COMMUNITY-AFFINITY: MEASURING STRENGTH OF MEMBERSHIPS OF NODES IN NETWORK COMMUNITIES

by

Nitin Yadav

A thesis submitted to the faculty of
The University of Utah
in partial fulfillment of the requirements for the degree of

Master of Science

in

Computing

School of Computing

The University of Utah

December 2015

The University of Utah Graduate School

STATEMENT OF THESIS APPROVAL

The thesis of **Nitin Yadav**

has been approved by the following supervisory committee members:

Suresh Venkatasubramanian	, Chair	**07/28/2015**
		Date Approved
Jeffrey Phillips	, Member	**07/28/2015**
		Date Approved
Vivek Srikumar	, Member	**07/28/2015**
		Date Approved

and by **Ross Whitaker** , Chair of

the **School of Computing** and by

David B. Kieda , Dean of The Graduate School.

ABSTRACT

Detecting community structure in networks has been a widely studied area. While most of the methods produce an exclusive membership of the nodes, the nodes in real-world networks tend to partially belong to more than one community. In this thesis, we study some methods that have been used to quantify the strength of memberships of nodes in different communities (or *community-affinity*, as we call it) and also define three of our own methods. Our first method is based on personalized PageRanks of the nodes, the second is based on the individual contribution of nodes to the modularity of the graph, and the last is based on the common neighborhood between two nodes. We first discuss different notions of community-affinity, each of which is followed by formulations that capture that notion. We then discuss the concept of *stability*, which uses community-affinity scores of the nodes to compute how "stable" each node is in a given community structure and how we can use this information in estimating the quality of a given community structure. Towards the end, we introduce a community detection algorithm, which "peels" communities one by one from a graph. The results of our experiments show that our algorithm is very accurate even for a large number of nodes in a graph. Our algorithm is fast and it performs very well on real-world graphs compared to the state of the art algorithms.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

I would like to express my sincere gratitude to my advisor Prof. Suresh Venkatasubramanian for his continuous support during my master's research work. I am thankful to him for his patience, the motivation that he provided (especially when I needed it most), and for sharing his immense knowledge. He has always been available for me, which has played a very crucial role in coming up with this thesis.

Besides my advisor, I would like to thank Prof. Jeffrey Phillips, who motivated me towards working in the field of Data Mining even before I started working with my advisor. His lectures and guidance have helped me throughout in understanding important concepts. My sincere thanks also goes to Prof. Vivek Srikumar for his lectures on Machine Learning and his insightful comments, which have helped me in widening my perspectives on various topics related to my research area.

My special thanks goes to Arun Allamsetty, Parasaran Raman, Mina Ghashami, John Moeller, Yan Zheng, Harshit Tiwari, Deepti Deshpande, and Liang Zhang for helping me in conducting my experiments on the DBLP database.

Last but not least, I would like to thank my parents, my sister, and my friends for supporting me throughout this journey and my life in general.

CHAPTER 1

COMMUNITY-AFFINITY

1.1 Introduction

In network science, a community in a network of nodes can be described as a group of nodes which are more likely to form a connection with each other than with nodes of other communities. A good example of community structure would be a social network such as Facebook, [1] where an individual might be friends (in other words, have a connection) with people in different groups/communities such as his/her high-school friends' group, college friends' group, or workplace friends' group. His/her high-school group of friends would be more likely to have connections with each other than with his/her group of friends at the workplace. Another example of a network, which is arguably the most studied network in the literature of community-detection, is Zachary's karate club [12]. The network as shown in the Figure 1.1, was documented by the sociologist Wayne Zachary in 1977. The 34 nodes in this network represent members of a karate club and 78 links formed between them are based on how they interacted outside the club. A conflict between the club's president and the instructor resulted in a split of the club into two factions. Some members followed the president and others followed the instructor, revealing the ground truth community structure. In Figure 1.1, ground truth communities are indicated with two colors.

A lot has been studied about detecting community structures in networks. However, in comparison to community-detection, there have been only a few efforts to quantify the strength of membership of a node in a community. In this thesis, we call the strength of membership of a node to a community as *community-affinity*. This idea has been inspired from [9], where the concept of *strength of membership* is studied in Euclidean and similar spaces. One of the benefits of this measure (as we shall see later in the Section 2.4) is that it can be used collectively for all the vertices as a scoring function for the quality of a given clustering of the graph. Another application is finding the nodes which belong strongly to their communities (community representatives) and ones that are difficult to be

[1] https://www.facebook.com/

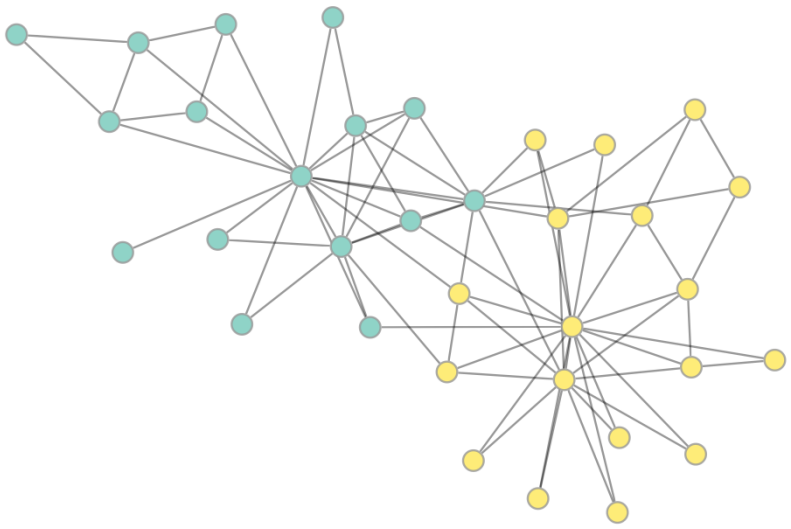

Figure 1.1: Zachary's karate club

recognized belonging to a single community. Towards the end of this thesis, we also present a community detection algorithm as an application of community-affinity.

The notion of strength of membership is very vague in the sense that there are multiple ways in which it can be formulated. What does belonging to a community mean and how should we quantify that? In simple words, community-affinity should be higher for a node towards a community if it has more similarity towards the nodes in that community and we can say that if that vertex was to be assigned a community, it would be that community. But then, how do we measure this similarity? In the following sections, we study different notions of community-affinity, but before that, we introduce the notations that we will be using throughout this thesis.

1.2 Notations used

In this thesis, we use the word *network* interchangeably with *graph*, *node* with *vertex*, *community structure* with *clustering*, and *community* with *cluster*. We use the common notations generally associated with graphs. V denotes the set of vertices and E denotes the set of edges in the graph G. $E(U)$ denotes the set of edges that have their both end vertices in the set of vertices U. $\zeta = \{c_1, c_2, .., c_k\}$ denotes the set of communities for a given clustering of G, where $c_1, c_2, .., c_k$ are the clusters of vertices. $N(v, S)$ denotes the set of neighboring vertices (or neighbors, in short) of v which are also present in the set of vertices S. $w(v, u)$ represents the weight of the edge between the vertices v and u and

$deg(v)$ denotes the degree of the node v. The set of edges that are incident on a vertex v and any vertex in the community c is denoted by $e(v, c)$. The community-affinity of a vertex v towards a community c is denoted by $\alpha(v, c)$. Throughout this thesis, we assume that the community c is one of the communities in the community structure (or clustering) ζ. Unless and until specified, all the graphs studied in this thesis are undirected and unweighted.

1.3 Notions of community-affinity

Community-affinity of a node towards a community can be defined in very simple words as a measure of how much the node is getting "pulled" by that community. There can be many different ways in which this "pull" can be formulated. We think that there are two basic properties that can be used to define community-affinity:

1. **Adhesion:** This can be thought of as a property of a vertex, which measures the strength of connectivity of the vertex with the vertices in the community. The greater the number of edges between the vertex and the community, the greater would be the adhesion of the vertex with the community.

2. **Cohesion:** This can be thought of as a property of a community, which measures the strength of connectivity of nodes in the community. A community which is a clique [2] would be considered to have maximum cohesion.

