Recovering Communities in Networks

Sam Rosen *

December 2019

Abstract

Modularity is a measure of the quality of community assignments in networks. We attempt a simulated annealing strategy focused on fast calculations of modularity. The results achieved for trivial cases are promising, but the algorithm fails to scale well. After noticing where pitfalls occur, we can easily produce cases where the algorithm fails greatly. We conclude the project was a strong learning experience.

1 Introduction

A graph or network is a collection of nodes that connect to other nodes via edges. They are commonly used in algorithm classes and applications are numerous. Graphs are used to model phenomena such as social networks, infrastructure, and biological processes. Graphs can be represented as adjacency matrices. An adjacency matrix has entry $A_{i,j} = w_{i,j}$ where the edge weight $w_{i,j} > 0$ if and only if there is an edge between nodes i and j. In our problem we will consider simple undirected graphs so $A_{i,j} = A_{j,i}$ and $w_{i,j} \in \{0, 1\}$.

A node is a neighbor to another node if there is an edge between them. A community or cluster in a graph is a collection of nodes where each node has a similar set of neighbors to other nodes in the community. A community could be a group of friends in a social network, a highly tangled road system, or a complex coupled system. The problem is ill-defined, but typically when finding clusters, the quality is determined by how many edges from nodes in the community go to nodes outside the community (inter-community edges) versus the amount of edges from nodes in the community to other nodes in the community (intra-community edges). An ideal community assignment has many edges between its members, and minimal going to other communities.

Modularity is a popular metric for measuring the quality of community assignments. It is well-studied and is often used to recover communities by maximizing it [2]. Although globally maximizing modularity is an NP-HARD problem, good community assignments can be found with sub-optimal values.

*SamRosen@umass.edu

Modularity scales well due to its local properties [2]. The Louvain algorithm was able to create strong community assignments graph with well-above 1 million nodes [1]. Although modularity generalizes to weighed and directed graphs, we will use the below definition for a simple undirected graph $G = (V, E)$:

$$Q(\mu) = \frac{1}{2m} \sum_{i,j} \left(A_{i,j} - \frac{k_i k_j}{2m} \right) \delta(\mu_i, \mu_j)$$

m : number of edges in $G, |E|$

$A_{i,j}$: entry (i, j) for adjacency matrix of G

k_i : degree of node i

$\delta(x, y)$: Dirac delta function; $\delta(x, y) = 1 \iff x = y, 0$ otherwise

μ_i : The community assignment for node i (a node can belong to only one community)

Because $\delta(\mu_i, \mu_j) = 0$ if nodes i and j are in separate communities, we only need to evaluate modularity over each community. For maximization, ideally the largest possible communities are assigned so the Dirac delta function is positive many times. However, if we are erroneous with our assignment, we will penalize modularity by placing two nodes that are not connected ($A_{i,j} = 0$) in the same cluster by evaluating $A_{i,j} - \frac{k_i k_j}{2m}$ as a negative number. Modularity derives its robustness from the ratio $\frac{k_i k_j}{2m}$ which balances the clout of nodes with high degree. The ratio causes two nodes of high degree to greatly penalize modularity if they are placed in the same community but do not have an edge between each other. Moreover, it also dampens the effect if there is an edge between them, since their high degree could skew the communities to simply be the neighbors of nodes with high degree.

The stochastic block model (SBM) is a random graph model with parameters, $S \in \mathbb{N}^k$, and $P \in [0, 1]^{k \times k}$. The model produces k blocks with $|C_1| = S_1, |C_2| = S_2, ..., |C_k| = S_k$. The probability of an edge between two nodes $x \in C_i$ and $y \in C_j$ is $P_{i,j}$. We will use networkx [4] to generate SBM's and handle all other graph functionality. By maximizing modularity, we hope to recover the parameters and communities for a generated stochastic block model.

2 Problem Formulation

Consider a graph $G = (V, E)$ with adjacency matrix $A \in \{0, 1\}^{N \times N}$ where $|V| = N$. Node i and node j have an edge between each other if and only if $A_{i,j} = 1$. We will consider simple undirected graphs, so $A_{i,j} = A_{j,i}$.

