. Their results indicate that one can establish a trade-off between the query complexity, the time complexity and the accuracy of clustering.

[33] provides a significant result in this area for a general form of community detection. They assume that the given data follows a similarity relation that is encoded in a matrix. For example, let this matrix be M, then $M_{ij} = M_{ji}$ gives the similarity relation between i, j. Further, it is assumed that if 2 vertices belong to the same cluster, their corresponding similarity value is drawn from a probability distribution f_+. Otherwise, their corresponding similarity value is drawn from a probability distribution f_-. They give an algorithm that makes $O(\frac{k^2 \log n}{\mathcal{H}^2(f_+|f_-)})$ queries and gives the optimal clustering with probability 1-$o(1/n)$.

Algorithms by both [12, 33] follow a similar framework. Some p vertices are randomly sampled and, pk queries are made among them. This gives a few candidate clusters. Further, these clusters are grown by making a few queries. Finally, these clusters are finalized by making use of some side information available in the problem instance.

[19] also provide such an algorithm for the k-means clustering problem. They have introduced a novel notion of 'clusterability' called the γ-margin property. This is defined for a computed clustering of the algorithm which tries to say how well separated data points in different clusters are, a quality function, much like Modularity. A good clustering of the data is a clustering with a higher γ-margin. The authors have proposed an algorithm that has time and query complexities expressible in terms of γ and δ, the probability of recovering the clustering. They have further proved that for a low enough γ-margin, k-means is NP Hard if it doesn't make at least x queries. They prove a lower bound on x as well.

3.3 Our Attempt at an Algorithm

We tried to formulate an algorithm with a very similar design to [33]. It works in three phases: (1) Identifying Candidate Clusters, (2) Growing the Clusters, (3) Finalizing the Clusters.

Phase 1: Identifying Candidate Clusters

We first sample x vertices uniformly at random and make queries for each of the sampled vertices, such that we know the community relation between all pairs of the sampled vertices. As discussed above, this can be done using $x.k$ queries, where k is the number of communities.

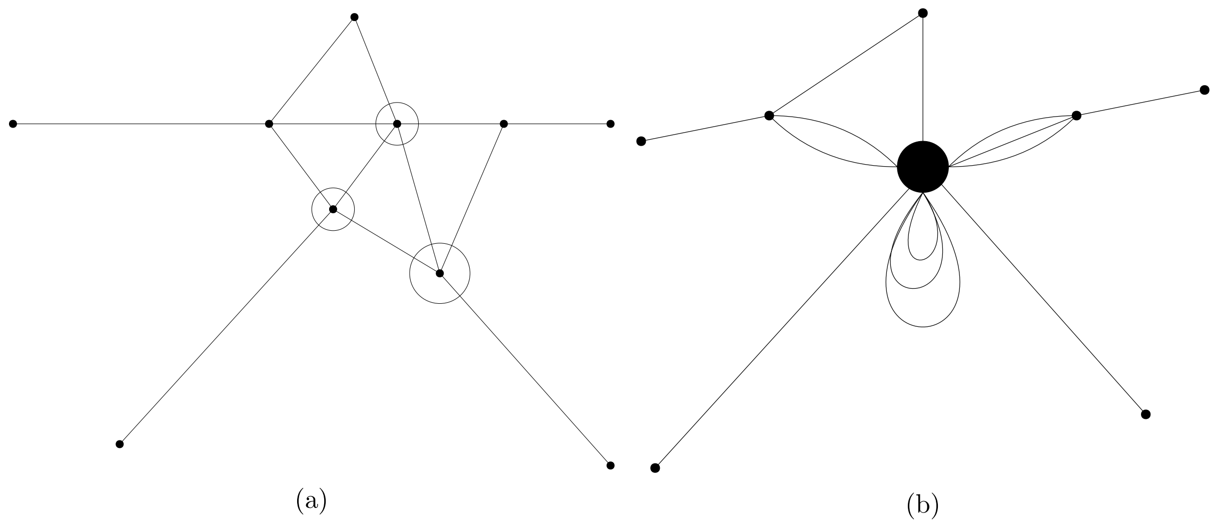

Figure 2: Consider this example graph in (a), where, the circled points are sampled in Phase 1 and all of them are found to be in the same community. The contraction procedure converts (a) to the graph shown in (b), where, the circle in black is obtained after contraction of the marked points in (a).