Using these two basic properties (adhesion and cohesion), we define three different notions of community affinity:

1.3.1 Notion of community-affinity based on adhesion

A very basic form of measuring community-affinity on the basis of adhesion is to define it in the following ways:

$$\alpha_{sum}(v, c) = \frac{|e(v, c)|}{\sum_{i \in \zeta} |e(v, i)|} \tag{1.1}$$

and,

$$\alpha_{max}(v, c) = \frac{|e(v, c)|}{\max_{i \in \zeta} |e(v, i)|} \tag{1.2}$$

Let us take a simple example of a graph as shown in Figure 1.2. Also, from here on, we shall refer to any community by the color with which it is colored in the corresponding

[2] A clique is a subgraph in which all vertices are connected to each other via an edge

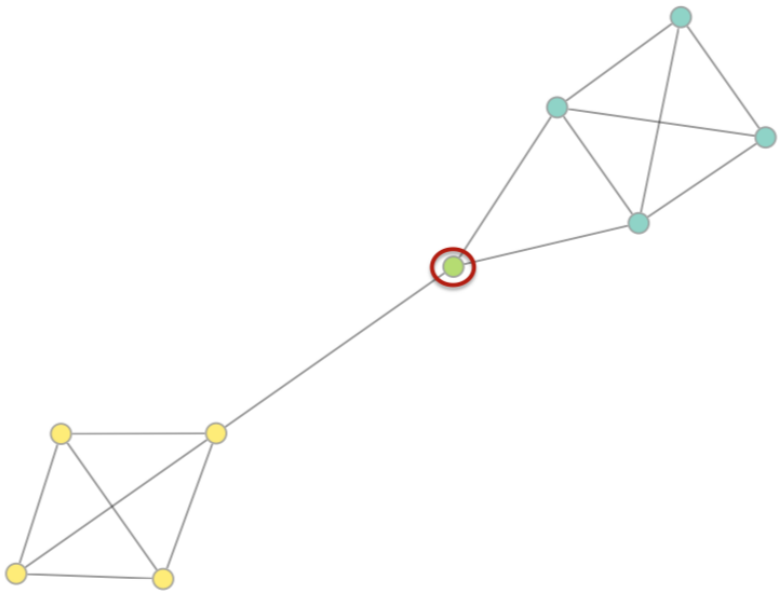

Figure 1.2: Toy example of a graph to show how the first notion for community-affinity works

diagrams and our node of interest v with a red circle in the diagrams. According to the formulations given above, the values of community-affinity are as shown in Table 1.1.

Now, let us take another example of a graph as shown in Figure 1.3. The values of community-affinity, α_{sum} and α_{max}, both amount to 0.5 for both the communities. However, we can see that the assignment of the vertex of interest to the green community would not be the same as its assignment to the yellow community. The two formulations α_{sum} and α_{max} based on adhesion fail to capture the impacts of shape and size of the communities. Had we considered the shape of the communities (i.e., how the vertices of the communities are linked with each other via internal edges) for the graph in Figure 1.3, we would have seen an imbalance between the community-affinities towards the green and the yellow community.

Table 1.1: Community-affinities for graph in Figure 1.2

	$c = green$	$c = yellow$
$\alpha_{sum}(v, c)$	0.667	0.333
$\alpha_{max}(v, c)$	1.0	0.5

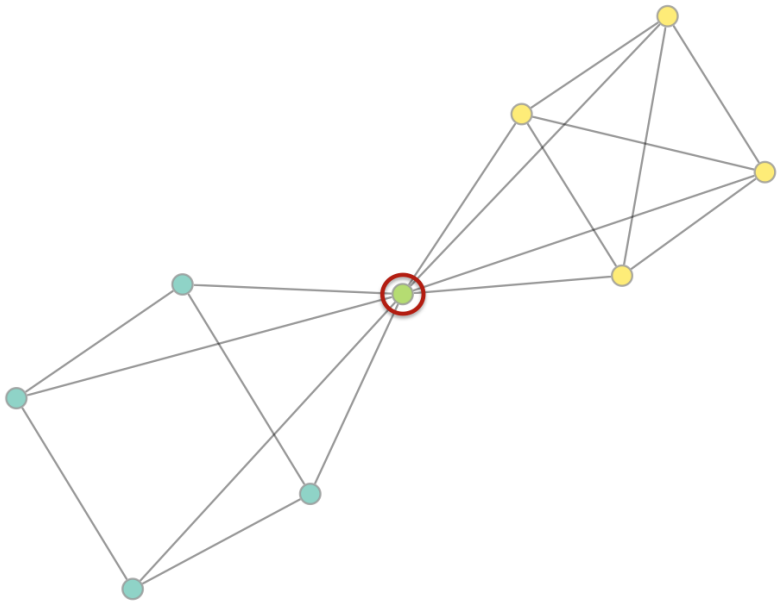

Figure 1.3: Toy example of a graph to demonstrate a drawback in the first notion of community-affinity

1.3.2 Notion of community-affinity based on adhesion and cohesion

The shortcomings of the previous notion can be rectified if we also take the size and shape of the community into consideration. One such formulation based on this notion is *permanence* [1]. Here, we refer community-affinity due to permanence as α_{perm}. The permanence is defined as follows:

$$\alpha_{perm}(v,c) = \Big(\frac{|e(v,c)|}{\max\limits_{i\in\zeta}|e(v,i)| \times deg(v)}\Big) + \Big(C_{in}(v,c) - 1\Big) \tag{1.3}$$

Here, $C_{in}(v,c)$ is the clustering coefficient of v with respect to its neighbors in the community c, which is calculated as:

$$C_{in}(v,c) = \frac{|E(N(v,c))|}{\binom{|N(v,c)|}{2}} \tag{1.4}$$

We can see from the formulation of α_{perm} that adhesion of v to c is taken care of by the first summand and the cohesion of c is taken care of by the second. The first summand is the degree normalized α_{max} that we have already seen before. The second

summand describes how densely the neighbors of v, which are also a part of community c, are connected. Greater value of $C_{in}(v, c)$ indicates greater cohesion in the community c. Hence, $\alpha_{perm}(v, c)$ can be said to be proportional to both adhesion as well as cohesion. The value of permanence ranges from 1 (v is strongly connected to c) to -1 (v is weakly connected to c). For the circled vertex in graph in Figure 1.3, $\alpha_{perm}(v, green) = -0.2083$ and $\alpha_{perm}(v, yellow) = 0.125$, which demonstrates the stated property of permanence.

We now introduce two new formulations of community-affinity, which are based on personalized PageRank (PPR) [7]. PPR for a source vertex u and a target vertex v can be defined as the probability with which one can find a random walker at v at stationary distribution when at each step, the random walker teleports back to u with the probability of t. Let the PPR from the source vertex u to a vertex v, with teleport probability t given as $ppr_t(u, v)$. Then, we can express PPR as:

$$ppr_t(u, v) = (1 - t) \times \sum_{z \in N(v,V)} ppr_t(u, z) \tag{1.5}$$

and,

$$ppr_t(u, u) = t \times \sum_{z \in V} ppr_t(u, z) + (1 - t) \times \sum_{z \in N(u,V)} ppr_t(u, z) \tag{1.6}$$

We can formulate community-affinity based on PPR in two possible ways as:

$$\alpha_{sprd}(v, c) = \sum_{u \in c} ppr(v, u) \tag{1.7}$$

and

$$\alpha_{conv}(v, c) = \sum_{u \in c} ppr(u, v) \tag{1.8}$$

In the equations 1.7 and 1.8, we fix t to the most commonly used value for PageRank computations as 0.15. Whereas α_{sprd} tells us how probable it is to find a random walker in the community c if it teleports to v at each step with a probability t, α_{conv} gives information on how probable it is to find the random walker at v which has teleport probability of t to vertices $\in c$. Both the formulations are normalized with respect to all the communities so that $\sum_{c \in \zeta} \alpha(v, c) = 1$. These formulations based on PPR consider both adhesion and cohesion. Both of them are directly proportional to adhesion as more edges with the community means easy access from community to node as well as from node to community. α_{sprd} is directly proportional to cohesion as the tighter the community connection, the easier it is to hold the random walker in the community (i.e., if there are more number of edges within the community members, there is more chance that the random walker moves to the community

members). However, α_{conv} is inversely proportional to cohesion because higher cohesion in the community implies more difficulty in escaping the community and reaching the node. For the graph in Figure 1.3, we can see that we get the results as expected. The results (community-affinities) are shown in Table 1.2

We see a higher value of α_{perm} towards the yellow community as the yellow vertices to which v is connected have a very good connectivity between themselves, as compared to the green community. We see opposite behavior between α_{conv} and α_{sprd}. Due to sparsity of the green community, it is easier for a random walker to reach v from the green community than from the yellow community, and hence, the value of α_{conv} affinity is higher for the green community. Since the yellow community is highly connected internally, it is more probable for the random walker to be absorbed in the yellow community (than in the green community) once it reaches it from v. Hence, the value of α_{sprd} is higher for the yellow community.

We now take the example of another toy graph, as shown in Figure 1.4. Table 1.3 shows the values of community-affinities obtained for this graph using different formulations discussed now and we can see some disagreements between the formulations. It is not yet very clear at this point whether the community-affinity should be proportional to cohesion or inversely proportional to it or if there is some other relation that we are missing. In a community, we would expect vertices to be connected to each other in a very similar fashion, i.e., have similar degree distributions. In other words, removing a vertex from the community to which it has high community-affinity should not make much difference in the look of the community. Going in this direction, we now introduce our final notion of community-affinity.

