International Journal of Advance Research and Innovation
Vol. 2(2), Apr-Jun 2014, pp. 52-60
Doi: 10.51976/ijari.221408

Article Info
Received: 05 Mar 2014 | Revised Submission: 20 Apr 2014 | Accepted: 28 May 2014 | Available Online: 15 Jun 2014

Algorithm and Techniques for Overlapping Community Detection

*Sakshi Singh**

ABSTRACT

A lot of phenomenon, real world and otherwise can be conveniently represented as graphs, with the nodes corresponding to the entities and the edges representing the interaction be- tween those entities. Communities or modules, which are groups of nodes densely connected to each other within the community but sparsely linked to other communities and the rest of the graph, often having similar structural and functional properties. A lot of algorithms have been proposed to partition the set of vertices into communities; such a partition exclusively puts a node into one community or the other. But in real life a node can belong to multiple communities simultaneously, i.e. the communities can overlap. Different metrics have been proposed. We reduce the modularity maximization problem for splitting the graph into two communities to the MAX-CUT problem with both positive and negative weights. We introduce and analyze three approximation algorithms to maximize modularity for the two community case; recursive bi-partitioning can be carried out as long as modularity increases to split into more than two communities.

Keywords: *Modularity MAX-CUT Chromosome.*

1.0 Introduction

Many real world phenomenon can be represented as graphs, directed or undirected, where the nodes represent the entities and the edges represent the interaction between those entities. Given an undirected unweighted graph G = (V, E), a community is an induced subgraph on a subset C of vertices satisfying some properties.

A rough guideline of a community is by Newman and Girvan [20]: "a community is a subgraph containing nodes which are more densely linked to each other than to the rest of the graph or, equivalently a graph has a community structure if the number of links into any subgraph is higher than the number of links between those subgraphs".But only rarely do real world graphs separate into non overlapping communities or „hard- partitions", most real networks have well defined overlapping and nested communities. There will be a set of nodes that can be put in more than one community, where a certain strict classification into one community or another is inaccurate, if not totally wrong. We reduce the modularity optimization for the two community „hard-partition" case to the MAX- CUT problem with both negative and positive weights. Further we propose and analyze three approximation algorithms for modularity maximization for the two community „hard-partition" case.We extend the Modularity metricso as to allow overlapping communities,we further discuss several properties of the extension.Reduces the extended modularity maximization problem into a Genetic Optimization problem, the algorithm is presented to maximize modularity. the results of application of our algorithm on real world graphs and computer generated overlap models and are presented and analyzed.

1.1 Modularity:a goodness measure

It is a measure of how good a particular division of a graph *G* into a set of communities *C* is larger values of Modularity indicate a better partition, or a more modular graph.

1.2 Conductance

For a Graph, G = (V, E) and a partition of the vertex set V into non-empty subsets S, S" Conductance of the cut (S, S") is defined as follows:

**Department of Computer Science & Engineering, Teerthanker Mahaveer University, Moradabad, India*
(E-mail: sakshisingh0009@gmail.com)

$\emptyset(s) = \frac{cut(s,s')}{\min(vol(s),vol(s'))}$ Here vol(S) = vol(S, V) $=\sum_{i\in s, j\in s} a_{ij}$ and cut(s, s‟) $=\sum_{i\in s, j\in s} a_{ij}$

2.0 Approximation Algorithm for Modularity Maximization

We formulate the problem of partitioning the graph into two communities so as to maximize modularity as a Strict Quadratic Program, we then relax it into a Vector Program which can be solved upto any degree of accuracy, and we finally round the solution to get back to the solution of the original problem. We use the rounding techniques based on Rietz‟ method of averaging with the Gaussian measure to get a randomized approximation algorithm with an approximation guarantee $\frac{4}{\pi} \approx$ 0.27.We reduce the modularity maximization problem into the MAX-CUT problem and then use the techniques described in Alon et.al [49] to im prove the approximation guarantee to $\frac{2\ln(1+\sqrt{2})}{\pi} \approx 0.56$