Finally, for each new community, we contract all sampled vertices in that community into a single 'super-vertex'.

We define this operation as follows: Consider the community c where X_c contains the set of the sampled vertices that belong to community c. We combine the vertices in X_c to form the super vertex C_c. For every edge (v_p, v_q), where $v_p, v_q \in X_c$, we add a self loop in C_c. For every edge (v_p, v_q), where one of v_p, v_q are not in X_c, we add an edge from C_c to v_q. Thus C_c has self loops and multiple edges with other vertices. See Figure 1 for more clarity about this procedure.

If we have to make a same-cluster query from some other vertex to C_c, we make a same-cluster query to any random member of C_c. From this point, we refer to C_c as the center of community c.

[34] proved that this operation preserves the Modularity of a graph and hence, doing this doesn't affect the final solution.

Phase 2: Growing the Clusters

We then grow the clusters with the idea of finding a distance q such that all vertices beyond this distance q from the center definitely won't lie in the same community as the center. Thus, we are sort of finding a 'boundary' for every cluster. We find this boundary by making queries for vertices with the center. We do this procedure by a modified BFS.

Suppose we are at some level p of the BFS, we iteratively make same-cluster queries for all vertices on this level with the center until we encounter such a vertex which is in the same cluster as the center. The order in which the vertices are queried may be arbitrary. This tells us that the boundary of this cluster is more than p units of distance away from the center. If we find such a level q where all the vertices are reported to be in separate clusters than the center, we stop and return $q-1$ as the distance of the boundary of community c from the center C_c.

Further, for every vertex that is in a different community than the center, we don't add its neighbours to the queue for the BFS. This can help us in optimizing the query complexity of the algorithm.

We also prove the following lemma to have some traction on the number of queries we make in this phase of the algorithm.

Lemma 1: $\mathbb{E}[q] = n_c/x_c$ for each cluster c, where $q+1$ is the distance of the boundary of community c from C_c, n_c are the number of points that would lie in c in the optimal community assignment and x_c is the number of points that are sampled for community c in Phase 1.

Proof. Let all vertices that would in c in the optimal community assignment be V_c. Now, we know that the subgraph formed by V_c is connected (Theorem 2). Also, for any vertex $v \in V_c$, we know that at least q points lie within a distance $q-1$ from v (Including v). Call this set D_v. Since, we are sampling vertices uniformly at random, $\mathbb{P}[\text{One of the vertices in } D_v \text{ is sampled in Phase 1}] = x_c * q_c/n_c$. Now, since $q+1$ is the distance of the boundary from C_c, all points in V_c must lie within distance q from C_c. Thus, $\mathbb{P}[\text{One of the vertices in } D_v \text{ is sampled in Phase 1}] = 1 = x_c * q_c/n_c$, and $\mathbb{E}[q] = n_c/x_c$. □

Thus, sampling only $O(\sqrt{n})$ points in Phase 1, would bound the level of this BFS to $O(\sqrt{n})$. Further, we believe that under some conditions, we can improve this bound. However, we were not able to prove a bound on the number of queries one would need to make at each level.

Phase 3: Finalizing the Clusters

Once we have these cluster boundaries for all clusters, we have significantly reduced the size of the problem as we can eliminate the possible communities for a lot of vertices. However, we couldn't find how to execute this final step such that we have some performance guarantee regarding the final community assignment.

Nevertheless, we feel that the first two phases can serve as very good initializers for any Modularity Maximization heuristic algorithm that can be conducted in this phase.

Chapter 4

Other Attempts

We also tried a few other things for the Modularity Maximization problem in general. In this semester, we explored the relationship of Modularity Maximization to Correlation Clustering and, Modularity Maximization in Planar Graphs by leveraging the Planar Separator Theorem. We explored Fixed-Parameter Tractable Algorithms for Modularity in the Monsoon 2019 semester.

4.1 Modularity Maximization and Correlation Clustering

Correlation Clustering is an alternative formulation of the Community Detection problem. Here, we are given a weighted graph, where all weights $\in \{-1, 1\}$. The weight of an edge between vertices v_i, v_j denote the 'correlatedness' of vertices v_i and v_j. The correlation clustering problem may be formulated in 2 ways, (1) Maximizing Agreement (sum of edge weights within the clusters) between the vertices (MAX-AGREE) or, (2) Minimizing Disagreement (sum of edge weights outside the clusters) between the vertices (MIN-DISAGREE).