1.3.3 Notion of community-affinity based on adhesion as per cohesion

In this section, we try to formulate the community-affinity with an idea that it is strongest towards a community if removing/adding the vertex of interest does not bring a major change in the structure of the community. With this idea, we should formulate the

Table 1.2: Community-affinities for graph in Figure 1.3

	$c = green$	$c = yellow$
$\alpha_{perm}(v, c)$	-0.2083	0.125
$\alpha_{conv}(v, c)$	0.4388	0.3934
$\alpha_{sprd}(v, c)$	0.3110	0.3718

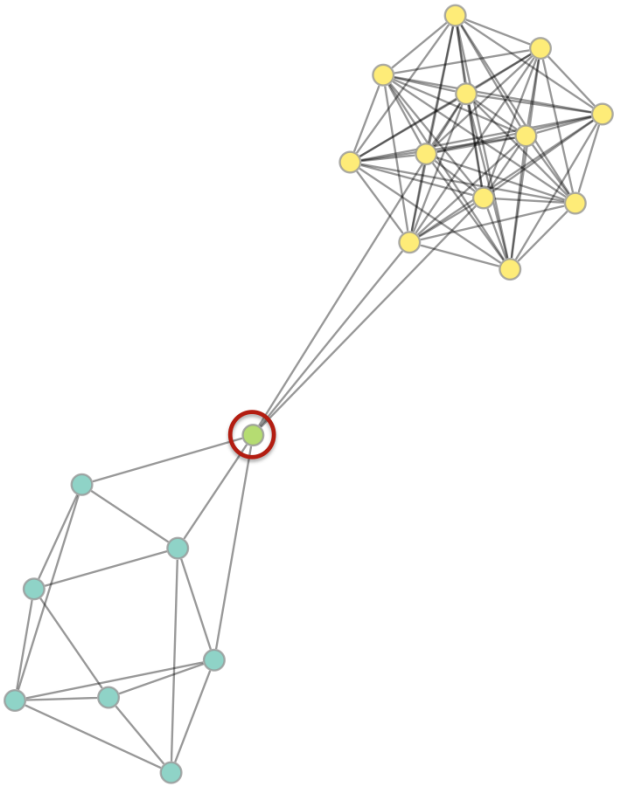

Figure 1.4: Toy example of a graph to demonstrate a drawback in the second notion of community-affinity

Table 1.3: Community-affinities for graph in Figure 1.4

	$c = green$	$c = yellow$
$\alpha_{perm}(v, c)$	-0.1667	0.6667
$\alpha_{conv}(v, c)$	0.5130	0.2635
$\alpha_{sprd}(v, c)$	0.3322	0.4731

community-affinity in such a way that its value is high if the adhesion is similar to cohesion and low if otherwise. Now consider the following formulation of community-affinity:

$$\alpha_{mod}(v, c) = \underbrace{|e(v, c)|}_{\text{X}} - \underbrace{\frac{deg(v) \times \sum_{u \in c \cup \{v\}} deg(u)}{2|E|}}_{\text{Y}} \tag{1.9}$$

We represent this formulation of community-affinity as α_{mod}, since it is based on the concept of modularity [6]. Modularity is defined as the fraction of the edges that fall within the given communities minus the expected such fraction if edges were distributed at random. α_{mod} relates to modularity as follows:

$$\frac{1}{2M}\sum_{v\in V}\alpha_{mod}(v,c) = modularity(\zeta) \tag{1.10}$$

where, c is the community assigned to v as per the clustering ζ. The part X in the equation 1.9 is the actual number of edges that the vertex v has with the community c, which is the adhesion component. Part Y is the expected number of edges that the vertex v would have had with the community c if the vertices were connected randomly (imagine each vertex as a stub formed by cutting all the edges in the middle and they connect randomly). Now if the cohesion is high in the community c, then the expectation part Y also becomes high and then, in order to have a high value of α_{mod}, the adhesion needs to be high. Similarly, if cohesion is low, then adhesion need not be high in order to have high value of α_{mod}.

We now introduce another formulation of community-affinity based on the neighborhood similarity of the vertices. By neighborhood similarity between two vertices, we mean the number of common neighbors they have with respect to the total number of neighbors that they have. This is defined as follows for vertices u and v:

$$neighborhood(v,u) = \lambda w(v,u) + \frac{|N(v,V)\cap N(u,V)|}{deg(v)+deg(u)} \tag{1.11}$$

Here, lambda is a constant to balance the importance of common neighborhood and the weight of the edge between the vertices. Value of $neighborhood(v,u)$ is high if a majority of the neighbors of v and u are common to both. The community-affinity based on neighborhood similarity could simply be written as a sum or average over all the vertices in the community:

$$\alpha_{nbr}(v,c) = \frac{1}{|c|}\sum_{u\in c} neighborhood(v,u) \tag{1.12}$$

or,

$$\alpha_{unbr}(v,c) = \sum_{u\in c} neighborhood(v,u) \tag{1.13}$$

The difference between α_{nbr} (normalized community-affinity due to neighborhood) and α_{unbr} (unnormalized community-affinity due to neighborhood) is of normalization over the number of vertices in the community. We shall see another difference between both in the Chapter 3 when we discuss community detection.

This idea of community-affinity is based on what ratio of neighborhood an average vertex in the community c shares with the vertex of interest v. It is the ratio, not the count of neighbors, that brings the factor of expected adhesion in this formulation of community affinity.

The values of community-affinities as calculated from the formulations α_{mod}, α_{nbr}, and α_{unbr} (with $\lambda = 1$) are shown in Table 1.4. We can see from these values that both α_{mod} and α_{nbr} favor the green community in the graph in Figure 1.4 to contain the vertex of interest v, but α_{unbr} does not (however, the values are quite close to each other).

1.4 Conclusion

In this chapter, we have studied a few different ways in which community-affinity can be formulated. There can be other ways and we encourage the reader to think about them. Saying that, we now move on to the next chapter, where we study our formulations in more detail and also present two applications of the idea of community-affinity.

Table 1.4: Community-affinities for graph in Figure 1.4

	$c = green$	$c = yellow$
$\alpha_{mod}(v, c)$	1.7935	-1.9891
$\alpha_{nbr}(v, c)$	0.5792	0.4145
$\alpha_{unbr}(v, c)$	4.0545	4.9737

CHAPTER 2

STABILITY

2.1 Introduction

In the previous chapter, we have seen different formulations with which we can compute community-affinity of a vertex towards a community. In this chapter, we shall try to interpret the "stability" of a vertex under a given clustering of the graph using the idea of community-affinity. *Stability* (term coined in [9]), as it translates literally, can be thought of as a measure of the unambiguity of the labeling of a vertex with a certain cluster. A more stable vertex in a given clustering would mean that we are very confident about the labeling of the vertex, whereas a less stable vertex would mean that there are other clusters that might qualify to own the vertex. We interpret stability of a vertex as the community-affinity towards the community with which it has been labeled. For a given community structure ζ, we represent the stability of a vertex v as $\sigma(v, \zeta)$. Formally,

$$\sigma(v, \zeta) = \alpha(v, c) \tag{2.1}$$

where, c is the community to which v belongs according to ζ. We shall denote the stabilities computed from different formulations of community-affinity by the appropriate subscripts. In the following sections, we will visualize and try to analyze how stability looks for different vertices using different formulations of community-affinity.

2.2 Analysis of stability as produced by different formulations

Let us take an example of another hand-constructed graph with known community structure as shown in Figure 2.1. We have colored the vertices as per the communities to which they belong. This graph also shows different kinds of communities (star-shaped, triangles/cliques, etc.) that are usually found in real-world networks.

We are interested in visualizing the stability calculated from different formulations of community-affinity. The visualizations are shown in Figures 2.2 and 2.3. Following are the steps explaining how these visualizations have been created.

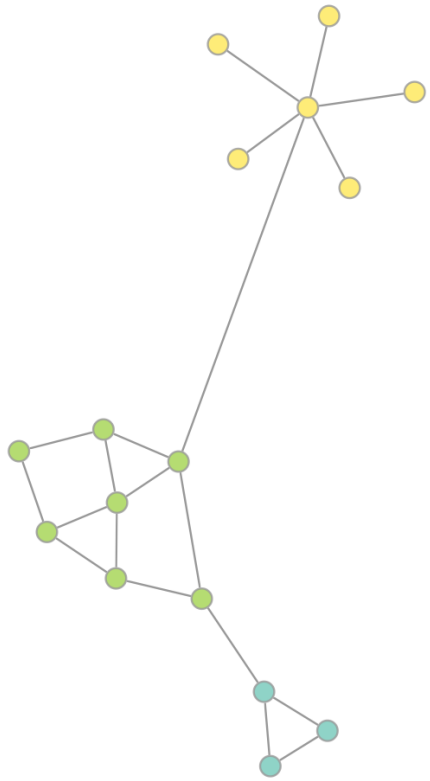

Figure 2.1: Toy example of a graph

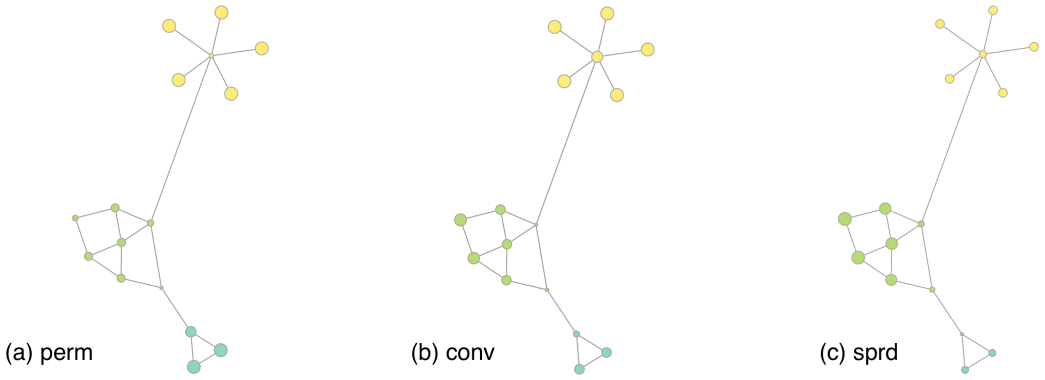

Figure 2.2: Toy example of a graph to demonstrate stability calculated from the formulations based on the second notion of community-affinity

