CS224W: Analysis of Networks **Fall 2017**

Problem Set 4

Due 11:59pm PDT November 30, 2017

General Instructions

These questions require thought, but do not require long answers. Please be as concise as possible. You are allowed to take a maximum of 1 late period (see the information sheet at the end of this document for the definition of a late period).

Submission instructions: You should submit your answers via GradeScope and your code via the SNAP submission site.

Submitting answers: Prepare answers to your homework in a single PDF file and submit it via GradeScope. The number of the question should be at the top of each page. We recommend that you use the submission template latex file to prepare your submission.

Fill out the information sheet located at the end of this problem set or at the end of the template file and sign it in order to acknowledge the Honor Code (if typesetting the homework, you may type your name instead of signing). This should be the last page of your submission. Failure to fill out the information sheet will result in a reduction of 2 points from your homework score.

Submitting code: Upload your code at `http://snap.stanford.edu/submit`. Put all the code for a single question into a single file and upload it.

Homework survey: After submitting your homework, please fill out the Homework 4 Feedback Form. Respondents will be awarded extra credit.

Questions

1 Variations on a Theme of PageRank [25 points – Silviana, Poorvi]

1.1 Personalized PageRank I [7 points]

Personalizing PageRank is a very important real-world problem: different users find different pages relevant, so search engines can provide better results if they tailor their page relevance estimates to the users they are serving. Recall from class that PageRank can be specialized with clever modifications of the teleport vector. In this question, we will explore how this can be applied to personalize the PageRank algorithm.

Assume that people's interests are represented by a set of representative pages. For example, if Zuzanna is interested in sports and food, then we could represent her interests with the set of pages {`www.espn.com`, `www.epicurious.com`}. For notational convenience, we will use integers as names for webpages.

Suppose you have already computed the personalized PageRank vectors for the following users:

- Agatha, whose interests are represented by the teleport set $\{1, 2, 3\}$
- Bertha, whose interests are represented by the teleport set $\{3, 4, 5\}$
- Clementine, whose interests are represented by the teleport set $\{1, 4, 5\}$

- DeShawn, whose interests are represented by the teleport set $\{1\}$

Assume that the weights for each node in a teleport set are uniform. Without looking at the graph, can you compute the personalized PageRank vectors for the following users? If so, how? If not, why not? Assume a fixed teleport parameter β.

i. **[2 points]** Eloise, whose interests are represented by the teleport set $\{2\}$.

ii. **[2 points]** Felicity, whose interests are represented by the teleport set $\{5\}$.

iii. **[3 points]** Glynnis, whose interests are represented by the teleport set $\{1, 2, 3, 4, 5\}$ with weights 0.1, 0.2, 0.3, 0.2, 0.2, respectively.

1.2 Personalized PageRank II [3 points]

Suppose that you've already computed the personalized PageRank vectors of a set of users (denote the computed vectors V). What is the set of all personalized PageRank vectors that you can compute from V without accessing the web graph?

1.3 Spam farms I [5 points]

The staggering number of people who use search engines to find information every day makes having a high PageRank score a valuable asset, which creates an incentive for people to game the system and artificially inflate their website's PageRank score. Since the PageRank algorithm is based on link structure, many PageRank spam attacks use special network configurations to inflate a target page's PageRank score. We will explore these configurations, called *spam farms*, in this part of the question.

Consider the spam farm shown in Figure 1. The spammer controls a set of *boosting pages* $1, \ldots, k$ (where page i has PageRank score p_i) and is trying to increase the PageRank of the *target page* 0. The target page receives λ amount of PageRank from the rest of the graph (represented by the dotted arrow in Figure 1). This means that $\lambda = \sum_{i \in S} \frac{r_i}{d_i}$, where S is the set of nodes in the rest of the network that link to the target page 0, r_i is the PageRank score of node $i \in S$, and d_i is the outdegree of node $i \in S$. Let N denote the number of pages in the entire web, including the boosting pages and target page. Calculate the PageRank p_0 of the target page with this configuration as a function of λ, k, β, and N. Your solution should not include other parameters (such as p_i).