Let $[x] = \{1, 2, ..., x\}, x \in \mathbb{N}$. A clustering vector $\mu \in [N]^N$ with entry $\mu_i = j$ signifies node i is in a cluster with node j as the centroid. Although the centroid has no meaning in this definition of community, this definition allows us to avoid specifying the number of communities and fitting a clustering matrix. Two nodes i, j are in the same cluster if and only if they have the same centroid ($\mu_i = \mu_j$).

Our optimization problem is as follows:

$$\underset{\mu \in [N]^N}{\text{maximize}} \quad \frac{1}{2m} \sum_{i,j} \left(A_{i,j} - \frac{k_i k_j}{2m} \right) \delta(\mu_i, \mu_j)$$

The state space of the problem is size N^N. It is unlikely we will find a global maximum, but there is no guarantee a perfect recovery of the blocks will correspond to a maximum. After maximizing Q we can generate parameters from the communities encoded in μ. S will be the size of the communities, and $P_{i,j}$ will be the density of edges between nodes in community i and community j.

3 Simulated Annealing

Simulated annealing is a method for optimization problems with large state spaces. [5] is a strong introduction to simulated annealing that assisted in implementing our solution. The write-up summarizes simulated annealing in 5 steps:

1. Set an initial temperature $T = T_0$ and choose a valid initial candidate in the state space to be the current best solution.
2. Pick a random "neighboring" candidate in the state space.
3. Accept the neighboring candidate if it is an improvement over the current best solution.
4. If the neighboring candidate is worse choose it to be the optimal candidate based off some probability. [6] uses the probability $\exp(\frac{\Delta}{T})$ where Δ is the difference between the optimal solution and the worse candidate and T is the current temperature.
5. Lower the temperature and go back to step 3, repeating many times. [6] recommends decreasing the temperature in an exponential decaying manner and finishing once convergence is achieved.

Most of these steps are vague since they must be defined in the context of the problem. For example, the range of the function to optimize could be 1, so a large initial temperature ($\exp(\frac{\Delta}{T}) \approx 1$) will cause a large portion of the iterations to simply almost always go to the random neighboring candidate, wasting processing time from no clever navigation of the state space. Choosing a neighboring candidate is where most simulated annealing problems vary. A good heuristic for defining neighboring candidates is to map the state space to a graph, where nodes are states with edges to neighboring states; a strong

definition should create a state space graph with a small diameter, so any state is not too many steps away from an optimal solution [7].

Using simulated annealing to maximize modularity has had previous success. [2] cites multiple successful cases, but notes [3] performed best by doing a divide-and-conquer strategy for choosing neighboring candidates. Other strategies noted for avoiding local minima include merging or splitting communities, and reassigning multiple nodes at a time.

4 Solution

For our solution we deviate from the standard steps for simulated annealing:

1. Instead of returning the current optimal candidate after completing the iterations, we maintain a single global optimal candidate, the candidate that had the highest modularity during the iterations. We do this since the objective function varies wildly over the iterations.

2. After returning our solution, we "smooth" it to potentially remove communities of a single node.

Algorithm 1 Simulated Annealing for Maximizing Modularity

$G = (V, E);\ r \in (0, 1); T_0 > 1$:
procedure SA(G, r, T_0)
    $\mu \leftarrow [N]$ ▷ Initial valid candidate
    $\mu^* \leftarrow \mu$ ▷ Global candidate tracking
    $T \leftarrow T_0$
    while $T > 1$ **do**
        $\mu' = \mu$
        Choose a random node i
        if $U[0, 1] < .5$ **then** ▷ Generate random number in $[0, 1]$
            $\mu'_i = i$ ▷ Set a node's centroid equal to itself
        else
            for each neighbor j of i **do**
                $\mu'_j = \mu_i$ ▷ Set all of a node's neighbors to be in its cluster
        Calculate $\Delta Q = Q(\mu') - Q(\mu)$
        if $Q(\mu') > Q(\mu^*)$ **then**
            $\mu^* \leftarrow \mu'$
            $\mu \leftarrow \mu'$
        else
            if $U[0, 1] < \min\left(\exp\left(\frac{\Delta Q}{T}\right), 1\right)$ **then**
                $\mu \leftarrow \mu'$
        $T \leftarrow T \cdot (1 - r)$
    return μ^*

Because we occasionally set a random node to be its own centroid, returned solutions often have several communities of a single node. However, some of these single node communities have a majority of neighbors in another community. After the simulated annealing process, the solution is smoothed: we set these singleton communities to be in the majority community (if it exists) of the node's neighbors.