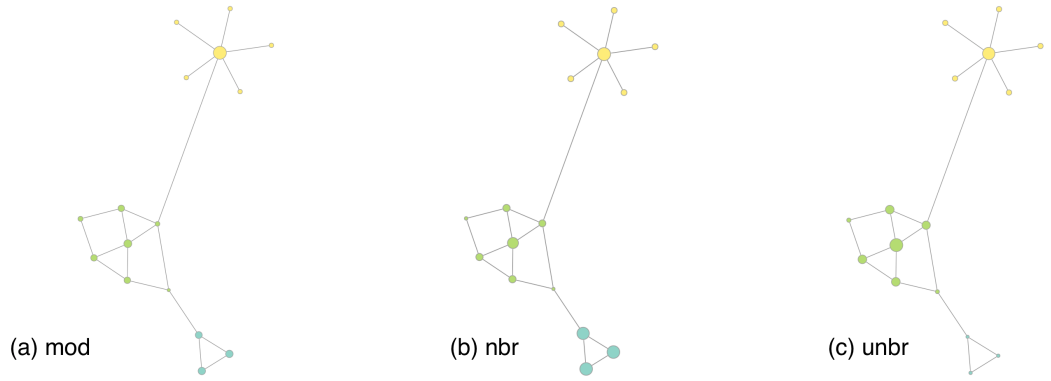

Figure 2.3: Toy example of a graph to demonstrate stability calculated from the formulations based on the third notion of community-affinity

1. Compute community-affinity of the vertices using different formulations: $\alpha_{perm}, \alpha_{conv}$, $\alpha_{sprd}, \alpha_{mod}, \alpha_{nbr}$, and α_{unbr}
2. Compute stability of the vertices as per the equation 2.1
3. Normalize the stability vector obtained in the previous step. We normalize it using linear interpolation between 0 and 15 and add 5. The values are chosen in such a way that a nice visualization is produced.
4. The normalized stability value is now used as the radius of the vertex in the visualization.

We choose cluster representative for a community as the vertex that has the highest stability in the community. One of the clear differences that we can see between different formulations of community-affinity is the selection of cluster representatives. For example, in the yellow community, α_{perm} can be seen to select the outer vertices as the cluster representatives, α_{conv} and α_{sprd} do not show much difference between the outer vertices and the inner, and the formulations based on the third notion of community-affinity select the central vertex as the cluster representative. Another difference can be seen in the community in the shape of a triangle. The formulations based on the second notion of community-affinity assign lower stability to the only vertex that is connected to another community as compared to the other vertices in the community, whereas the formulations based on the third notion do not differentiate much between these three vertices. Having a fair idea of what stability represents in a clustering of a graph, we proceed to the case study of the MNIST graph in the next section.

2.3 MNIST case study

To further study the differences in formulations of community-affinity and their effectiveness in figuring out the cluster representatives and less stable nodes, we conduct a study using a graph constructed using MNIST handwritten digits database [1]. The MNIST database consists of 28×28 pixels images, and hence, each image can be considered as a 784 dimensional object. With each image as a vertex and the distance between them calculated as the Euclidean distance in 784-dimensional space, we construct a k-nearest-neighbor graph with $k = 10$. The graph has implicit communities as the actual digit that the image (vertex) represents. We force symmetry into the graph by adding an edge from vertex u to vertex v if there is an edge from v to u. This increases the degree of some vertices and the graph does not retain its regularity property. The graph can be seen in Figure 2.4 with different colors representing different communities.

We compute stability of vertices as we did in the previous section and find the most stable and least stable vertices from different communities. The images corresponding to these vertices are shown in Figures 2.5, 2.6, 2.7, 2.8, and 2.9. As we can see from the images, the most stable digits are actually easier to recognize than the least stable ones. An exception to this are the images produced by using α_{perm} as there is not much difference that we can observe between the least stable and most stable images (we had already seen in our previous experiment that α_{perm} has a different way of selecting the cluster representatives). The images that are easy to recognize represent the cluster that they are labeled with more accurately than the images that are difficult to recognize. Hence, it makes sense to call the most stable vertices the cluster representatives.

2.4 Scoring a clustering using stability

If we are able to compute the stabilities of vertices in a given clustering, then aggregation of the stabilities should reveal something about the quality of clustering. We call such a measure of quality of clustering derived from the stabilities of the individual vertices *graph-stability* (GS). A simple way to aggregate the stabilities of individual vertices is to sum them up. Formally, we can define this as:

$$GS_{sum}(G, \zeta) = \sum_{v \in V} \sigma(v, \zeta) \tag{2.2}$$

In order to see how well GS_{sum} performs as a measure for quality of clustering, we perform the following experiment:

[1] http://yann.lecun.com/exdb/mnist/

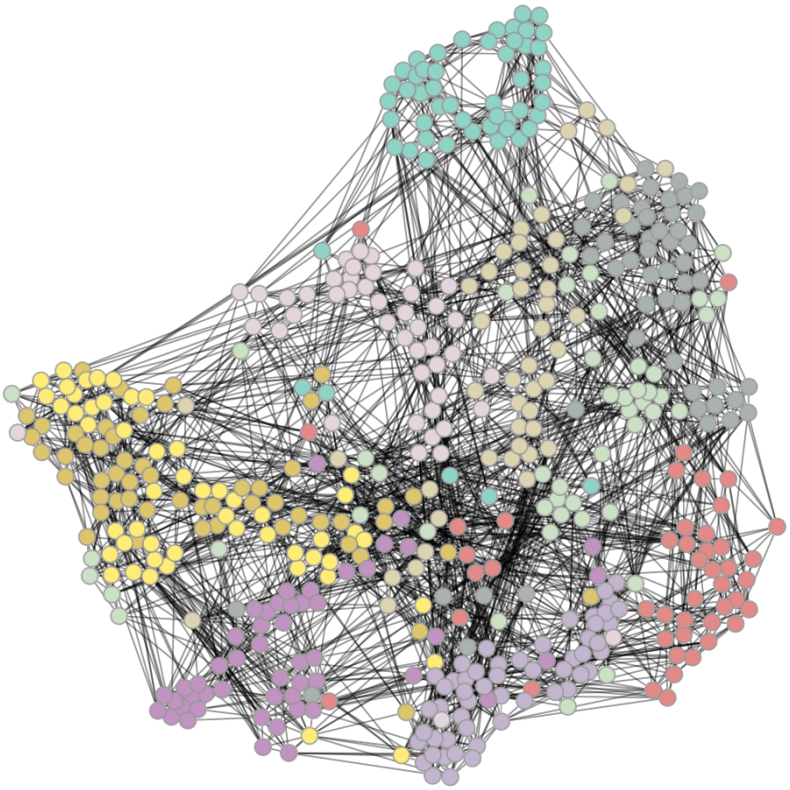

Figure 2.4: Graph constructed from MNIST handwritten digits database

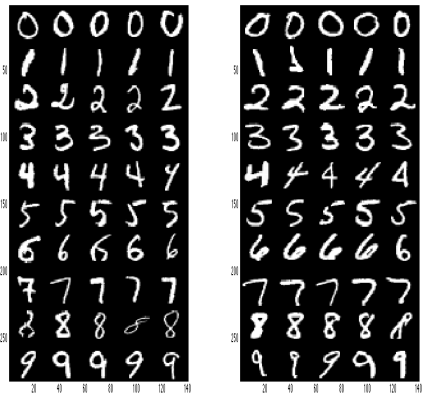

Figure 2.5: Least and most stable images found using α_{perm}

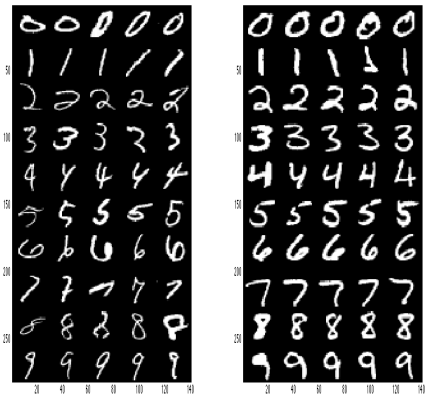

Figure 2.6: Least and most stable images found using α_{conv}

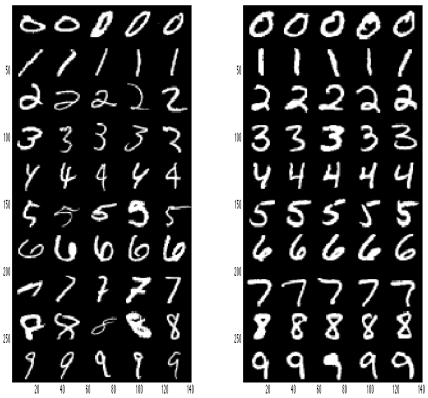

Figure 2.7: Least and most stable images found using α_{sprd}

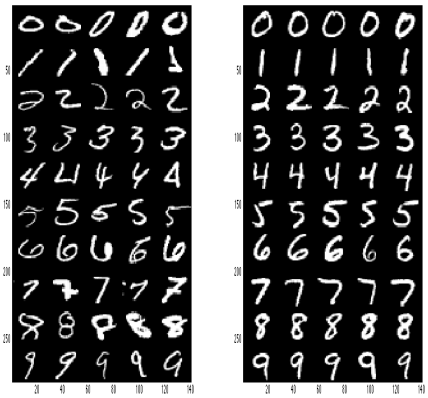

Figure 2.8: Least and most stable images found using α_{mod}

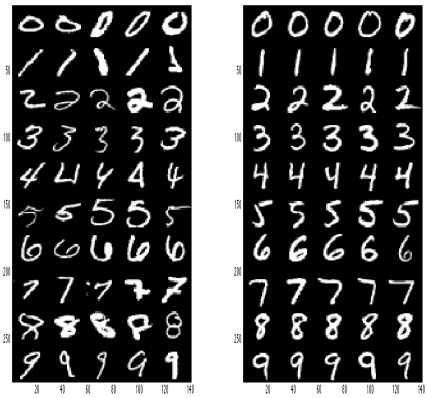

Figure 2.9: Least and most stable images found using $\alpha_{nbr}/\alpha_{unbr}$

1. Pick a graph with ground truth community structure. For this, we create a synthetic graph with 200 vertices and 6 communities. More about synthetic graphs is explained in Section 3.2

2. Using a clustering algorithm that takes "number of clusters" as input, cluster the graph with number of clusters = {1,2,3,4,5,6,7,8,9}. Obtain nine different clusterings by doing this step.

3. Measure Adjusted Rand Index (in short ARI, more can be found about this in Section 3.4) of each of the nine clusterings obtained in the previous step by comparing the clusterings with the ground truth. Normalize the values between 0 and 1.

4. Compute GS_{sum} using different formulations of community-affinity on each clustering. For the nine values obtained for each formulation of community-affinity, normalize them between 0 and 1 so that the values can be compared with the values of ARI.

5. Compare values from GS_{sum} against ARI by plotting them in a line graph.

The plots for GS_{sum} can be seen in Figure 2.10. From the plots, we can see that GS_{sum} computed from α_{mod} (which is in fact, modularity) varies perfectly according to ARI. GS_{sum} computed from α_{nbr} also does fairly well in keeping up with ARI, but other measures do not perform well.

One of the reasons, for example, for α_{unbr} not performing well can be attributed to the way it is formulated. Since, $\alpha_{unbr}(v, c)$ is computed by summing up the neighborhood similarities of v to all the vertices in the community c, it has more chances of increasing if two communities are merged together because then the size of c increases, and hence,

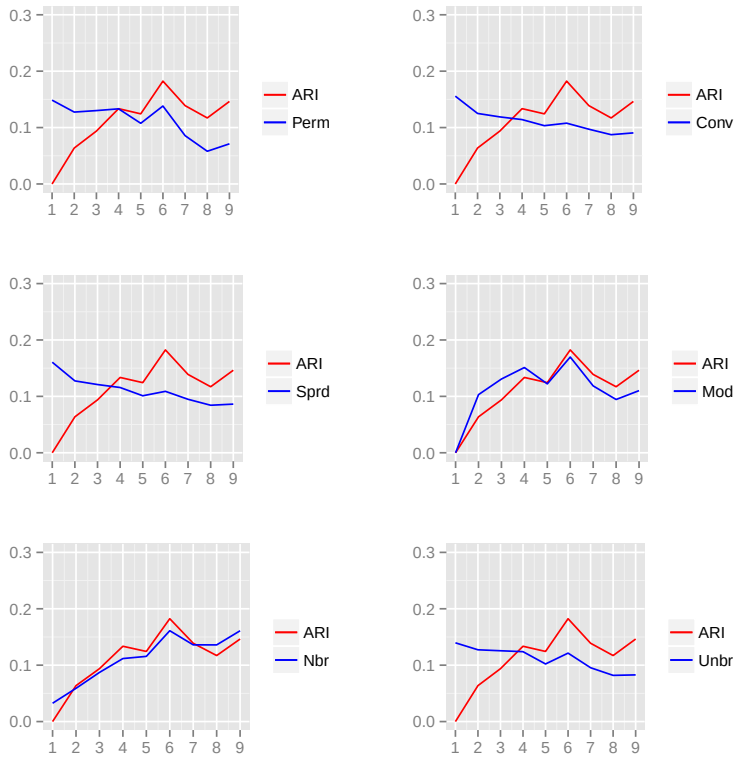

Figure 2.10: GS_{sum} variation compared with ARI for different clusterings. Number of clusters are shown in the x-axis

more neighborhood similarities get added to $\alpha_{unbr}(v, c)$. In order to compensate for this difference in number of clusters, we derive another measure for computing the quality of clustering as $GS_{sum}\times$(number of communities). We denote this as GS_{adjsum}. We perform the same experiment with GS_{adjsum} and report the results in Figure 2.11. We see from Figure 2.11 that GS_{adjsum} computed from α_{perm} varies almost exactly like ARI. GS_{adjsum} computed from α_{mod} and α_{unbr} do fairly well too.

2.5 Conclusion

In this chapter, we have derived stability from the idea of community-affinity as a measure of confidence with which a vertex has been labeled. We have also shown with the MNIST example why the more stable vertices can be called better representatives of clusters with which they have been labeled. Estimating the quality of clustering is a frequently discussed topic in the literature of community detection, for which we have seen that stability can be very handy (permanence multiplied by the number of clusters and modularity being the best of all the measures for computing quality of clustering using

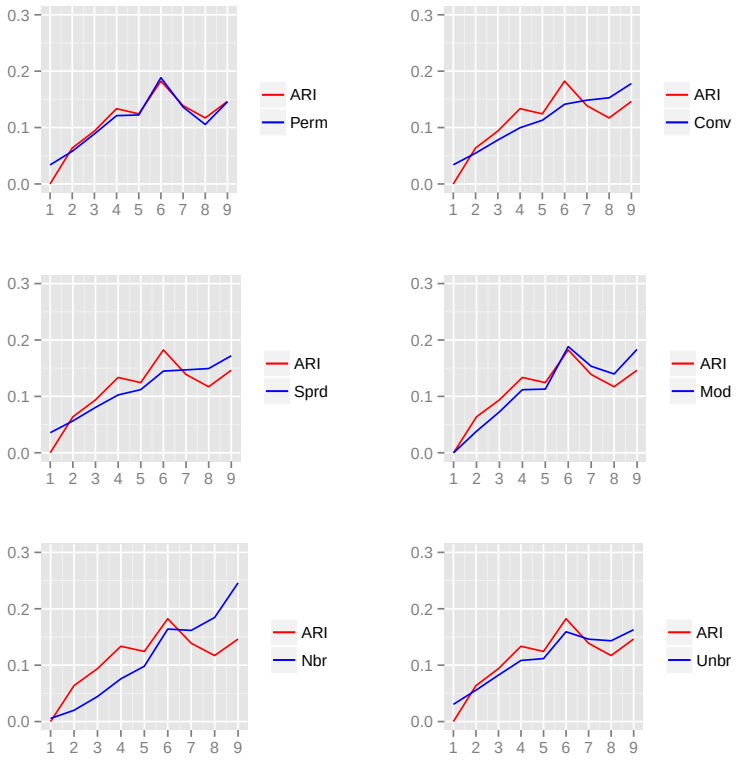

Figure 2.11: GS_{adjsum} variation compared with ARI for different clusterings. Number of clusters are shown in the x-axis

stability). In the next chapter, we introduce another application of community-affinity as community-detection, which is arguably the most researched topic in network science.

CHAPTER 3

CLUSTERING USING COMMUNITY-AFFINITY

3.1 Introduction

In this chapter, we will present a community-detection algorithm as an application of community-affinity. In general, one would think of exploiting the idea of community-affinity in the process of re-organizing the nodes in communities already formed, while solving the community-detection problem . For example, if we have a rough idea of the existing communities in the graph, we can move nodes from one community to another if there is a gain in the value of community-affinity towards the new label, and refine on the community structure found earlier. We will study two such algorithms which use this principle for finding community structure. We first describe the graphs, which we will be using for testing the community-detection algorithms.