Hint: You can write an expression for p_0 in terms of $p_1, \ldots p_k$, β, k, and N. You can also write the PageRanks of the boosting pages in terms of p_0, β, k, and N. Remember that λ is just the PageRank score from the webpages linking to the node from outside the spam farm. p_0 should include teleportation.

1.4 Spam farms II [5 points]

It turns out that the structure in Figure 1 is optimal in the sense that it maximizes the target's PageRank p_0 with the resources available. However, it may still be possible to do better by joining forces with other spammers. It also turns out that the λ contribution from the rest of the graph is mathematically equivalent to having some extra number of boosting pages, so for the rest of this question we will ignore λ. Consider the case where two spammers link their spam farms by each linking to the other's target page as well as their own, as shown in Figure 2. Let p'_0 and q'_0 denote

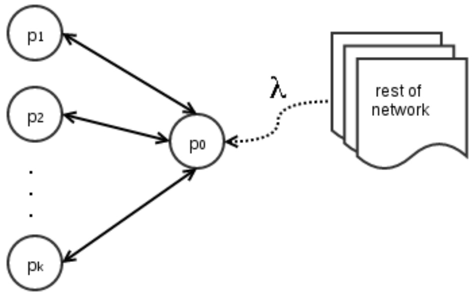

Figure 1: A spam farm.

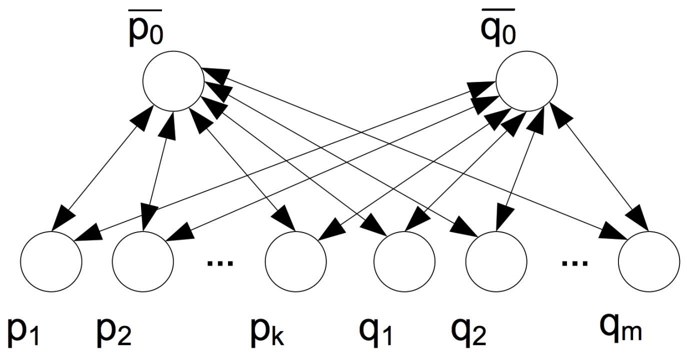

Figure 2: Linked spam farms.

the PageRank values of the target pages if the spammers use the individual spam configuration that we discussed in the previous question (without λ this time). Calculate the PageRanks of the target pages $\overline{p_0}$ and $\overline{q_0}$ with the new configuration as a function of k, m, β, and N (where again N denotes the number of pages in the web graph). What are $\overline{p_0} - p_0'$ and $\overline{q_0} - q_0'$? Are the spammers better off than they would be if they operated independently, i.e., is $\overline{p_0} + \overline{q_0} > p_0' + q_0'$?

1.5 Spam farms III [5 points]

There are other ways spammers can form alliances. Consider the setup shown in Figure 3, where the spammers only link their target pages together. Again let p_0' and q_0' denote the PageRank values of the target pages that the spammers would get on their own. Calculate the PageRank of the target pages $\overline{\overline{p_0}}$ and $\overline{\overline{q_0}}$ with this configuration as a function of k, m, β, and N (where again N denotes the number of pages in the web graph). What are $\overline{\overline{p_0}} - p_0'$ and $\overline{\overline{q_0}} - q_0'$? Are the spammers better off than they would be if they operated independently, i.e., is $\overline{\overline{p_0}} + \overline{\overline{q_0}} > p_0' + q_0'$?

What to submit

Page 1: • For each of (i),(ii), and (iii), 'yes' or 'no' and a brief explanation of why or why not.

Page 2: • A mathematical expression for the set in terms of V and a brief explanation.

Page 3: • An expression for p_0 in terms of k, β, and N. Also show how you derived the expression.