2.1 Modularity maximization as a strict quadratic program

In this section we briefly describe the formulation of modularity maximization as a Strict Quadratic Program and it‟s subsequent relaxation as a Vector Program and the rounding scheme used. Then we analyze the approximation guarantee given by this formulation. For a quick overview of Quadratic Programming and Vector Programming

2.2 Programming formulation

For every vertex u we have a variable yu $\in$ {−1, +1}. The value of yu depends upon which community it is in, if u lies in C1 then yu = 1 else yu = −1. Modularity, the objective function can be written as follows

$Q=\sum_{u,v\in V} c_{u,v\in V}\delta c_{uc_v}$

The above can be re-written as follows:

$Q=\sum_{u,v\in V} c_{uv}\delta c_{uc_v}$

$Q=\sum_{u,v\in V} c_{uv}\frac{(1+y_u y_v)}{2}$

This can be simplified as follows

$Q=\sum_{u,v\in V} c_{uv}\frac{(1+y_u y_v)}{2}$

$=\frac{1}{2}\sum_{u,v\in V} c_{uv(1+y_u y_v)}$

$=\frac{1}{2}\sum_{u,v\in V} c_{uv}(1)+\frac{1}{2}\sum_{u,v\in V} c_{uv(y_u y_v)}$

$=\frac{1}{2}\sum_{u,v\in V} c_{uv(y_u y_v)}$ because $\frac{1}{2}\sum_{u,v\in V} c_{uv}(1)=0$

$=\frac{1}{2}\sum_{u,v\in V} c_{uv}\, y_u y_v$

Therefore the optimization problem can be written as follows:

$$\text{Max: } \frac{1}{2}\sum_{u,v\in V} c_{uv}\, y_u y_v$$

Subject to: y2 = 1, ∀ u ∈ V The condition y2 = 1 simply states that yu ∈ {+1, −1}, ∀ u ∈ V. The Quadratic Program (3.4) is an example of a Strict Quadratic Program, here both the objective function and the conditions are quadratic in the input variables. Such problems are often solved by relaxing the QP into the corresponding Vector Programming problem and then rounding the solution of the VP to get the required solution.

2.3 Vector programming relaxation

To turn a Strict QP into a VP, one replaces each variable yu by an n dimensional vector yu and each product yuyv by the inner product between those vectors, yu • yv. The modularity optimization problem then becomes as follows:

$$\text{Max: } \frac{1}{2}\sum_{u,v\in V} c_{uv}\, y_u y_v$$

Subject to: $y2 = 1, \forall u \in V$

The Vector Program (3.5) is a relaxation of the Strict Quadratic Program (3.4) because any feasible solution to (3.4) yields a solution to (3.5) having the same objective value by assigning the vector (yu, 0, 0...0) to yu. Vector programs are solvable in polynomial time upto any degree of accuracy by the Ellipsoid Algorithm.

2.4 Rounding the solution

Now all that remains is to get from the vector yu to the scalar yu. This process of getting to the solution of the original problem from the solution of the relaxation is called rounding. We could start by using the randomized rounding process of Goemans and Williamson [50] for the MAX-CUT problem. In this we choose a random vector r on the surface of the unit (n − 1) dimensional sphere, and assign yu = sign(yu • r) ∀ u ∈ V . The problem is that unlike in the MAX-CUT case where the weights are all taken to be all non-negative, we can‟t assume the same things about our cuv s, as even if each term cuv yu • yv maybe rounded well by cuv yuyv the sum may not be, because there maybe cancellations.We use Rietz‟ method to round the solution yu ∈ Rn by averaging

over Rn with the normalized Gaussian method. Let g1, g2 ...gn ∼ N (0, 1) be standard independent Gaussian random variables, define $g=\frac{g1,g2,\ldots\ldots,gn}{\sqrt{g1^2+g2^2\cdots\cdots+gn^2}}$ we see that g • g = 1, i.e. it lies on the unit hypersphere and is selected to belong randomly anywhere on the surface of the hypersphere.We then simply take the inner product between the vectors yu and g and assign yu = sign(yu.g)