It has been proven that the Modularity Maximization problem can be reduced to MAX-AGREE [10]. We thus tried to explore a Semi-Supervised Active algorithm for MAX-AGREE. There already exist PTAS's for MAX-AGREE when the weights are either +1, or -1 [35,36], but not for fractional weights.

We explored Correlation Clustering as we felt that we could find some property of the Correlation Clustering problem in general and tried to formulate a Semi-Supervised Active Algorithm for this problem instead.

4.2 Modularity in Planar Graphs

We explored MODULARITY for Planar Graphs. We tried to leverage the Planar Separator Theorem [37] for the same. The Planar Separator Theorem states that any planar graph can be split into smaller connected subgraphs by removing $O(\sqrt{n})$ vertices, where the smaller subgraphs have at most $2n/3$ vertices.

We felt that we can find a recursive splitting algorithm for Modularity Maximization. In particular, if we would have been able to prove that some cluster boundary would entirely lie in the subgraph formed by the removed vertices, we could have further used queries to find the exact boundaries of the cluster and obtain a good clustering.

4.3 Fixed-Parameter Tractable Algorithms

FPT for short, this is a class of algorithms which have complexities of the form $O(f(k) * polynomial(x_1, x_2, ..., x_n))$, where $x_1, x_2, ..., x_n$ denote the input size and k is some parameter associated with the input. For example, in our case, n, the number of vertices in the graph, and m, the number of edges in the graph are the input sizes and other properties of the graph like the minimum cut size or the minimum vertex cover of the graph can be treated as the parameters of the graph. Here, $f(k)$ can be of any order like $O(2^k)$ or even $O(2^{2^k})$. Generally the argument for these algorithms is that for graphs with small k, the complexity of the algorithm is essentially polynomial.

4.3.1 Fixed-Parameter Tractability of Modularity

[20] claims that there exists an FPT algorithm for Modularity when parameterized by the size of the minimum vertex cover, and prove that it is solvable in polynomial time whenever the treewidth or max leaf number of the Graph is bounded by some fixed constant (an XP algorithm). They also obtain an FPT algorithm parameterized by the treewidth of the graph, to compute maximum c-Modularity. By Theorem 1 we can say that this algorithm is also an FPT algorithm for any constant value approximation of maximum Modularity.

We were able to find a mistake in their first algorithm which is parameterized by the size of the minimum vertex cover in the graph. This is basically saying that there is no proof that MODULARITY is in FPT. Their algorithm which is parameterized by the treewidth of the graph uses dynamic programming techniques on tree decompositions. With a few tweaks to this algorithm, they also obtain an FPT algorithm parameterized by the treewidth of the graph for solving c-MODULARITY.

Our attempt at an Algorithm

We however felt that parameterizing using treewidth or minimum vertex cover is not very practical. Thus, we sought to find an FPT algorithm for Modularity when parameterized by the size of the cut induced by the optimal Modularity clustering. Knowing that for most small graphs, Modularity cuts won't be too large, being able to provide accurate Modularity clustering for these graphs can be very useful for biological networks. We devised the following algorithm for the 2-Modularity. However, this algorithm is not in FPT because of an issue which we discuss later.

Our algorithm uses dynamic programming techniques. It returns the optimal Modularity cut only if the size of the optimal Modularity cut, C^*, is $\leq k$. We first find a suitable cut of the graph G, C. Now this cut C may not be equal to C^*. All edges of C^* may lie in $G[X]$ or $G[Y]$ or in C itself. Now, we make guesses regarding where this cut may lie and simply select the guess with the best Modularity. We tried to devise our algorithm in a way such that this value of Modularity was the maximum Modularity.

Now we take the sets X' and Y' which are the sets of vertices in which endpoints of C lie in X and Y respectively. Lets focus only on X' for now, we repeat the same procedure for Y'. We then find all possible bipartitions of X' and for all these $\leq 2^k$ bipartitions, we again find cuts that separate the vertices in a single bipartition. From here on, we simply recurse till the end. Suppose at any level of recursion, we get x candidate cuts from X and y candidate cuts from Y. Of these, we select the cuts from the respective sets such that their combination would maximize Modularity in $G[X \cup Y]$.