Page 4: • Expressions for $p_0' + q_0'$, $\overline{p_0} - p_0'$, $\overline{q_0} - q_0'$ and $\overline{p_0} + \overline{q_0}$ in terms of k, m, β and N. Show

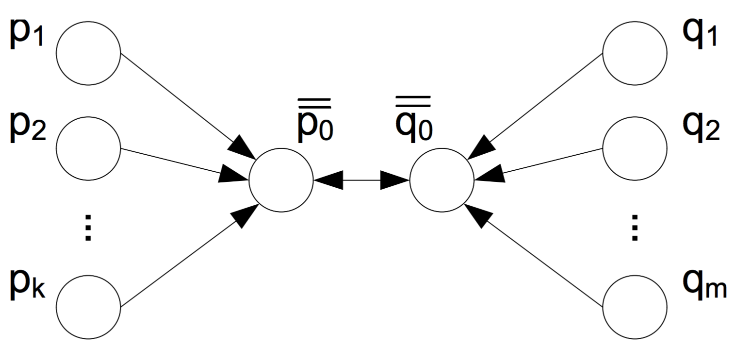

Figure 3: Another way of linking two spam farms.

how you derived these expressions.

- A yes/no answer to whether spammers are better off and a brief explanation.

Page 5:
- Expressions for $p_0' + q_0'$, $\overline{\overline{p_0}} - p_0'$, $\overline{\overline{q_0}} - q_0'$ and $\overline{\overline{p_0}} + \overline{\overline{q_0}}$ in terms of k, m, β and N. Show how you derived the expressions.
- A yes/no answer to whether spammers are better off and a brief explanation.

2 Approximate Betweenness Centrality [25 points – Anunay, Praty]

Betweenness centrality is an important concept in network analysis. For instance, in class you have seen the Girvan–Newman algorithm for community detection, which uses betweenness centrality as a subroutine.

Given an undirected graph $G = (V, E)$, the betweenness centrality of an edge $\{v, w\} \in E$ is

$$B(\{v,w\}) = \sum_{(s,t)\in V^2} \frac{\sigma_{st}(\{v,w\})}{\sigma_{st}},$$

where σ_{st} is the number of shortest paths between s and t, and $\sigma_{st}(\{v,w\})$ is the number of shortest paths between s and t that contain the edge $\{v, w\}$.

In this question, we assume graph G is a connected graph. That is, there always exists a path between any two nodes in G.

Betweenness centrality can be computed with the following algorithm.

Algorithm 1 (exact betweenness centrality). For each vertex $s \in V$, perform a BFS from s, which induces a BFS tree T_s. For each $v \in V$, let v's *parent set* $P_s(v)$ be defined as the set of nodes $u \in V$ that immediately precede v on some shortest path from s to v in G. During the BFS, also compute, for every $v \in V$, the number σ_{sv} of shortest paths between s and v, according to the recurrence

$$\sigma_{sv} = \begin{cases} 1 & \text{if } v = s \\ \sum_{u\in P_s(v)} \sigma_{su} & \text{otherwise.} \end{cases}$$

After the BFS has finished, compute the so-called *dependency* of s on each edge $\{v, w\} \in E$ using the recurrence

$$\delta_s(\{v,w\}) = \begin{cases} \frac{\sigma_{sv}}{\sigma_{sw}} & \text{if } w \text{ is a leaf of } T_s \\ \frac{\sigma_{sv}}{\sigma_{sw}} \left(1 + \sum_{x:w\in P_s(x)} \delta_s(\{w,x\})\right) & \text{otherwise} \end{cases}$$

Assume without loss of generality that the shortest path from s to v is shorter than to w. This has two important implications:

- The δ values are only really defined for edges that connect two nodes, u and v, where u is further away from s than v. Assume that δ is 0 for cases that are undefined (i.e., where an edge connects two nodes that are equidistant from s).
- You probably shouldn't iterate over all edges when computing your dependency values. It makes more sense to just look at edges that are in some BFS tree starting from s, since other edges will have zero dependency values anyways (they are not on any shortest paths). (Though iterating over all edges is not technically wrong if you just set the dependency values to 0 when the nodes are equidistant from s.)
- Additionally, you need to be very careful about only computing dependency for each edge once, and you need to make sure that you have the right σ values in the numerator/denominator. The node that is closer to the start node s should always have it values in the numerator.