2.5 Reducing modularity maximization for two communities to the max-cut problem

We reduce the two community modularity maximization problem to the MAX-CUT problem with both positive and negative weights. The MAX-CUT problem is NP-Hard and for the case when all the weights are positive the best known algorithm is the 0.878 approximation algorithm by Goemans and Williamson [50]. Since we do have negative weights the GW algorithm‟s analysis doesn‟t carry over.

We note that Alon et.al. have reduced the MAX-CUT problem to a quantity called the cut-norm of a real matrix A, ||A||C . They give a randomized $\frac{2\ln(1+\sqrt{2})}{\pi}$ 0.56. Algorithm for optimizing ||A||C, thereby leading to an algorithm for Modularity Maximization with the same approximation ratio.

In this section, we will describe the reduction of the Modularity Maximization problem to the MAX-CUT problem, then we briefly describe the cut-norm and another related norm$\|A\|_{\infty\to 1}$, we then describe the reduction of the MAX-CUT problem to the cut-norm problem.In the next section we describe the algorithm for obtaining an approximation guarantee of 0.56.

2.6 Reduction to the MAX-CUT problem

In addition to the above notation, we have the following

- Note that $\sum_{i,j} c_{ij}=0$, where $c_{ij} = 1/2m\,(a_{ij}-\frac{d_i d_j}{2m}$, here $A=(a_{ij})$ is the adjacency matrix of the graph G.
- Let xi = 1 if node i belongs to community 1 and xi = −1 if it belongs to community 2. So that $x2 = 1\ \forall i \in V$. The reduction is as follows:

$$Q=\sum_{i,j\in V} c_{ij}\,\delta\, c_i c_j$$

$$=\sum_{i,j\in V} c_{ij}\frac{(1+x_i x_j)}{2}$$

$$=\frac{1}{2}\sum_{i,j\in V} c_{ij}+\frac{1}{2}\sum_{i,j\in V} c_{ij}(x_i x_j)$$

$$=0+\frac{1}{2}\sum_{i,j\in V} c_{ij}(x_i x_j)\ \text{because}\ \frac{1}{2}\sum_{i,j\in V} c_{ij}=0$$

$$=\frac{1}{2}\sum_{i,j\in V} c_{ij}(x_i x_j)-0$$

$$=\frac{1}{2}\sum_{i,j\in V} c_{ij}(x_i x_j)-\frac{1}{2}\sum_{i,j\in V} c_{ij}\ \text{because}$$

$$\frac{1}{2}\sum_{i,j\in V} c_{ij}=0$$

$$=\frac{1}{2}\sum_{i,j\in V} c_{ij}(x_i x_j-1)$$

$$=\frac{1}{2}\sum_{i,j\in V} -c_{ij}(1-x_i x_j)$$

$$=2\sum_{1\le i<j\le n} -c_{ij}\frac{(1-x_i x_j)}{2}+\sum_{i=j} -c_{ij}\frac{1-x_i^2}{2}$$

$$=2\sum_{1\le i<j\le n} -c_{ij}\frac{(1-x_i x_j)}{2}+\sum_{i=j} -c_{ij}\frac{1-1}{2}\ \text{because}$$

$$x_i^2=1\forall i\in V$$

$$=2\sum_{1\le i<j\le n} -c_{ij}\frac{(1-x_i x_j)}{2}$$

The RHS of the above equation is exactly identical to the mathematical programming formulation of the MAX-CUT problem on the graph with same vertices as G and with weight on the edge (i, j) as (−cij).