Algorithm 2 Smoothing Community Assignments

$G = (V, E)$; $\mu \in [N]^N$; $\rho \in [.5, 1]$:
procedure SMOOTH(G, μ, ρ)
    for each community with a single node i **do**
        if ρ% or higher of neighbors of i are in community j **then**
            $\mu_i = j$
    if modularity has increased **then**
        return smoothed μ
    else
        return original μ

Speeding up the process

One can speed up the modularity calculations by storing the RHS as a matrix:

$$A_{i,j} - \frac{k_i k_j}{2m} = \left(A - \frac{1}{2m} \deg(G) \cdot \deg(G)^T \right)_{i,j}$$

Additionally the modularity function ($O(N^2)$ calculations) only needs to evaluated once in the initialization. Change in modularity can be found in $O(N)$ calculations by first calculating the change of removing a node from a community and then adding the change from setting that node to a new community. Let $\mu_{i \to j}$ be the resulting clustering vector from changing node i to have centroid j (setting $\mu_i = j$). We then have the change in modularity by removing a node i from a community is:

$$\Delta Q^-(i; \mu) = -\frac{1}{2m} \sum_l \left(A_{i,l} - \frac{k_i k_l}{2m} \right) \delta(\mu_i, \mu_l) - \frac{1}{2m} \sum_l \left(A_{l,i} - \frac{k_l k_i}{2m} \right) \delta(\mu_l, \mu_i)$$

The change in modularity from adding node i to community j:

$$\Delta Q^+(i \to j; \mu) = \frac{1}{2m} \sum_l \left(A_{i,l} - \frac{k_i k_l}{2m} \right) \delta(j, \mu_l) + \frac{1}{2m} \sum_l \left(A_{l,i} - \frac{k_l k_i}{2m} \right) \delta(\mu_l, j)$$

Since we are handling undirected graphs these can be simplified to:

$$\Delta Q^-(i; \mu) = -\frac{1}{m} \sum_l \left(A_{i,l} - \frac{k_i k_l}{2m} \right) \delta(\mu_i, \mu_l)$$

$$\Delta Q^{+}(i \to j; \mu) = \frac{1}{m} \sum_{l} \left(A_{i,l} - \frac{k_i k_l}{2m} \right) \delta(j, \mu_l)$$

Then the overall change in modularity is:

$$Q(\mu_{i \to j}) = Q(\mu) + \Delta Q^{-}(i; \mu) + \Delta Q^{+}(i \to j; \mu)$$

$$Q(\mu_{i \to j}) = Q(\mu) + \frac{1}{m} \sum_{l} \left(A_{i,l} - \frac{k_i k_l}{2m} \right) \big(\delta(j, \mu_l) - \delta(\mu_i, \mu_l) \big)$$

These calculations can be easily vectorized. After making these improvements we were able to achieve reasonable speed for larger graphs (≈ 5000 nodes).

5 Results

We achieve good results on trivial examples and SBM's with a small number of clusters. However, once we generate SBM's with a large amount of clusters our algorithm deviates greatly from the true size of the communities. Our solution seems to prioritize creating large communities with many small communities, making it a poor choice for a network with many communities of similar size.