Also, since the edges are undirected

$$\delta_s(\{v,w\}) = \delta_s(\{w,v\})$$

Finally, the betweenness centrality of $\{v,w\}$ or $B(\{v,w\})$

$$B(\{v,w\}) = \sum_{s \in V} \delta_s(\{v,w\})$$

Algorithm 1 has a time complexity of $O(nm)$, where n and m are the numbers of nodes and edges, respectively, and there is no faster known algorithm for computing betweenness centrality exactly in undirected, unweighted graphs. Note that with Algorithm 1, one must compute the betweenness centrality of *all* edges, even if one is only interested in some select edges.

In this problem you'll explore an approximate version of Algorithm 1. It is nearly identical to Algorithm 1, with the difference that it doesn't start a BFS from every node but rather samples starting nodes randomly with replacement. Also, it can approximate betweenness for any edge $e \in E$ without necessarily having to compute the centrality of all other edges as well.

Algorithm 2 (approximate betweenness centrality). Repeatedly sample a vertex $v_i \in V$; perform a BFS from v_i (as in Algorithm 1) and maintain a running sum Δ_e of the dependency scores $\delta_{v_i}(e)$ (one Δ_e for each edge e you're interested in). Sample until Δ_e is greater than cn for some constant $c \geq 2$. Let the total number of samples be k. The estimated betweenness centrality score of e is given by $\frac{n}{k}\Delta_e$.

Theorem 1 is the approximation guarantee about Algorithm 2. You will see during implementation that Algorithm 2 runs much faster in practice.

Theorem 1. For $0 < \epsilon < \frac{1}{2}$, if the centrality of an edge e is $\frac{n^2}{t}$ for some constant $t \geq 1$, then with probability at least $1 - 2\epsilon$, its centrality can be estimated within a factor of $\frac{1}{\epsilon}$ with $\epsilon \cdot t$ samples of source vertices.

Simulation. For this question, you will implement Algorithms 1 and 2 and compare them. Proceed as follows:

- Generate a random graph following the Barabási–Albert preferential attachment model, on $n = 1000$ nodes and attaching each node to 4 existing nodes when adding it to the network (use the SNAP function `snap.GenPrefAttach(1000, 4)`).
- Implement Algorithm 1 and use it to compute the exact betweenness centrality for all edges $e \in E$.
- Implement Algorithm 2 and use it to compute the approximate betweenness centrality for all edges $e \in E$. Use $c = 5$ and sample at most $\frac{n}{10}$ random starting nodes v_i.
- Each algorithm induces an ordering over edges with respect to betweenness centrality. Hand in the following plot: One curve for Algorithm 1, one for Algorithm 2, overlaid in the same plot. If the edge with the x-th largest betweenness centrality (according to the respective algorithm) has betweenness centrality y, draw a dot with co-ordinates (x, y). Please use a logarithmic y-axis. Briefly comment on the curves.
- Examine the difference between the runtimes of the two algorithms and briefly comment on the same.

What to submit

Page 6:
- A plot and a brief comment on the curves.
- A brief comment on the difference between runtimes of the two algorithms.

3 Stochastic Kronecker Graphs [20 points – Yokila]

In this problem, we will study some mathematical properties of Stochastic Kronecker Graphs.

3.1 Bit vector representation [2 points]

In this sub-question, you will derive a useful bit vector representation for the nodes of a stochastic Kronecker graph. In the remaining sub-questions, you will use these bit vector representations to derive closed form expressions for edge probabilities and other graph properties.

Given the symmetric 2×2 probability matrix Θ_1, let $\Theta_1[1,1] = \alpha$, $\Theta_1[1,0] = \Theta_1[0,1] = \beta$ and $\Theta_1[0,0] = \gamma$, where $0 \leq \gamma \leq \beta \leq \alpha \leq 1$. The stochastic Kronecker graph based on the k^{th} Kronecker power Θ_k has 2^k nodes.