Which means that given a graph G = (V, E) if we make another weighted graph G′ =

(V, E′) where wij = (−cij), i.e. the weight of an edge between i and j is −cij then the above can be

written as

$$Q=2\sum_{1\le i<j\le n} -c_{ij}\frac{(1-x_i x_j)}{2}$$

=2MAXCUT (G‟)

Therefore the optimal value of modularity of a bi-partition of G is twice the optimal value of the max-cut of the graph G′ as constructed above. Hence the reduction.

optimizing Q ≤P MAX-CUT

2.7 The cut-norm ||*||C and the ||*|| $\infty \rightarrow$ 1 norm

For a real matrix $A = (aij)_{i\in R, j\in S}$, the *cut-norm* of A is the maximum, over all $I \subset R$, $J \subset S$ of the

quantity $\left| \sum_{i\in I, j\in J} aij \right|$. It is often convenient to study the related norm,

$$\|A\|_{\infty\rightarrow 1} = max \sum_{i\in R} \sum_{j\in S} a_{ij} x_i y_j$$

Where the maximum is taken over all xi, yj ∈ {−1, +1}.We now show that for a matrix A, whose sum of each row and the sum of each column is zero, both the norms are related as

$$\|A\|_C = \frac{\|A\|_{\infty\rightarrow 1}}{4}$$

Suppose that ||A||C = Pi∈I,j∈J aij (similar analysis for the negative case, − Pi∈I,j∈J aij). Define xi = 1 for i ∈ I and xi = −1 for i ∈/ I. Similarly, yj = 1 for j ∈ J and yj = −1 for j ∈/ J . Then we can write,

$$\|A\|_C=\sum_{i,j} a_{ij}\frac{1+x_i}{2}\frac{1+y_i}{2}$$

$$=\sum_{i,j} a_{ij}\frac{1+x_i+y_j+x_i y_j}{4}$$

$$=\sum_{i,j} a_{ij}\frac{1}{4}+\sum_{i,j} a_{ij}\frac{x_i}{4}+\sum_{i,j} a_{ij}\frac{x_i}{4}+\sum_{i,j} a_{ij}\frac{x_i y_j}{4}$$

$=\frac{1}{4}\sum_{i,j} a_{ij}\, x_i y_j$ because each row sum and column sum is zero.

$$=\frac{1}{4}\|A\|_{\infty\rightarrow 1}$$

So, for a matrix A with vanishing row and column sums,

$$\|A\|_{\infty\rightarrow 1}=\|A\|_c$$

2.8 Reduction of the MAX-CUT Problem to the cut-norm

Let G = (V, E) be the given weighted graph with W = (wij) as the weight matrix. Label the vertices and edges arbitrarily V = {v1, v2 ...vn} and E = {e1, e2 ...em}. We describe the construction of a 2m × n matrix M = (mij) with vanishing row and column sums such that

M AXCU T (G) = ||M ||C and therefore by (3.13) M AXCU T (G) = 1 ||M ||∞→1.

The construction of the matrix M is as follows, for each edge ek = (vi, vj) 1 ≤ k ≤ m, if i < j then assign m2k−1,i = m2k,j = wij and m2k,i = m2k−1,j = −wij . The rest of the entries being zeroes. It is easy to see that by the above construction the row and the column sums are zero. It is also easy to see that M AXCU T (G) = ||M ||C ⇒ M AXCU T (G) = 1 ||M ||

∞→1. Asn2m

||M ||∞→1 = max Pi=1 Pj=1 mij xiyj , we observe that in the maximum cut all the vertices on one side of the cut (i.e in the same community) will get the same value of y, and the vertices on the other side of the cut will get the value −y, the values of x adjust themselves accordingly so as to maximize ||M ||∞→1.

Optimizing Q ≤P MAX-CUT ≤P Cut-norm

In the next section we describe the algorithm due to Alon et.al [49] to maximize the || ∗ ||∞→1 norm of a real valued matrix. Therefore the algorithm from the next section to maximize the ||∗ ||∞→1 norm along with the chain of reductions from modularity maximization to the || ∗ ||∞→1 norm supplies us with an algorithm for modularity maximization with the same approximation guarantee.