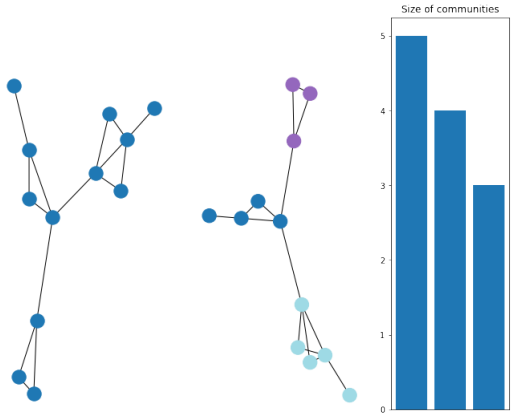
All processing was done on a standard Google Colab notebook. Our implementation is written with speed in mind so despite modest hardware it achieves decent speeds. Every result was run with the following parameters:

$\rho : .5,$ We pick a simple majority for smoothing results

$T_0 : 5,$ Modularity is bounded above by 1, so we do about 5 times its possible max

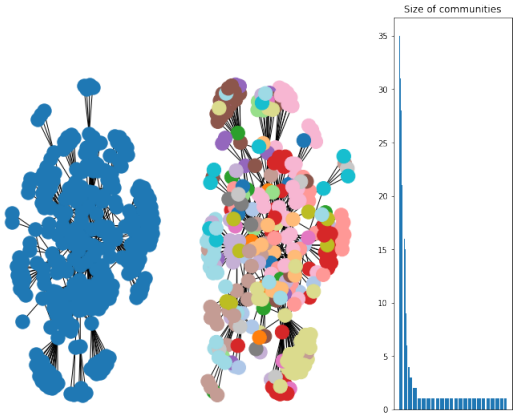
$r : .00003,$ A very small cooling rate which will give about 50000 iterations for each problem.

The cooling rate is incredibly over-conservative for the easier problems as they converge in $< 10^3$ iterations. However, the large cooling rate is reasonable for the large problems as improvement typically stops before 30000 iterations. Because we do not have any convergence criteria, each trial runs for the same number of iterations. An implementation of our solution and copy of these results can be found here. An ideal Q is the modularity calculated if every node was properly classified. Below is a sequence of runs with the algorithm explaining various results:



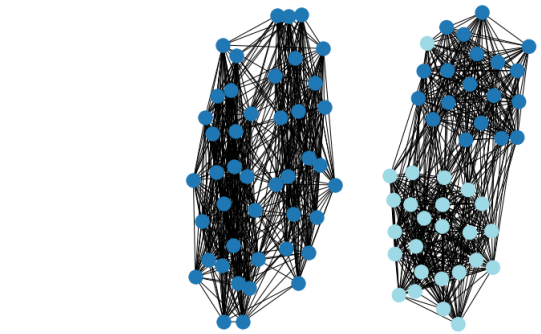
Nodes	Edges	Runtime (seconds)	Ideal Q	Solution Q
12	15	3.098	0.5133	0.5133

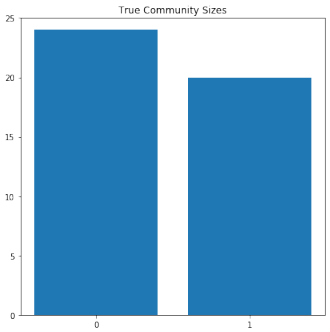
On this tiny graph used for initial testing we achieve the clustering the gives the maximum modularity.

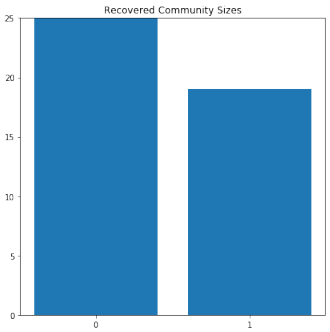


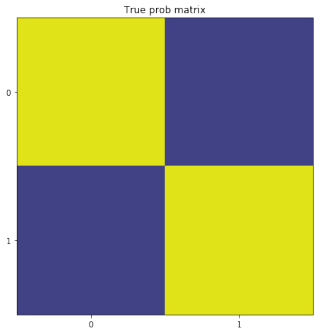
Nodes	Edges	Runtime (seconds)	Ideal Q	Solution Q
300	427	5.545	N/A	0.3288

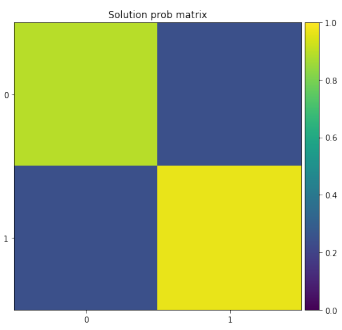
This randomly generated graph is meant to resemble the internet with several large hubs that connect to smaller neighborhoods with few nodes [4]. We achieve a community size distribution that matches this with several large communities and many small ones.