By fixing the size of the maximum Modularity cut, we had hoped that this recursion tree would have maximum depth k or $\log n$. However, we found that this may not be the case as our selection of cut may partition G into something like $(\{v_j\}, G \setminus \{v_j\})$ and may keep doing so till the end. This would force the size of the resulting recursion tree to be 2^n which is not good. This problem was particularly prominent when selecting min-cuts as our suitable cuts. We also tried experimenting with balanced cuts, however, the problem was not solved using them either as it may be the case that after, say, the first recursion level, the remaining graph G''s optimal Modularity cut may be of the form $(\{v_j\}, G' \setminus \{v_j\})$. Using balanced cuts may again result in a recursion tree of size 2^n.

Nevertheless, we tried implementing the algorithm by using min-cuts and as we feared, the algorithm took very long to run even for graphs with 100-200 nodes.

Important cuts Apart from this, we tried another slightly different direction and explored the relation of maximum 2-Modularity cuts with Important cuts. Important cuts are defined below.

Let C be an XY-cut of some Graph G for some $X, Y \in V(G)$ and let R be the set of vertices that are reachable from X in $G \backslash C$. Then we say that C is an Important cut if it is inclusion-wise minimal and there is no XY-cut C' such that $|C'| \leq |C|$ and $R \subset R'$ where R' is the set of vertices reachable from X in $G \setminus C'$.

There is an FPT algorithm [29] which returns all the Important cuts in a graph. We chose Important cuts as we felt that they were similar in nature to maximum Modularity cuts. Important cuts try to reduce the size of the cut while making sure that one of the partitions remain connected at all times. This is very similar to what happens in a maximum Modularity cut. As Theorem 2 says, the partitions in a maximum Modularity cut are always connected, and Modularity is defined such that increasing Modularity will reduce the number of intra-community edges, or the cut size in the 2-Modularity Case. So, we conjectured that a maximum 2-Modularity cut could be an Important cut.

However, we were able to find a counter example for the same, where the maximum 2-Modularity Cut of a graph is not an Important Cut.

Theorem 3: The Maximum 2-Modularity cut in a Graph is not necessarily an Important cut.

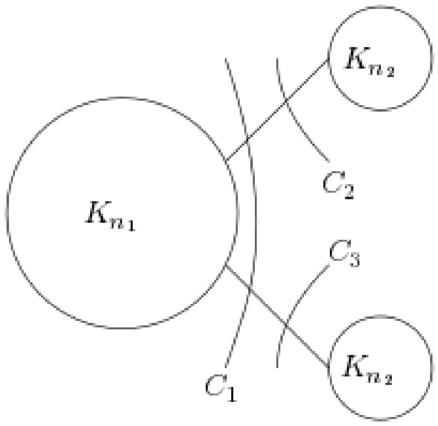

Figure 3

Consider the graph G in Figure 3. K_n signifies a clique. So, this graph has 3 maximal subgraphs which are cliques and the bigger clique is connected to each of the smaller cliques through a single edge. Let us say that these 2 single edges are incident on different vertices in the bigger clique. Here, assume $n_1 >> n_2 >> 1$. In G, C_2 and C_3 are the only important cuts. We prove that these are not the maximum 2-Modularity cuts by showing that C_1 induces a higher Modularity.

$$
\begin{aligned}
Q(G, C_1) &= 1 - \frac{2}{m} - \left(\frac{(n_1(n_1-1)+2)^2 + (2(n_2(n_2-1)+1))^2}{4m^2}\right) \\
Q(G, C_2) &= 1 - \frac{1}{m} - \left(\frac{(n_1(n_1-1)+2+n_2(n_2-1)+1)^2 + (n_2(n_2-1)+1)^2}{4m^2}\right)
\end{aligned}
$$

Now, $Q(G, C_2) = Q(G, C_3)$. Thus, we have to show that $Q(G, C_1) > Q(G, C_2)$ or $Q(G, C_1) - Q(G, C_2) > 0$.

$$
Q(G, C_1) - Q(G, C_2) = \left(\frac{2(n_1(n_1-1)+2)(n_2(n_2-1)+1) - 4(n_2(n_2-1)+1)^2}{4m^2}\right) - \frac{1}{m}
$$