3.0 Extended Modularity

We have extended the definition of Newman‟s Modularity metric so as to allow overlapping communities; here we present the extension and its basic idea, the sharing function. We also discuss some salient properties of this metric and the sharing function.

3.1 Intuition

Since we want to allow overlaps in the communities we create, we must then have a notion of a vertex belonging to different communities at once. Let G = (V, E) be the graph, and let C be the set of communities and let pc denote the

probability/contribution/strength with which the vertex i belongs to the community c ∈ C. So for each node i we have a *probability vector* $p_i = [p_i^{c_1}, p_i^{c_2}, \ldots\ldots\ldots\ldots p_i^{c_{|c|}}]$ Where $c_1, c_2, c_3 \ldots\ldots\ldots$ $\ldots\ldots\ldots c_{|c|}$ *are the* $|c|$ communities.So basically the strength with which a node I belongs to a particular communities c is denoted by p_i^c For the original modularity case this reduces to the following: If *i* belongs to community *k*, then $p_i^k = 1$ and $p_i^c = 0\ \forall c \in C\backslash\{k\}$.We would require the following from the p^c's:

$$0 \leq p_i^c \leq 1\ \forall i \epsilon V, \forall c \in C \qquad (3.1)$$

$$\sum_{c \in C} p_i^c = 1 \qquad \forall i \epsilon V \qquad (3.2)$$

We could re-write this equation into the following

$$Q = \frac{1}{2m} \sum_{i,j \in V} \left[a_{i,j} - \frac{d_i d_j}{2m}\right] \delta_{c_i c_j}$$

$$Q = \frac{1}{2m} \sum_{i,j \in V} \left[\delta_{c_i c_j} a_{i,j} - \delta_{c_i c_j} \frac{d_i d_j}{2m}\right]$$

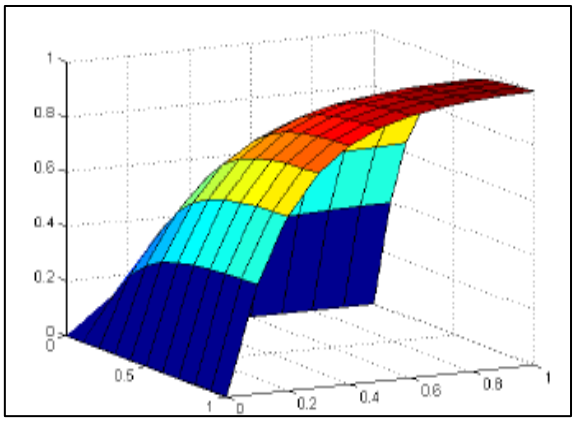
Concentrating on the first term on the right hand side *aij δci, cj*, we see that the *δ* function is the one that leads to an edge contributing either 0 (when *i* and *j* are in different communities) or 1 (when *i* and *j* are in the same community) to a particular community. So the contribution of an edge to a particular community is weighed by the *δ* function.If suppose instead we were to replace this *δci, cj* function by some other suitable function $f^{p_i^c, p_j^c)}$.

3.2 The sharing function f

Ideally we would like the shoaring function f: [0, 1] × [0, 1] → [0, 1] to have the following properties f (0, x) = 0, ∀ x (3.3)

f (1, 1) = 1 (3.4) f(x, The function should have a concave surface and actually be congenial to sharing. Some functions that fit our bill:

1. $f(p1,p2;a) = \frac{(1-e^{-ap1})(1-e^{-ap2})}{(1-e^{-a})^2} : a>0$



2. $f(p1,p2;\ ) = 0;\quad \beta$:

There is yet another function, the two dimensional logistic function, which is considered the smooth version of the delta function, though strictly speaking it does not follow the two properties mentioned above, but it approximately follows them i.e.

f (0, x) ≈ 0, ∀ x

f (1, 1) ≈ 1

The function being:

f (p1, p2; $= \frac{1}{(1+e^{a-2ap1})(1+e^{a-2}}$

Along both the axes its value is approximately 0, and though not apparent in the value at (1, 1) is 0.995. Since this function is also decomposable, we can transform it to satisfy the above properties exactly.