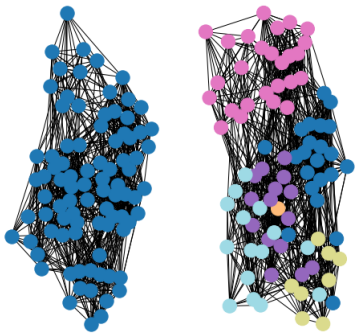


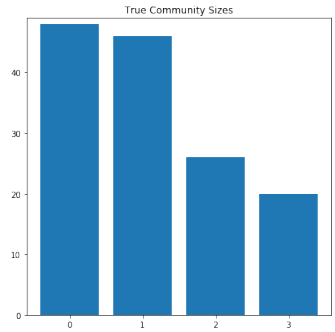


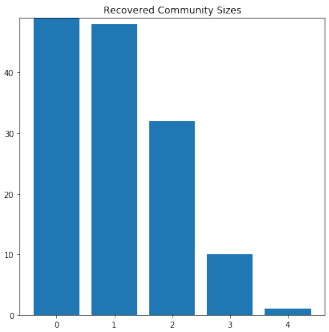


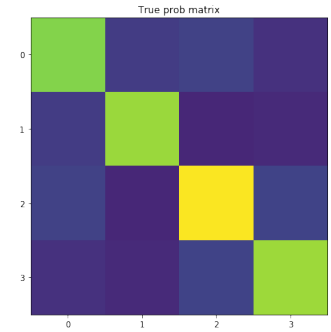
Nodes	Edges	Runtime (seconds)	Ideal Q	Solution Q
44	549	9.2588	0.3029	0.2717

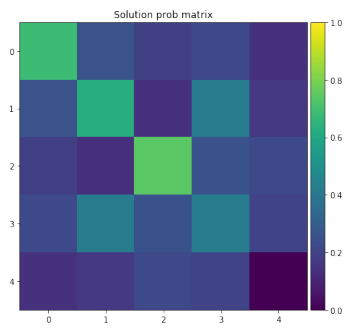
On this example we try a simple SBM with 2 communities and little overlap. We manage to recover the communities almost completely and get a strong estimate of the true probability matrix, however, we still do not get a perfect recovery for an easy example.





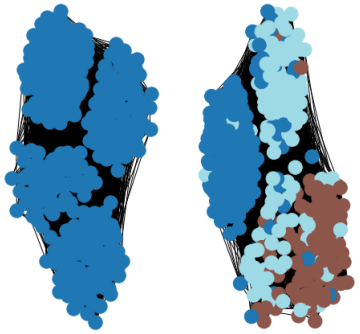


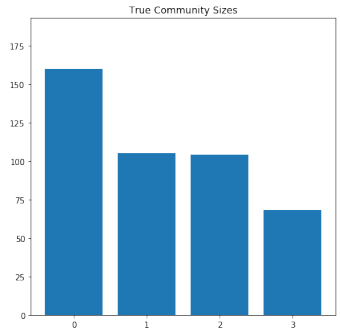


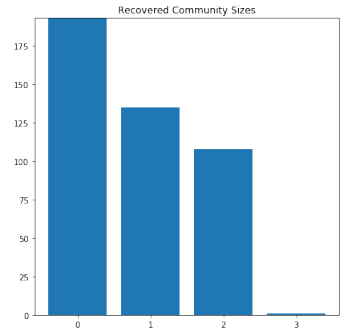


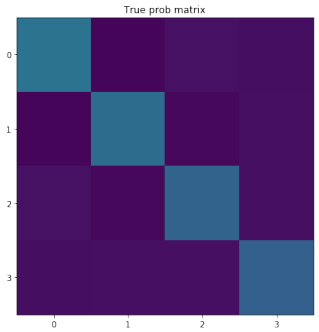
Nodes	Edges	Runtime (seconds)	Ideal Q	Solution Q
97	991	10.1179	0.4293	0.3139