3.2 Test graphs

1. **LFR benchmark graphs**: These are synthetic graphs with built-in community-structure [5] and resemble networks in the real world. As described in [5], these are generated as follows:

   - The degree and the community size distributions are assumed to be power laws, with exponents γ and β, respectively. The number of nodes is N and average degree is k_{avg}.
   - Each node is given a degree taken from a power law distribution with exponent γ. The extremes of the distribution k_{min} and k_{max} are chosen such that the average degree is k_{avg}.
   - Each node shares a fraction $1-\mu$ of its links with the other nodes of its community and a fraction μ with the other nodes of the network; μ is the mixing parameter.
   - The sizes of the communities are taken from a power law distribution with exponent β, such that the sum of all sizes is equal to N. The minimal and maximal

community sizes s_{min} and s_{max} are chosen so as to respect the constraints imposed by our definition of community: $s_{min} > k_{min}$ and $s_{max} > k_{max}$. This ensures that a node of any degree can be included in at least a community.

- At the beginning, all nodes are homeless, i.e., they are not assigned to any community. In the first iteration, a node is assigned to a randomly chosen community; if the community size exceeds the internal degree of the node (i.e., the number of its neighbors inside the community), the node enters the community, otherwise it remains homeless. In successive iterations, a homeless node is placed to a randomly chosen community: if the latter is complete, a randomly selected node is removed from the community, which becomes homeless. The procedure stops when there are no more homeless nodes.
- To enforce the condition on the fraction of internal neighbors expressed by the mixing parameter μ, several rewiring steps are performed, such that the degrees of all nodes stay the same and only the split between internal and external degree is affected, when needed. In this way, the ratio between external and internal degree of each node in its community can be set to the desired share μ with good approximation.

The LFR benchmark graphs that we use vary in the number of vertices N and the mixing coefficient μ. An LFR graph with 20 vertices and 0.2 mixing coefficient is denoted in this thesis as L(20,0.2). The rest of the parameters are set with the default values mentioned in the authors' implementation.[1]

2. **Zachary's karate club** [12]: Already described in Section 1.1, this network consists of 34 vertices, 78 edges, and 2 ground-truth communities.

3. **American Football Network**: A network of American football games between Division IA colleges during regular season Fall 2000. This network was proposed by Newman and Girvan in [4] but later modified by T.S. Evans in [3]. The network consists of 115 vertices, 613 edges, and 12 ground-truth communities.

4. **Political Books**: A network of books about recent US politics sold by the online bookseller Amazon.com. Edges between books represent frequent co-purchasing of books by the same buyers. The network was compiled by V. Krebs and is unpublished,

[1] https://sites.google.com/site/santofortunato/inthepress2

but can found on Krebs' web site [2]. The network consists of 105 vertices, 441 edges, and 3 ground-truth communities.

There are not many real-world graphs with ground-truth community structure available, and hence, we use only the ones described above. To compensate, we use a variety of LFR benchmark graphs. All the graphs used here are well-known as benchmarks for community-detection.

3.3 Community-detection algorithms for comparison

We compare our community-detection algorithm with some state of the art algorithms for community detection based on different principles.

1. **Combo** [11]: Combo is a modularity-optimization algorithm which performs several operations such as merging two communities, splitting a community into two, and moving nodes between two distinct communities. As the proposed technique combines all three possible types of steps to optimizae modularity, it is referred to as Combo. Its performance over other well-known algorithms can be seen here: `http://senseable.mit.edu/community_detection/`.

2. **Infomap** [10]: Infomap is an information theoretic approach that uses probability flow of random walks as a proxy for information flows in the real system and decomposes the network into modules by compressing a description of the probability flow.

3. **Walktrap** [8]: Walktrap is a community detection algorithm based on the fact that a random walker tends to be trapped in the dense part of a network corresponding to communities.

3.4 Metrics to compute quality of clustering

We use Adjusted Mutual Information (AMI) and Adjusted Rand Index (ARI) to compute similarity between the clustering obtained by community-detection algorithms and the ground truth. Description of the methods can be seen here: `http://scikit-learn.org/stable/modules/clustering.html#clustering-evaluation`. We also use the implementation of these methods from the same resource.

[2] http://www.orgnet.com/

3.5 Extremal optimization

Before we introduce our novel peeling algorithm for community-detection, we would like to mention another algorithm for community detection, which was introduced by Jordi Duch and Alex Arenas in [2]. It uses Extremal Optimization technique (and hence, is known as the EO algorithm) to optimize modularity using a fitness function that resembles degree normalized α_{mod}. We consider it important to mention this algorithm here as we think it uses the concept of community-affinity to detect communities in a given graph. The fitness function used in the algorithm, which measures fitness of a vertex v in a given community c, is described as:

$$fitness(v, c) = \frac{|e(v, c)|}{deg(v)} - \frac{\sum_{u \in c \cup \{v\}} deg(u)}{2M} \tag{3.1}$$

which can be obtained as $\alpha_{mod}/deg(v)$. Normalizing with respect to the degree of the vertex removes the bias due to high degree of a vertex. The EO algorithm, in short, is as follows:

- Split the nodes of the whole graph in two random partitions having equal number of nodes. This splitting creates an initial communities division, where communities are understood as connected components in each partition.
- At each time step, the system self-organizes by moving the node with the lower fitness (extremal) from one partition to the other.
- The process is repeated until an "optimal state" with a maximum value of modularity (the objective function) is reached. After that, we delete all the links between both partitions and proceed recursively with every resultant connected component. The process finishes when the modularity cannot be improved.

The EO algorithm is not widely used for community-detection as it is both slow and less accurate compared to the other algorithms such as the ones mentioned in the Section 3.3.

3.6 Peeling algorithm

We now introduce our peeling algorithm, which uses α_{nbr} and α_{unbr} to detect communities. We have devised two versions of the algorithm. One is deterministic and the other is probabilistic. They only differ in the first step, and are described as follows:

- **Prepare a seed community**: In the deterministic algorithm, we pick the vertex that has the highest degree as the initial seed and in the probabilistic algorithm, we pick a vertex with probability proportional to its degree as the initial seed. A seed

community is formed by adding the initial seed, its first degree neighbors, and second degree neighbors, which are neighbors to at least δ (a hyper-parameter) first degree neighbors. Let us call this as community A. The rest of the nodes in the graph form community B.

- **Organize communities**: We get the community-affinities of the vertices towards these two communities. We first move the vertices that have been placed in community A to B if they have more affinity towards B than towards A. We do this and recalculate community-affinities until we have no vertex to move. After this, using the same process, we move vertices from B to A if they have more affinity towards A than towards B.

- **Peel**: Remove the vertices (and their edges) from the graph, which are members of the smaller community among A and B, and recurse for the graph that is left over.

- We stop when either A or B is found empty.

There are two hyper-parameters in this algorithm: λ (in the formulation of α_{nbr} and α_{unbr}) and δ. The only advantage of the deterministic algorithm over the probabilistic one is the speed of execution. As we shall see in the next section, the probabilistic algorithm performs on a much wider range of graphs as compared to the deterministic algorithm.

3.7 Experiments and results

In this section, we present experiments and their results to establish the usefulness of the peeling algorithms that we have just described.

3.7.1 Selecting hyper-parameters for the peeling algorithms

We first test our peeling algorithms on different combinations of hyper-parameters. We denote the peeling algorithms as $Peel_{det}(nbr)$ (deterministic and using α_{nbr}), $Peel_{prb}(nbr)$ (probabilistic and using α_{nbr}), $Peel_{det}(unbr)$ (deterministic and using α_{unbr}), and $Peel_{prb}(unbr)$ (probabilistic and using α_{unbr}). Tables 3.1, 3.2, 3.3, and 3.4 show a matrix of ARI values obtained by comparing the clusterings obtained with different combinations of hyper-parameters with corresponding ground truths. For the probabilistic algorithms, we take the average over 10 executions. The graph that we use for testing is $\mathrm{L}(200, 0.3)$, which is neither very easy nor very difficult for detecting communities. We see from the tables that the algorithms that use α_{nbr} ($Peel_{det}(nbr)$ and $Peel_{prb}(nbr)$) do not perform well for any combination of hyper-parameters.