The best results we have got is by using the function f (p1, p2; $^{a)} = \frac{1}{(1+e^{a-2ap1})(1+e^{a-2ap2})}$

With the parameter a = 1. From this point onwards we will simply call this sharing function f.

3.3 Extended modularity

Our extension to modularity for overlapping networks is as follows:

$$Q = \frac{1}{2m} \sum_{c \in C} \sum_{i,j \in V} f(p_i^c, p_j^c) \left[a_{i,j} - \cdot\right] \qquad (3.6)$$

For weighted graphs this naturally generalizes to

$$Q = \frac{1}{2wm} \sum_{c \in C} \sum_{i,j \in V} f(p_i^c, p_j^c) \left[w_{i,j} - \frac{w d_i}{2r}\right] (3.7)$$

Where, wij is the weight on the edge (i,j), $w d_i = \sum_{k=1}^{n} w_{ik}$ and $2wm = \sum_{i=1}^{n} w_{di}$

4.0 Algorithm to Maximize Extended Modularity

We have two algorithms that try to maximize modularity, one of them OSG (One Step Greedy), roughly finds the overlapping configuration, subsequent incremental improvement can be done on this initial configuration. The other is basically a formulation of overlapping modularity optimization as a Genetic Optimization problem.

4.1 One step greedy algorithm: osg

4.1.1 Terminology

A node i is called a boundary node if it has edges going into any community apart from it‟s own community, and let the set Ci denote all those communities to which the node i has links to, therefore any node on the boundary will have $|Ci| \geq 2$. Sharing a node i between a set C of communities in this case means $pc = 1/|C| \; \forall \, c \in C$.

4.1.2 Idea

The idea is simple, using the original definition of modularity, the one for the non-overlapping case, we first hard-partition the graph into communities. Then for each of the boundary nodes we try to share it with the communities it is bordering, we then select the node that gives the largest increase in modularity and add it into that community and iterate. We do this till there is no more increase in the modularity.

4.1.3 Criticism

Basically the algorithm is a quick and dirty guide to what all overlaps can be there; it is na¨ıve in the sense that even if a node can belong overwhelmingly to one or the other community even while being shared, it will contribute equal weights to both the communities in which it is shared. This is the main drawback, but it does serve it‟s purpose, that of showing approximately where the sharing is taking place.

4.2 Genetic algorithm: GA

The Genetic Algorithms are adaptive search heuristic based algorithms, these aims to simulate the natural process of evolution and natural selection to produce the „fittest‟ individuals. We transform the modularity maximization problem as a Genetic Optimization problem in which the number of communities can be given as a parameter.

4.2.1 Introduction to GAs

Genetic Algorithms aim to mimic what nature seems to be doing in real life to produce the best and the fittest individuals, i.e. the axioms of survival of the fittest, crossover and genetic mutation, natural selection etc.

Basically any genetic algorithm will have the following two things

1. A representation of the solution space.
2. A fitness function to evaluate a particular solution.

A particular candidate solution is called an individual, and the genetic material of each individual is called the chromosome. These individuals are evaluated on the basis of the fitness function; the fitness function is the quantity that we are aiming to optimize. The algorithm applies various search heuristics to come up with an optimal solution. Typically the heuristics are as follows:

1. Initialization: The individuals are initialized randomly to form an initial population this can range between a few hundred to a few thousand individuals.
2. Selection: A chosen group of individuals is used to breed the next generation of the population. Usually it is the fittest individuals of the previous generation who are selected to breed and form the next generation. The individuals who are not deemed to be fit are neglected in this step.
3. Reproduction: Genetic Operators like Crossover and Mutation are applied to make up for the individuals that we discarded in the above step.