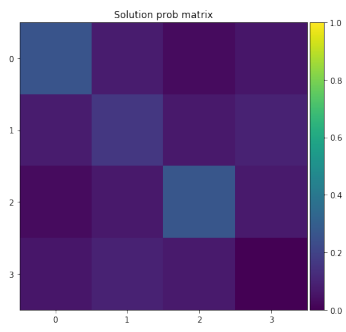
On a more coupled SBM with 4 communities we manage to approximately recover the 4 communities and classify a single node that has particularly high overlap as its own community. However the approximate communities do not give a strong estimate of the original probability matrix.





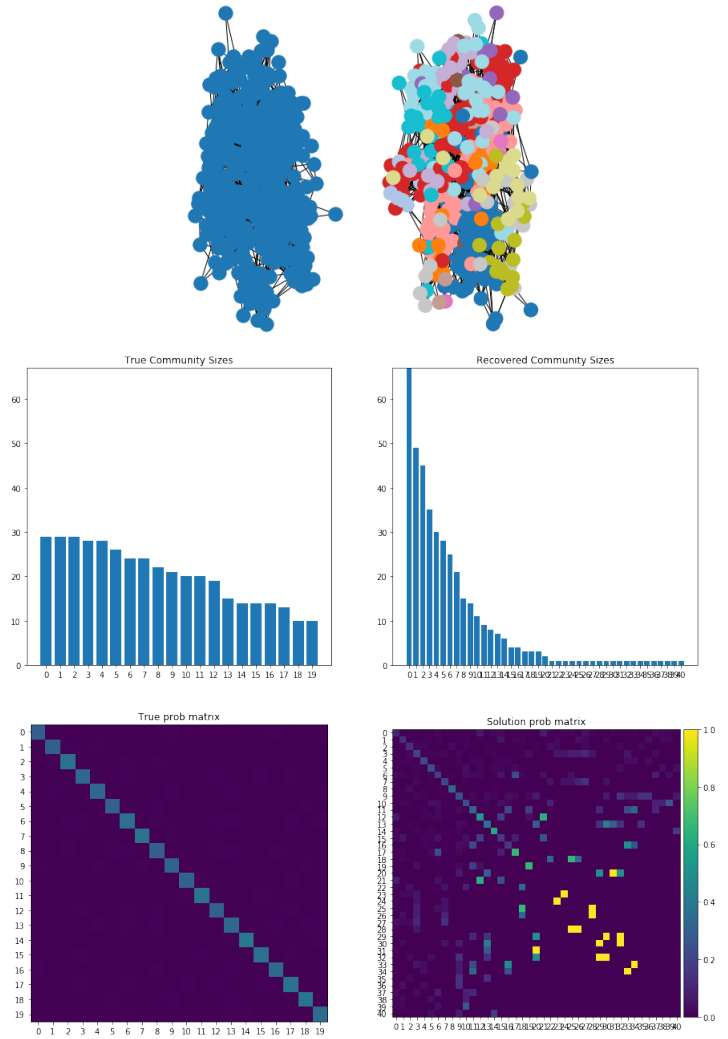






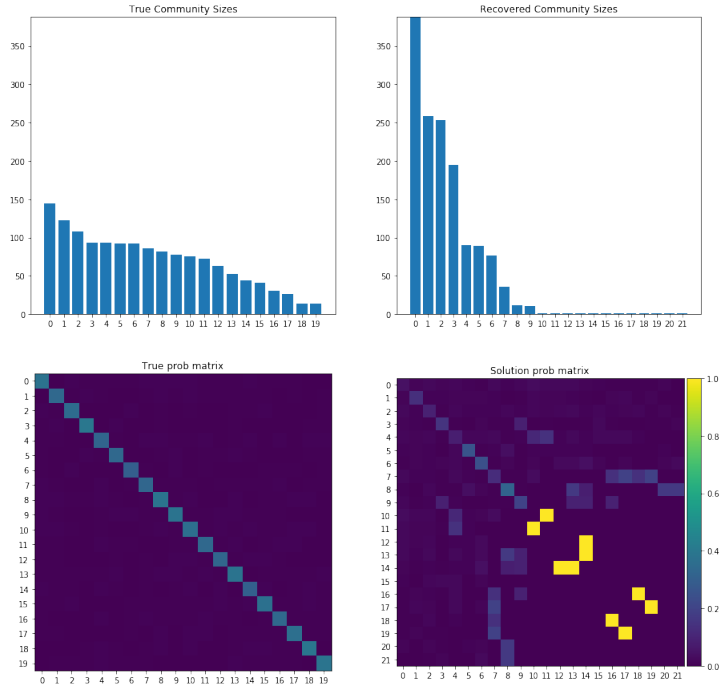
Nodes	Edges	Runtime (seconds)	Ideal Q	Solution Q
437	11378	24.2063	0.4720	0.2891