Table 3.1: ARI values for L(200, 0.3) using $Peel_{det}(nbr)$

λ \ δ	0	1	2	3	4	5	6	10000
0	0	0.50199	0.43717	0.67097	0.35314	0.36192	0.38851	0.46745
0.5	0	0.59422	0.59422	0.68151	0.59524	0.53699	0.39025	0.39278
1	0	0.59422	0.59422	0.68151	0.66009	0.55118	0.46538	0.44437
2	0	0.59422	0.59422	0.68151	0.66602	0.55512	0.42663	0.48841
3	0	0.60885	0.60885	0.68233	0.65617	0.55512	0.54765	0.48662
4	0	0.60885	0.60885	0.68233	0.65756	0.55512	0.54765	0.44186
5	0	0.60885	0.60885	0.68233	0.65756	0.55512	0.54705	0.48319
6	0	0.60885	0.60885	0.68233	0.65756	0.55512	0.54705	0.48319

Table 3.2: ARI values for L(200, 0.3) using $Peel_{prb}(nbr)$

λ \ δ	0	1	2	3	4	5	6	10000
0	0	0.54973	0.50618	0.48134	0.41673	0.44662	0.48748	0.42309
0.5	0	0.55503	0.57233	0.51985	0.50665	0.48371	0.47496	0.44385
1	0	0.59304	0.60642	0.50551	0.46964	0.44048	0.43704	0.43536
2	0	0.59744	0.55275	0.56425	0.55352	0.47664	0.43104	0.35557
3	0	0.58133	0.57104	0.45707	0.49449	0.42855	0.43903	0.42297
4	0	0.58512	0.58862	0.52771	0.47108	0.48631	0.44862	0.38553
5	0	0.55653	0.55915	0.52141	0.48345	0.44954	0.44613	0.35231
6	0	0.57112	0.57879	0.4926	0.5134	0.46084	0.44558	0.32367

Table 3.3: ARI values for L(200, 0.3) using $Peel_{det}(unbr)$

λ \ δ	0	1	2	3	4	5	6	10000
0	0	0	0	0	0	0	0	0
0.5	0	0	0	0.29263	1	1	0.98659	0.98659
1	0	0	0	0.60885	1	1	1	1
2	0	0	0	0.60885	1	1	1	1
3	0	0	0	0.60885	1	1	1	1
4	0	0	0	0.60885	1	1	1	1
5	0	0	0	0.60885	1	1	1	1
6	0	0	0	0.60885	1	1	1	1

Table 3.4: ARI values for L(200, 0.3) using $Peel_{prb}(unbr)$

λ \ δ	0	1	2	3	4	5	6	10000
0	0	0.68628	0.66628	0.80706	0	0	0	0
0.5	0	0.72719	0.87079	0.85147	0.94939	0.99685	0.99732	0.99239
1	0	0.7774	0.88726	0.85852	0.95491	0.99752	0.99743	0.98349
2	0	0.7296	0.94722	0.93173	0.96326	0.99306	1	0.99875
3	0	0.76201	0.92178	0.93244	0.98225	1	1	1
4	0	0.78249	0.91402	0.88628	0.98225	1	1	1
5	0	0.78785	0.94076	0.91942	0.98225	1	1	1
6	0	0.80954	0.9364	0.91942	0.98225	1	1	

The reason for this poor performance lies in the normalization. Since α_{nbr} in its computation of community-affinity of a vertex v towards a community c divides the sum of neighborhood similarities by the size of c, even a part of c (if c has edges almost evenly distributed) would offer the same community-affinity to v as the complete c does. This is clearly illustrated in the videos `https://www.youtube.com/watch?v=hXEFL3DPP2c` and `https://www.youtube.com/watch?v=bD_JTE-qtzQ`, which show step by step how the communities are peeled away.

In the future experiments, we test with only $Peel_{det}(unbr)$ and $Peel_{prb}(unbr)$ since the other two are not very useful for community-detection and we have shown why. From here on, we will simply refer to the algorithms $Peel_{det}(unbr)$ and $Peel_{prb}(unbr)$ as $Peel_{det}$ and $Peel_{prb}$, respectively. We further test our peeling algorithms and narrow down on the choices of hyper-parameters that give decent results. We test using the LFR graphs having $\mu = 0.3, 0.4$ (our algorithms do not work for $\mu = 0.5$) and $N = 200, 500, 1000, 5000, 10000$. The results for the graphs can be seen in the Tables 3.5, 3.6, 3.7, 3.8, 3.9, and 3.10. From these results, we conclude the following:

- $Peel_{det}$ is unable to cluster the LFR graphs that have mixing coefficient greater than 0.3 and $Peel_{prb}$ is unable to cluster the ones having mixing coefficient greater than 0.4.

- $Peel_{det}$ gives good results for $\lambda = 1, 2, 3, 4, 5, 6$ and $\delta = 4, 5, 6$, while $Peel_{prb}$ gives good results for $\lambda = 4, 5, 6$ and $\delta = 2, 3, 4, 5$.

We suggest two variations of each $Peel_{det}$ and $Peel_{prb}$ now: one that takes a single combination of hyper-parameters and returns a clustering (denoted as fast $Peel_{det}$ and

Table 3.5: ARI values for L$(200, 0.4)$ using $Peel_{det}(unbr)$

λ \ δ	0	1	2	3	4	5	6	10000
0	0	0	0	0	0	0	0	0
0.5	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0.16851	0	0
3	0	0	0	0	0	0.17906	0	0
4	0	0	0	0	0	0.91813	0	0
5	0	0	0	0	0	0.91813	0	0
6	0	0	0	0	0	0.91813	0	0

Table 3.6: ARI values for L$(1000, 0.3)$ using $Peel_{det}(unbr)$

λ \ δ	0	1	2	3	4	5	6	10000
0	0	0	0	0	0	0	0	0
0.5	0	0.38922	0.96407	1	1	0.99787	0.99787	0.00536
1	0	0.38922	0.92179	1	1	1	1	0.01242
2	0	0.36252	0.92179	1	1	1	1	0.01991
3	0	0.36252	0.92179	1	1	1	1	0.01991
4	0	0.36252	0.92179	1	1	1	1	0.01991
5	0	0.36252	0.92179	1	1	1	1	0.01991
6	0	0.36252	0.92179	1	1	1	1	0.01991

Table 3.7: ARI values for L$(1000, 0.4)$ using $Peel_{det}(unbr)$

λ \ δ	0	1	2	3	4	5	6	10000
0	0	0	0	0	0	0	0	0
0.5	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0
3	0	0.40204	0	0	0	0	0	0
4	0	0.40248	0	0	0	0	0	0
5	0	0.40248	0	0	0	0	0	0
6	0	0.40248	0	0	0	0	0	0

Table 3.8: ARI values for L(200, 0.4) using $Peel_{prb}(unbr)$

λ \ δ	0	1	2	3	4	5	6	10000
0	0	0	0	0	0	0	0	0
0.5	0	0.23162	0.7833	0	0.62422	0	0	0
1	0	0.62091	0.71075	0	0.61466	0	0	0
2	0	0.81956	0.83128	0.90432	0.6852	0.16851	0	0
3	0	0.81715	0.83472	0.97568	0.84652	0.17906	0	0
4	0	0.80954	0.89353	0.93607	0.88348	0.84422	0	0
5	0	0.86095	0.90316	0.95762	0.7992	0.84422	0	0
6	0	0.87341	0.85621	0.96094	0.87769	0.91813	0	0

Table 3.9: ARI values L(1000, 0.3) using $Peel_{prb}(unbr)$

λ \ δ	0	1	2	3	4	5	6	10000
0	0	0	0	0	0	0	0	0
0.5	0	0.57252	0.96906	0.99359	0.99175	0.98991	0.98556	0.98926
1	0	0.53023	0.97381	0.9968	0.99625	0.99677	0.995	0.99404
2	0	0.54083	0.98202	0.99886	0.99882	0.99783	0.99953	0.99837
3	0	0.53407	0.97903	0.99875	0.9992	0.99988	0.99889	1
4	0	0.50325	0.96902	1	0.99969	0.99969	0.99896	0.9978
5	0	0.55681	0.98031	0.99885	0.99853	0.99924	1	0.99816
6	0	0.5794	0.97173	0.99964	0.99912	1	1	0.99872

Table 3.10: ARI values for L(1000, 0.4) using $Peel_{prb}(unbr)$

λ \ δ	0	1	2	3	4	5	6	10000
0	0	0	0	0	0	0	0	0
0.5	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0
2	0	0.71956	0.96617	0.96077	0.95336	0.0029656	0.00259	0
3	0	0.74153	0.96607	0.9685	0.96957	0.94981	0.00640	0
4	0	0.68501	0.97732	0.97506	0.96791	0.95833	0.75967	0
5	0	0.74046	0.97242	0.97131	0.96335	0.95535	0.94175	0
6	0	0.68988	0.96242	0.96454	0.97314	0.96145	0.94428	0

$Peel_{prb}$) and the other that performs clustering on a set of hyper-parameter values and selects the clustering that has highest quality of clustering. For fast $Peel_{det}$, we choose the hyper-parameters as $\lambda = 5$ and $\delta = 4$ and for fast $Peel_{prb}$, we choose the hyper-parameters as $\lambda = 4$ and $\delta = 3$. We judge the quality of clustering by the highest value of modularity. We have already seen in the Section 2.4 that modularity works well for measuring the quality of clustering.