(a) Crossover: Here basically two individuals from the previous generation come to- gether and via some operation form a new individual. Some of the balance elements required to complete the population are made in this way.

(b) From Scratch: The rest of the individuals to complete the population are made brand new from scratch.

(c) Mutation: Here we take an existing individual and change it"s chromosome. We do this so as to improve our chances of finding the global optimum.

4. Termination: Some criteria is kept, to terminate the solution, like a fixed number of iterations (epochs), or some other cost measure is used.

4.2.2 Chromosome representation

As the name suggests, we make the rest of the individuals from scratch. Normalization should also be done.

4.2.4.5 Mutation

This depends on the exact problem at hand. Given a graph G = (V, E) and the set of communities C we are trying to get for each i ∈ V the set of belonging probabilities pc ∀c ∈ C. So our chromosome looks like a |V | × |C| matrix I. Where Iij = pj.

4.2.3 Fitness function

In our case it is nothing but the overlapping modularity,

$$Q=\frac{1}{2m}\sum_{c\in C}\sum_{i,j\in V} f(p_i^c, p_j^c)\,[a_{i,j}-\frac{d_i d_j}{2m}] \qquad (4.1)$$

4.2.4 Algorithm

The algorithm takes the following three parameters, nindiv, ncomms, and nepocs. Where nindiv are the number of individuals that we initialize randomly, ncomms is a parameter for the number of communities, and nepochs is the number of iterations which we allow the algorithm to run before we report the solution.

4.2.4.1 Intialization

All the chromosomes of all the individuals are initialized to random variables. We have to normalize those to ensure that the probabilities for each node sum upto 1 i.e.

$$\sum_{k=1}^{n_{comms}} p_i^{ck} 1 \qquad (4.2)$$

For this we divide each entry by I_{ij} the quantity $\sum_j I_{ij}$, thus ensuring that now each is I_{ij} between 0 and 1.

4.2.4.2 Selection

All the individuals are now ordered on the basis of the fitness of their chromosome; here we have selected a fixed percentage of individuals to carry forward to the next generation. The rest of the individuals are made up partly by crossover from the better individuals of the previous generation, while the rest are made from scratch.

4.2.4.3 Crossover

Here we take two individuals I_1 and I_2 and we randomly pick a community j. Now we make a third individual I_3 which is a copy of I_2 but now we replace the column I_3 (: j) by the column I_1(: j) i.e. we just change the probability vector of community j in I_2 thereby making a new individual and keeping I_1 , I_2 unchanged. Need to keep in mind that we would still need to normalize the rows in I_3.

4.2.4.4 From scratch

For a fixed percentage of individuals, we select a node and a community at random and change the belonging factor of that individual for that particular node and community, keeping in mind to keep normalizing.

4.2.4.6 Clean up

In addition to the above „mandatory" steps, Nicosia et.al. implement a cleanup function, it basically sees the following two parameters for a randomly selected node, i and a randomly selected community c of every individual,

$$1.\text{avgNeigh}(i,c)=\frac{\sum_{k\in Neighbourhood(i)} p_k^c}{|Neighbourhood(i)|} \qquad (4.3)$$

$$2.\text{avgN otNeight}(I,c)=\frac{\sum_{k\in Neighbourhood(i)} p_k^c}{n-|Neighbourhood(i)|} \qquad (4.4)$$

If avgN eigh(i, c) ≥ avgN otN eigh(i, c) then we increase pc by a small quantity p, that we call the momentum.Else we decrease the same quantity by the momentum, if pc < p then we set it to 0.We have presented and analyzed the Original Modularity Optimization problem as a Math- ematical Programming problem; we have also reduced the problem of partitioning a graph into two communities

to maximize modularity to the MAX-CUT problem and presented three approximation algorithms to maximize modularity.