Show how to label each node by a unique bit vector of length k so that the probability of an edge existing between node $u = (u_1 u_2 ... u_k)$ and $v = (v_1 v_2 ... v_k)$ is $\prod_{b=1}^{k} \Theta_1[u_b, v_b]$.

3.2 Edge probabilities [5 points]

We define the *weight* of a vertex to be the number of 1's in its bit vector label. Consider a node u with weight l. For any node v, let i be the number of bits where $u_b = v_b = 1$, and let j be the number of bits where $u_b = 0$, $v_b = 1$ (here, u_b and v_b are the bth bits in the representation of u and v). Assume that there are k bits in each node's representation. Using your result from part 1, derive an expression for $P[u, v]$ in terms of i, j, and l, where $P[u, v]$ is the probability of an edge occurring between nodes u and v.

Hint: Your expression should involve α, β, and γ from part 1.

3.3 Degrees [5 points]

What is the expected degree of node u with weight l represented with k bits?

Hint: you may need to use the binomial formula, $(x+y)^n = \sum_{k=0}^{n} \binom{n}{k} x^k y^{n-k}$.

3.4 Number of edges [4 points]

Based on the result from part 3, you can easily calculate out other properties of the graph. Assuming that the graph is undirected, what is the expected number of edges in the graph? (You do not need to treat the self-loop as a special case for this part.)

3.5 Self-loops [4 points]

Again assume the graph is undirected. What is the expected number of self-loops?

What to submit

Page 7:	• Short derivation.
Page 8:	• $P[u, v]$ in terms of the variables given and show/explain work.
Page 9:	• The expected degree of node u (show and explain your work).
Page 10:	• The expected number of edges in the graph (show and explain your work).
Page 11:	• The expected number of self-loops in the graph (show and explain your work).

4 Spectral clustering [35 points – Anthony, Ziyi]

This question derives some spectral clustering algorithms which we then use to analyze a real-world dataset. These algorithms use eigenvectors of matrices associated with the graph. You might find this handout on graph clustering helpful for solving this question.

We first discuss some notation. Let $G = (V, E)$ be a simple (that is, no loops or multiple edges) undirected connected graph with $n = |V|$ and $m = |E|$. Let A be the adjacency matrix of the graph G, i.e., A_{ij} is equal to 1 if $(i, j) \in E$ and equal to 0 otherwise. Let D be the diagonal matrix of degrees: $D_{ii} = \sum_j A_{ij}$. We define the *graph Laplacian* of G by $L = D - A$.

For a set of nodes $S \subset V$, we will measure the quality of S as a cluster with a "cut" value and a "volume" value. We define the cut of the set S to be the number of edges that have one end point in S and one end point in the complement set $\bar{S} = V \backslash S$:

$$\text{cut}(S) = \sum_{i \in S, j \in \bar{S}} A_{ij}.$$

Note that the cut is symmetric in the sense that $\text{cut}(S) = \text{cut}(\bar{S})$. The *volume* of S is simply the sum of degrees of nodes in S:

$$\text{vol}(S) = \sum_{i \in S} d_i,$$

where d_i is the degree of node i.

4.1 A spectral algorithm for normalized cut minimization [10 points]

We will try to find a set S with a small normalized cut value:

$$\text{NCUT}(S) = \frac{\text{cut}(S)}{\text{vol}(S)} + \frac{\text{cut}(\bar{S})}{\text{vol}(\bar{S})} \tag{1}$$

Intuitively, a set S with a small normalized cut value must have few edges connecting to the rest of the graph (making the numerators small) as well as some balance in the size of the clusters (making the denominators large).