Now we have to show:

$$
\begin{aligned}
&\frac{2(n_1(n_1-1)+2)(n_2(n_2-1)+1) - 4(n_2(n_2-1)+1)^2}{4m^2} > \frac{1}{m} \\
\implies &2(n_1(n_1-1)+2)(n_2(n_2-1)+1) - 4(n_2(n_2-1)+1)^2 > 4m
\end{aligned}
$$

putting $m = n_1(n_1-1)/2 + n_2(n_2-1) + 2$

$$
\begin{aligned}
&(n_1(n_1-1)+2)(n_2(n_2-1)+1) - 2(n_2(n_2-1)+1)^2 > n_1(n_1-1) + 2n_2(n_2-1) + 4 \\
\implies &(n_2(n_2-1)+1)((n_1(n_1-1)+2) - 2(n_2(n_2-1)+1)) > n_1(n_1-1) + 2n_2(n_2-1) + 4 \\
\implies &(n_2(n_2-1)+1)(n_1(n_1-1) - 2n_2(n_2-1)) > n_1(n_1-1) + 2n_2(n_2-1) + 4 \\
\implies &n_1(n_1-1)n_2(n_2-1) > 2n_2(n_2-1)(n_2(n_2-1)+2) + 4
\end{aligned}
$$

Since $n_1 >> n_2$ it is pretty easy to see that the above is true. We have shown that C_1 induces a higher Modularity than both C_2 and C_3, which are the only Important cuts in G. Therefore, it cannot be the case that the maximum 2-Modularity cut in G is an Important cut. □

Chapter 5

Conclusion and Future Plan

In the first semester (Winter 2019), we seeked to study about the field of Community Detection in graphs. We surveyed the work that had been done in the past few years and tried to identify how we can contribute. From there on, we arrived at Modularity and thought about pursuing SSA and FPT Algorithms.

In the second semester (Monsoon 2019), we sought to understand Modularity. We tried to understand the challenges behind maximizing it, how it can be used for community detection and how we should pursue this function. In particular, we explored the FPT Framework for constructing an algorithm for Modularity Maximization.

In the third semester (Winter 2020), we explored Modularity even more. We sought to understand its relationships with Correlation Clustering and other known problems. In particular, we explored the SSA framework and gave a blueprint of an algorithm that may be developed further.

In the future,

- we will further develop the algorithm proposed in section 4.3.
- we will explore Representation Learning for Modularity Maximization. Representation Learning [38] can be used to encode a graph into a Euclidean space. Once, we have such an encoding, we can use the results of k-means for this problem [19].
- we will explore the notion of Perturbation Resilience [39] for Modularity. This is a very recent framework for combinatorial optimization problems. It is based on the idea that more 'practical' instances of combinatorial optimization problems generally obey some rules and assuming such rules, one can formulate algorithms that may have a much higher approximation constant, or an exact algorithm.

Chapter 6

Work Timeline

Objective	Dates
Community Detection Survey	1 January 2019 - 31 January 2019
Bias in Community Detection Problem Formulation	1 February 2019 - 28 February 2019
Semi-Supervised Active Community Detection Survey along with Problem Formulation	1 March 2019 - 15 April 2019
Modularity Survey and Hardness Results	1 August 2019 - 15 September 2019
Fixed Parameter Tractable Algorithms for Modularity	15 September 2019 - 15 November 2019
Modularity and Correlation Clustering	1 January 2020 - 15 February 2020
Modularity in Planar Graphs	15 February 2020 - 31 March 2020
Semi-Supervised Active Algorithms for Modularity	1 April 2020 - 31 May 2020

Bibliography

[1] Community Detection in Graphs. S. Fortunato, 2010

[2] Computers and Intractability; A Guide to the Theory of NP-Completeness. M. R. Garey and D. S. Johnason, 1990

[3] Exact Recovery in the Stochastic Block Model. E. Abbe, A. S. Bandeira and G. Hall, 2014

[4] A proof of the block model threshold conjecture. E. Mossel, J. Neeman and A. Sly, 2014

[5] Active Community Detection: A Maximum Likelihood Approach. B. Mirabelli and D. Kushnir, 2018

[6] Active Semi-Supervised Community Detection Based on Must-Link and Cannot-Link Constraints. J. Cheng, M. Leng, L. Li, H. Zhou and X. Chen, 2014