On a sparse SBM with only 4 communities we only detect 3 communities of notable size. The communities are of reasonable size, but our estimate of the probability matrix is mediocre.



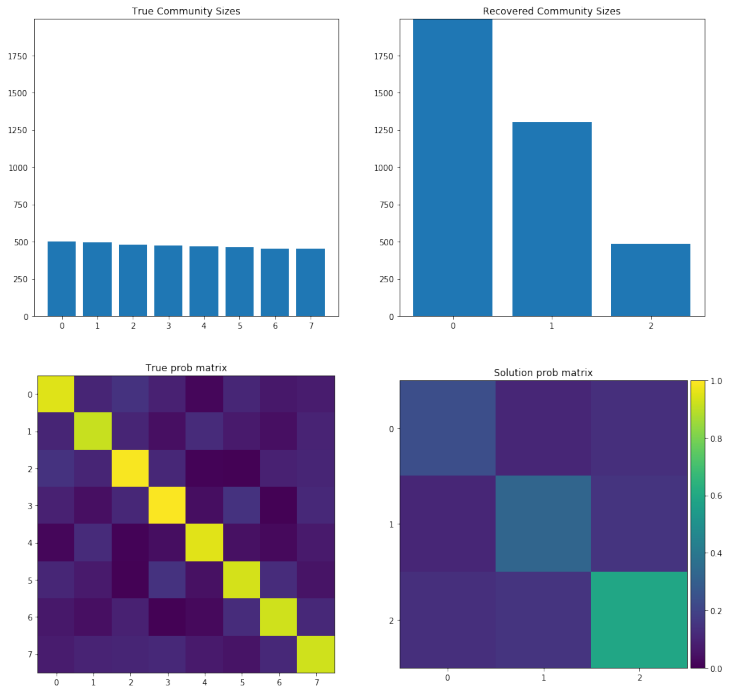
Nodes	Edges	Runtime (seconds)	Ideal Q	Solution Q
409	1855	8.2613	0.7449	0.4580

On a somewhat sparse SBM with a large amount of relatively uniform community sizes, our algorithm seems to prioritize creating large communities and struggles to retrieve the true community size distribution. We can see by the ideal modularity that we struggle to truly maximize modularity and remain stuck in a local maximum.