Define the assignment vector x for some set of nodes S such that

$$x_i = \begin{cases} \sqrt{\frac{\text{vol}(\bar{S})}{\text{vol}(S)}} & i \in S \\ -\sqrt{\frac{\text{vol}(S)}{\text{vol}(\bar{S})}} & i \in \bar{S} \end{cases} \tag{2}$$

Prove the following properties:

(i) $x^T Lx = c \cdot \text{NCUT}(S)$ for some constant c (in terms of the problem parameters).

(ii) $x^T De = 0$, where e is the vector of all ones.

(iii) $x^T Dx = 2m$.

Since $x^T Dx$ is just a constant ($2m$), we can formulate the normalized cut minimization problem in the following way:

$$\begin{aligned} \underset{S\subset V,\, x\in\mathbb{R}^n}{\text{minimize}} \quad & \frac{x^T Lx}{x^T Dx} \\ \text{subject to} \quad & x^T De = 0,\ x^T Dx = 2m,\ x \text{ as in Equation } 2 \end{aligned} \tag{3}$$

The constraint that x takes the form of Equation 2 makes the optimization problem NP-hard. We will instead perform a "relax and round" technique where we relax the problem to make the optimization problem tractable and then round the relaxed solution back to a feasible point for the original problem. Our relaxed problem will eliminate the constraint that x take the form of Equation 2 which leads to the following relaxed problem:

$$\begin{aligned} \underset{x\in\mathbb{R}^n}{\text{minimize}} \quad & \frac{x^T Lx}{x^T Dx} \\ \text{subject to} \quad & x^T De = 0,\ x^T Dx = 2m \end{aligned} \tag{4}$$

Show that the minimizer of Equation 4 is $D^{-1/2}$v, where v is the eigenvector corresponding to the second smallest eigenvalue of the *normalized graph Laplacian* $\tilde{L} = D^{-1/2}LD^{-1/2}$. Finally, to round the solution back to a feasible point in the original problem, we can take the indices of all positive entries of the eigenvector to be the set S and the indices of all negative entries to be $\bar{S}$.

Hint 1: Make the substitution $z = D^{1/2}x$.

Hint 2: Note that e is the eigenvector corresponding to the smallest eigenvalue of L.

Hint 3: The normalized graph Laplacian $\tilde{L}$ is symmetric, so we can write any vector x as a linear combination of orthonormal eigenvectors of $\tilde{L}$.

4.2 A spectral algorithm for modularity maximization [8 points]

Let $y \in \{1, -1\}^n$ be an assignment vector for a set S:

$$y_i = \begin{cases} 1 & \text{if } i \in S \\ -1 & \text{if } i \in \bar{S} \end{cases} \tag{5}$$

The *modularity* of the assignment y is

$$Q(y) = \frac{1}{2m} \sum_{1 \le i,j \le n} \left[A_{ij} - \frac{d_i d_j}{2m} \right] I_{y_i = y_j}. \tag{6}$$

Here, each term $\frac{d_i d_j}{2m}$ is approximately the expected number of edges between i and j in a random multi-graph generated with the same degree sequence as the graph G (think about the configuration model from Problem Set 2 but allow multiple edges between a pair of nodes). Thus, $A_{ij} - \frac{d_i d_j}{2m}$ measures how "surprising" the link is between nodes i and j.

Prove that $Q(y) = cy^T By$ for some constant c, where $B = A - \frac{1}{2m} dd^T$.

Hint: first show that $Be = 0$ and then use the fact that $y_i y_j + 1 = 2I_{y_i = y_j}$.

Note that for any assignment vector y we have $y^T y = n$. Thus, we can write the modularity maximization problem as

$$\begin{aligned} \underset{S \subset V,\, y \in \mathbb{R}^n}{\text{maximize}} \quad & y^T By \\ \text{subject to} \quad & y^T y = n, \ y \text{ as in Equation 5} \end{aligned} \tag{7}$$

We again take the relax and round approach to make the optimization problem computationally tractable. By removing the constraint that y has the form of Equation 5, we are left with the optimization problem

$$\begin{aligned} \underset{y \in \mathbb{R}^n}{\text{maximize}} \quad & y^T By \\ \text{subject to} \quad & y^T y = n \end{aligned} \tag{8}$$

Prove that the optimizer to Equation 8 is the eigenvector corresponding to the maximum (i.e., largest algebraic) eigenvalue of B. Finally, to round the solution, we can take the indices of all positive entries of the eigenvector to be the set S and the indices of all negative entries to be $\bar{S}$.