[7] Active Semi-supervised Community Detection Algorithm with Label Propagation. M. Leng, Y. Yao, J. Cheng, W. Lv and X. Chen, 2013

[8] Active link selection for efficient semi-supervised community detection. L. Yang, D. Jin, X. Wang and X. Cao, 2015

[9] Finding and evaluating community structure in networks. M. Newman and M. Girvan, 2004

[10] Network Clustering via Maximizing Modularity: Approximation Algorithms and Theoretical Limits. T. N. Dinh, X. Li and M. T. Thai, 2016

[11] On modularity clustering. U. Brandes, D. Delling, M. Gaertler, R. Gorke, M. Hoefer, Z. Nikoloski and D. Wagner, 2008

[12] Clustering with Noisy Queries. A. Mazumdar and B. Saha, 2017

[13] Metrics for Community Analysis: A Survey. T. Chakraborty, A. Dalmia, A. Mukherjee and N. Ganguly, 2016

[14] Community Detection in Networks: A User Guide. S. Fortunato, 2016

[15] Semi-Supervised Community Detection Based on Distance Dynamics. L. Fan, S. Xu, D. Liu and Y. Ru, 2018

[16] Semi-supervised community detection based on discrete potential theory. D. Liua, X. Liua, W. Wang and H. Bai, 2014

[17] Phase transitions in semi-supervised clustering of sparse networks. P. Zhang, C. Moore1 and L. Zdeborov, 2014

[18] Enhanced Community Structure Detection in Complex Networks with Partial Background Information. Z. Zhang, K. Sun and S. Wang, 2013

[19] Clustering with Same-Cluster Queries. H. Ashtiani, S. Kushagra and S. Ben-David, 2016

[20] The Parameterised Complexity of Computing the Maximum Modularity of a Graph. K. Meeks and F. Skeman, 2018

[21] Resolution Limit in Community Detection. S. Fortunato and M. Barthelemy, 2007

[22] Fast algorithm for detecting community structure in networks. M. Newmann, 2004

[23] Optimization by Simulated Annealing, S. Kirkpatrick, C. D. Gelatt and M. P. Vecchi, 1983

[24] Modularity from fluctuations in random graphs and complex networks. R. Guimerà, M. Sales-Pardo, and L. Amaral, 2004

[25] Complexity results for multiprocessor scheduling under resource constraints. M. R. Garey and D. S. Johnson, 1975

[26] Graph bisection algorithms with good average case behavior. T. Bui, S. Chaudhuri, F. Leighton and M. Sipser, 1987

[27] On the complexity of Newmann's community finding approach for biological and social networks. B. Dasgupta and D. Desai, 2013

[28] Clustering with qualitative information. M. Charikar, V. Guruswami and A. Wirth, 2005

[29] Parameterized Algorithms. M. Cygan, F. V. Fomin, L. Kowalik, D. Lokshtanov, D. Marx, M. Pilipczuk, M. Pilipczuk and S. Saurabh, 2016

[30] Fast unfolding of communities in large networks. V. Blondel, J. Guillame, R. Lambiotte and E. Lefebvre, 2008

[31] Constant communities in large networks. T. Chakraborty, S. Srinivasan, N. Ganguly, S. Bhowmick and A. Mukherjee, 2013

[32] Towards Optimal Community Detection: From Trees to General Weighted Networks. T. N. Dinh and M. T. Thai, 2012

[33] Query Complexity of Clustering with Side Informatoin, A. Mazumdar and B. Saha, 2017

[34] Size reduction of complex networks preserving modularity. A Arenas, J Duch, A Fernandez and S Gomez, 2007

[35] Correlation Clustering with Same-Cluster Queries Bounded by Optimal Cost. B. Saha and S. Subramaniam, 2019

[36] Approximate Correlation Clustering Using Same-Cluster Queries. N. Ailon, A. Bhattacharya and R. Jaiswal

[37] A separator theorem for planar graphs. R. J. Lipton and R. E. Tarjan, 1979

[38] Representation Learning on Graphs: Methods and Applications. W.l. Hamilton, R. Ying and J. Leskovec, 2017

[39] Algorithms for Instance-Stable and Perturbation-Resilient Problems. H. Angelidakis, K. Makarychev, Y. Makarychev and A. Vijayaraghavan. 2017