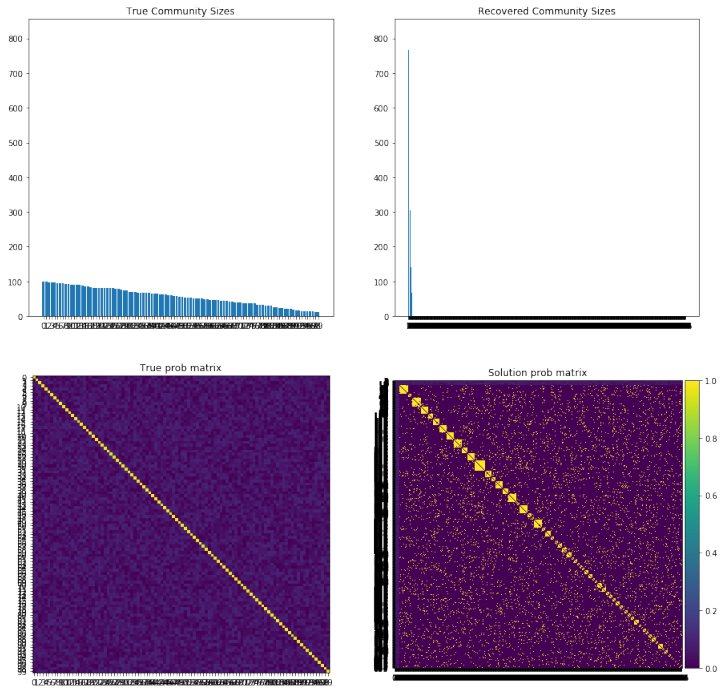
Nodes	Edges	Runtime (seconds)	Ideal Q	Solution Q
1421	26891	30.8627	0.7327	0.4045

(Graph not pictured due to lack of clear structure) On another sparse SBM with a more variation in the true community size, we recover several very large communities and few communities of the correct size. As a result our recovery of the probability matrix falters.



Nodes	Edges	Runtime (seconds)	Ideal Q	Solution Q
3779	1327070	662.6589	0.5124	0.2152

(Graph not pictured due to lack of clear structure) On an SBM with strong density, we fail to get reasonably close to the number of communities or their size. Instead we predict a relatively sparse SBM with 3 communities. Our Q is still a respectable $\approx .21$, but it is far from the ideal value of $\approx .51$.



Nodes	Edges	Runtime (seconds)	Ideal Q	Solution Q
5636	984602	505.457	0.1753	0.02407

(Graph not pictured due to lack of clear structure) After looking at a few results we can easily produce a case our algorithm struggles on. We create an SBM with 100 blocks ranging from size 10 to 100 nodes. Our algorithm discerns a network with few giant communities and about 800 communities of only a single node.

6 Conclusion

Overall we conclude our algorithm performs well on relatively small graphs with little coupling of communities. It struggles with local maxima on larger graphs but seems to scale well for internet-like graphs that typically contain several large hubs and many small communities. It may have some utility for performing many quick clustering jobs on graphs with less than 1000 nodes. We fail to improve over leading methods for maximizing modularity such as [1] and [3] as these implementations are just as fast and accurate. The modularity problem was a good introduction to simulated annealing, and shows that a stronger simulated annealing algorithm could be successful. Although our results were not fantastic, the code is well-curated and computes modularity quickly over many iterations.

References

[1] Vincent D. Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2008(10):10008, Oct 2008.

[2] Mingming Chen, Konstantin Kuzmin, and Boleslaw K. Szymanski. Community detection via maximization of modularity and its variants. *IEEE Transactions on Computational Social Systems*, 1(1):46–65, Mar 2014.

[3] Roger Guimerà and Luís A. Nunes Amaral. Functional cartography of complex metabolic networks. *Nature*, 433(7028):895–900, Feb 2005.

[4] Aric A. Hagberg, Daniel A. Schult, and Pieter J. Swart. Exploring network structure, dynamics, and function using networkx. In Gaël Varoquaux, Travis Vaught, and Jarrod Millman, editors, *Proceedings of the 7th Python in Science Conference*, pages 11 – 15, Pasadena, CA USA, 2008.

[5] Todd Schneider. The traveling salesman with simulated annealing, r, and shiny, Sep 2014. https://toddwschneider.com/posts/traveling-salesman-with-simulated-annealing-r-and-shiny/.

[6] John Walker. Simulated annealing: The travelling salesman problem, June 2018. https://www.fourmilab.ch/documents/travelling/anneal/.

[7] Wikipedia contributors. Simulated annealing wikipedia, the free encyclopedia, 2019. [Online; accessed 1-December-2019].