3.7.2 Comparison with other algorithms

We now compare the accuracy and efficiency of the community-detection algorithms that we have discussed so far. Table 3.11 shows the accuracy comparisons and Table 3.12 shows the efficiency comparisons. The results are obtained by running the algorithms (C,C++ implementations) on a computer running OS X 10.9.5 with 16 GB memory and using 2.6 GHz Intel Core i5 processor. Our results show that the peeling algorithms are quite accurate but not very fast. They are more accurate than Combo and Walktrap but less accurate than Infomap. The MATLAB implementations of peeling algorithms are faster than that

Table 3.11: Accuracy produced by the algorithms for different graphs in AMI,ARI

	fast $Peel_{det}$	$Peel_{det}$	fast $Peel_{prb}$	$Peel_{prb}$	Combo	Infomap	Walktrap
L(200, 0.2)	1,1	1,1	1,1	1,1	1,1	1,1	1,1
L(200, 0.3)	1,1	1,1	1,1	1,1	1,1	1,1	1,1
L(200, 0.4)	0,0	0.93,0.92	0.98,0.99	0.98,0.99	1,1	1,1	0.95,0.96
L(500, 0.2)	1,1	1,1	1,1	1,1	1,1	1,1	1,1
L(500, 0.3)	1,1	1,1	1,1	1,1	1,1	1,1	1,1
L(500, 0.4)	0,0	0,0	0.96,0.95	0.99,0.99	1,1	1,1	0.99,0.99
L(1000, 0.2)	1,1	1,1	1,1	1,1	0.99,0.98	1,1	1,1
L(1000, 0.3)	1,1	1,1	1,1	1,1	0.99,0.98	1,1	0.98,0.99
L(1000, 0.4)	0,0	0,0	0.98,0.98	0.98,0.98	0.98,0.97	1,1	0.99,0.98
L(1000, 0.5)	0,0	0,0	0,0	0,0	0.95,0.92	1,1	0.97,0.96
L(5000, 0.2)	1,1	1,1	1,1	1,1	0.88,0.77	1,1	0.95,0.90
L(5000, 0.3)	1,1	1,1	1,1	1,1	0.82,0.66	1,1	0.91,0.81
L(5000, 0.4)	0,0	0,0	0.98,0.98	0.99,0.99	0.80,0.60	1,1	0.85,0.69
L(10000, 0.4)	0,0	0,0	0.99,0.98	0.99,0.98	0.73,0.44	1,1	0.79,0.55
Karate	1,1	1,1	1,1	0.46,0.57	0.50,0.54	0.58,0.70	0.46,0.52
Football	0.90,0.90	0.90,0.90	0.84,0.82	0.90,0.90	0.82,0.81	0.90,0.90	0.82,0.82
Polbooks	0.51,0.64	0.51,0.64	0.51,0.64	0.56,0.67	0.47,0.66	0.46,0.65	0.49,0.65

Table 3.12: Time taken by the algorithms for different graphs in seconds

	fast $Peel_{det}$	$Peel_{det}$	fast $Peel_{prb}$	$Peel_{prb}$	Combo	Infomap	Walktrap
L(200, 0.2)	0.009	0.02	0.009	0.02	0.02	0.06	0.02
L(200, 0.3)	0.01	0.02	0.01	0.02	0.02	0.07	0.01
L(200, 0.4)	0.01	0.01	0.01	0.08	0.02	0.09	0.02
L(500, 0.2)	0.03	0.09	0.03	0.1	0.14	0.16	0.04
L(500, 0.3)	0.04	0.11	0.03	0.12	0.14	0.21	0.04
L(500, 0.4)	0.02	0.04	0.04	0.38	0.15	0.21	0.04
L(1000, 0.2)	0.12	0.65	0.13	0.82	0.77	0.33	0.09
L(1000, 0.3)	0.13	0.73	0.14	0.96	0.88	0.41	0.10
L(1000, 0.4)	0.06	0.15	0.17	2.4	1.0	0.41	0.10
L(1000, 0.5)	0.06	0.16	1.8	15.6	1.5	0.65	0.11
L(5000, 0.2)	10.6	93.0	12.9	113.8	40.5	1.9	1.4
L(5000, 0.3)	11.5	100.3	13.6	128.6	40.3	2.2	1.7
L(5000, 0.4)	1.9	9.5	17.7	284.6	45.1	2.7	1.8
L(10000, 0.4)	10.3	29.6	152.6	2084.8	234.2	5.4	6.9
Karate	0.005	0.006	0.004	0.005	0.007	0.02	0.008
Football	0.007	0.011	0.006	0.014	0.015	0.04	0.008
Polbooks	0.006	0.006	0.005	0.016	0.01	0.06	0.01

of the C++ ones because of the involvement of numerous sub-matrix extraction operations, which are more optimized in MATLAB than in C++ Armadillo [3] library.

An interesting aspect of the peeling algorithms can be seen in detecting communities in the real-world networks. No algorithm (to our knowledge) detects perfect communities in the Zachary's karate club network. Our algorithm not only achieves that but also performs either better or as good as other algorithms in other real-world networks. One of the reasons for the algorithms, based on modularity optimization, not working well is the well-known resolution limit problem, because of which, the modularity optimizing algorithms fail to identify small communities and tend to merge them to form larger ones. This reason can be attributed to Combo identifying just 10 communities out of 12 in the football network. The peeling algorithms face no such restrictions. The clusterings obtained for the football network, as an example, are shown in Figure 3.1.

[3] http://arma.sourceforge.net/

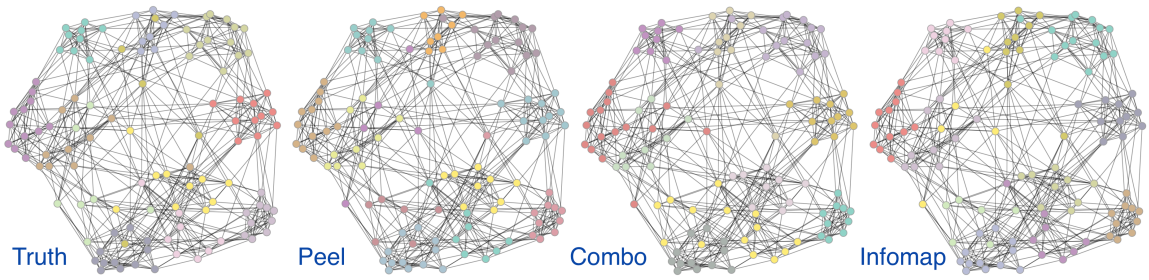

Figure 3.1: Clusterings obtained on the football network using different algorithms

3.8 Conclusion and future work

We have seen in this chapter that community-affinity measures can be employed for detecting communities as well. The peeling algorithms that we discussed are fast and accurate; however, they fail to identify clusters when mixing coefficient is high (≥ 0.5). In future, we plan to study another possible application of community-affinity in the seed expansion problem. The seed expansion problem is to discover as many community members as possible when we are given a small set of community members from the same community. We believe community-affinity might be useful in solving this problem as this problem is formulated in a very similar fashion to that in which we formulated the peeling algorithm (though the seed community members were chosen in the peeling algorithm using a heuristic).

REFERENCES

[1] Tanmoy Chakraborty, Sriram Srinivasan, Niloy Ganguly, Animesh Mukherjee, and Sanjukta Bhowmick. On the permanence of vertices in network communities. *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '14*, pages 1396–1405, 2014.

[2] Jordi Duch and Alex Arenas. Community detection in complex networks using extremal optimization. *Physical Review E*, 72(2):2–5, 2005.

[3] Tim S Evans. Clique graphs and overlapping communities. *Journal of Statistical Mechanics: Theory and Experiment*, 2010(12):P12037, 2010.

[4] Michelle Girvan and Mark EJ Newman. Community structure in social and biological networks. *Proceedings of the national academy of sciences*, 99(12):7821–7826, 2002.

[5] Andrea Lancichinetti, Santo Fortunato, and Filippo Radicchi. Benchmark graphs for testing community detection algorithms. *Physical Review E - Statistical, Nonlinear, and Soft Matter Physics*, 78(4):1–5, 2008.

[6] Mark EJ Newman. Modularity and community structure in networks. *Proceedings of the National Academy of Sciences*, 103(23):8577–8582, 2006.

[7] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: bringing order to the web. 1999.

[8] Pascal Pons and Matthieu Latapy. Computing communities in large networks using random walks. In *Computer and Information Sciences-ISCIS 2005*, pages 284–293. Springer, 2005.

[9] Parasaran Raman and Suresh Venkatasubramanian. Power to the points: Validating data memberships in clusterings. *Proceedings - IEEE International Conference on Data Mining, ICDM*, pages 617–626, 2013.

[10] Martin Rosvall and Carl T Bergstrom. Maps of random walks on complex networks reveal community structure. *Proceedings of the National Academy of Sciences*, 105(4):1118–1123, 2008.

[11] Stanislav Sobolevsky, Riccardo Campari, Alexander Belyi, and Carlo Ratti. General optimization technique for high-quality community detection in complex networks. *Physical Review E*, 90(1):012811, 2014.

[12] Wayne W Zachary. An information flow model for conflict and fission in small groups. *Journal of anthropological research*, pages 452–473, 1977.