Hint: The modularity matrix B is also symmetric, so your arguments should be similar to the ones in question 4.1.

4.3 Relating modularity to cuts and volumes [7 points]

In the previous part, we presented modularity as a measurement of "surprise" in the clustering compared to a null model. However, modularity actually relates to cuts and volumes as well. Let y be the assignment vector in Equation 5. Prove that

$$Q(y) = \frac{1}{2m} \left(-2 \cdot \text{cut}(S) + \frac{1}{m} \text{vol}(S) \cdot \text{vol}(\bar{S}) \right) \tag{9}$$

Thus, maximizing modularity is really just minimizing the sum of the cut and the negative product of the partition's volumes.

4.4 Network analysis of political blogs [10 points]

Now we will use the spectral algorithms to cluster a network. Download the `polblogs` network: `http://cs224w.stanford.edu/homeworks/hw4/data_sets/polblogs.txt`.
This is a network where the nodes are political blogs and there is an undirected edge between two nodes i and j if either blog i links to blog j or blog j links to blog i. The blogs are labeled as liberal (label 0) or conservative (label 1). Download the labels from `http://cs224w.stanford.edu/homeworks/hw4/data_sets/polblogs-labels.txt`.

Implement the spectral algorithms for normalized cut minimization and modularity maximization. Use the algorithms to partition the `polblogs` network into two sets of nodes S and $\bar{S}$. Compute and report the sizes of S and $\bar{S}$ and the purity of the cluster assignments from each of the two algorithms.

What to submit

Page 12:
- Proofs of the three properties.
- Proof of the minimizer.

Page 13:
- Proof of the quadratic form of modularity.
- Proof of the maximizer.

Page 14:
- Proof of the relationship between modularity and cuts and volumes.

Page 15:
- Number of nodes in S and $\bar{S}$ in the clusters found by the normalized cut minimization spectral algorithm and the modularity maximization spectral algorithm.
- Purity of the clusters with respect to the true labels for both algorithms.

Information Sheet
CS224W: Analysis of Networks

Assignment Submission Fill in and include this information sheet with each of your assignments. This page should be the last page of your submission. Assignments are due at 11:59pm and are always due on a Thursday. All students (SCPD and non-SCPD) must submit their homeworks via GradeScope (`http://www.gradescope.com`). Students can typeset or scan their homeworks. Make sure that you answer each (sub-)question on a separate page. That is, one answer per page regardless of the answer length. Students also need to upload their code at `http://snap.stanford.edu/submit`. Put all the code for a single question into a single file and upload it. Please do not put any code in your GradeScope submissions.

Late Homework Policy Each student will have a total of *two* free late periods. *Homeworks are due on Thursdays at 11:59pm PDT and one late period expires on the following Monday at 11:59pm PDT.* Only one late period may be used for an assignment. Any homework received after 11:59pm PDT on the Monday following the homework due date will receive no credit. Once these late periods are exhausted, any assignments turned in late will receive no credit.

Honor Code We strongly encourage students to form study groups. Students may discuss and work on homework problems in groups. However, each student must write down their solutions independently i.e., each student must understand the solution well enough in order to reconstruct it by him/herself. Students should clearly mention the names of all the other students who were part of their discussion group. Using code or solutions obtained from the web (github/google/previous year solutions etc.) is considered an honor code violation. We check all the submissions for plagiarism. We take the honor code very seriously and expect students to do the same.

Your name: ______________________________

Email: ____________________ **SUID:** ____________________

Discussion Group: ______________________________

I acknowledge and accept the Honor Code.

(Signed) ______________________________