We see that the extended modularity based method correctly groups the nodes into com- munities which are either known apriori, or goes one step

better and discovers a hidden structure in the network, also that the sharing is pretty intuitive with the nodes which have an even degree distribution between communities coming out as shared. The extent of sharedness and allowed overlap can be controlled by varying the parameter in the sharing function.

Though Modularity inherently has some problems regarding the resolution limit, we haven‟t encountered it in our real-life examples that we have tested upon.The extension to modularity is simple, logical and straightforward, and we have given general characteristics for the sharing function, so as to keep the door open for further improvements in that area. It also reduces to the original modularity function in case of a hard-partition and follows its salient features.

The Genetic Algorithm based approach takes $O(n2kei)$ time where n is the number of nodes, k the number of communities to be found, e number of epochs and i the number of individuals to be generated. This is a heavy on space too, requiring nki floating numbers to be stored. We would like to have approximation algorithms based on Mathematical Programming to optimize extended modularity, but even drastic simplifications of our extended modularity have resisted being formulated as tractable mathematical programs.

5.0 Conclusion

We have presented and analyzed the Original Modularity Optimization problem as a Math- ematical Programming problem; we have also reduced the problem of partitioning a graph into two communities to maximize modularity to the MAX-CUT problem and presented three approximation algorithms to maximize modularity.We see that the extended modularity based method correctly groups the nodes into com- munities which are either known apriori, or goes one step better and discovers a hidden structure in the network, also that the sharing is pretty intuitive with the nodes which have an even degree distribution between communities coming out as shared. The extent of sharedness and allowed overlap can be controlled by varying the parameter in the sharing function.Though Modularity inherently has some problems regarding the resolution limit, we haven‟t encountered it in our real-life examples that we have tested upon.The extension to modularity is simple, logical and straightforward, and we have given general characteristics for the sharing function, so as to keep the door open for further improvements in that area. It also reduces to the original modularity function in case of a hard-partition and follows its salient features.

The Genetic Algorithm based approach takes O(n2kei) time where n is the number of nodes, k the number of communities to be found, e number of epochs and i the number of individuals to be generated. This is a heavy on space too, requiring nki floating numbers to be stored. We would like to have approximation algorithms based on Mathematical Programming to optimize extended modularity, but even drastic simplifications of our extended modularity have resisted being formulated as tractable mathematical programs.

References

[1] D. J. Watts, S. H. Strogatz, Collective dynamics of small-world networks, Nature, 393(6684), 440-442, 1998

[2] M. Girvan, M. E. Newman, Community structure in social and biological networks, Proceedings of the National Academy of Sciences of the United States of America, 99(12), 2002, 7821-7826

[3] Detecting community structure in networks, M. E. J. Newman, Eur. Phys. J. B 38,321-330, 2004

[4] Using Correspondence Analysis for Joint Displays for Affiliation Networks. Models and Methods in Social Network Analysis. K Faust, Chapter 7 (Cambridge University Press, New York, 2005).

[5] J. Scott, Social Network Analysis: A Handbook. Sage, London, 2nd edition.

[6] F. Radicchi, C. Castellano, F. Cecconi, V. Loreto, D. Parisi, Defining and identifying communities in networks. Proc. Natl. Acad. Sci. USA 101, 2658-2663, 2004

[7] G. W. Flake, S. R. Lawrence, C. L. Giles, F. M. Coetzee, Self-organization and identification of Web communities. IEEE Computer 35, 66-71, 2002

[8] L. Danon, J. Duch, A. Diaz-Guilera, A. Arenas, 2005, J. Stat. Mech., P09008

[9] Jeong, H., Tombor, B., Albert, R., Oltvai, Z. N.and Barabasi, A.-L. (2000) Nature(London) 407, 651-654.

[10] D. A. Fell, A. Wagner, 2000, Nat. Biotechnol. 18, 1121-1122