CME 305: Discrete Mathematics and Algorithms

Instructor: Reza Zadeh (rezab@stanford.edu)

Final Review Session – 03/20/17

1. Let $G = (V, E)$ be an unweighted, undirected graph. Let λ_1 be the maximum eigenvalue of the adjacency matrix (A) of G. Suppose Δ is the max degree of G and $\bar{d}$ is the average degree of G, then show the following:

$$\bar{d} \leq \lambda_1 \leq \Delta$$

Note that the above also holds for weighted graphs and weighted degrees, using pretty much the same proof.

Solution: First we show the lower bound. By definition of the top eigenvector:

$$\lambda_1 = \max_x \frac{x^T A x}{x^T x} \geq \frac{1^T A 1}{1^T 1} = \frac{\sum_{(u,v)\in E} A(u,v)}{n} = \sum_u \frac{d(u)}{n} = \bar{d}$$

where 1 is the all-ones vector. For the upper bound, let v_1 be the eigenvector corresponding to λ_1. Let v be the vertex on which it takes its maximum, i.e. $v_1(v) \geq v_1(u)$ for all $u \in V$. We know $v_1(v) > 0$ by Perron-Frobenius thm (see hint below). Therefore, we have the following:

$$\lambda_1 = \frac{(Av_1)(v)}{v_1(v)} = \frac{\sum_{u:(u,v)\in E} v_1(u)}{v_1(v)} = \sum_{u:(u,v)\in E} \frac{v_1(u)}{v_1(v)} \leq \sum_{u:(u,v)\in E} 1 \leq \Delta$$

2. What are the eigenvalues of the adjacency matrix and laplacian matrix for the complete graph?

Solution: For the complete graph, we have $A_{ij} = 1$ if $i \neq j$ and 0 along the diagonal. Therefore, $A = O - I$, where O is the all ones matrix. Because O is rank 1, it has $n-1$ 0-eigenvalues and we know trace$(A) = \sum_i \lambda_i = \lambda_1 = n$. Subtracting I decreases all eigenvalues by 1, so A has one eigenvalue of $n-1$ and $n-1$ eigenvalues of -1. Note that the Laplace matrix of the complete graph is given by $L = nI - O$, so all eigenvalues of $-O$ are scaled by n and L's spectrum has one eigenvalue of 0 and $n-1$ eigenvalues of n.

3. What are the eigenvalues of the adjacency matrix for the star graph?

Hint: you may use the fact (without proof) that a connected graph G with maximum eigenvalue (adjacency) λ_1 is bipartite if and only if $-\lambda_1$ is also an eigenvalue.

Solution: Without loss of generality label the central node of the star graph as node 1. Then the adjacency matrix of the star graph has the following form:

$$A_{ij} = \begin{cases} 1, & \text{if } i = 1,\ j \neq 1. \\ 1, & \text{if } i \neq 1,\ j = 1. \\ 0, & \text{otherwise.} \end{cases}$$

Then as A has $n-1$ identical columns (of e_1), it is rank 2. It has $n-2$ eigenvalues of 0. Furthermore, the star graph is bipartite. We may show that the eigenvalues of adjacency matrices of bipartite graphs are symmetric around 0 - we may just use this fact as its given as hint (it's not too hard to prove but we will omit those details).

Now consider a vector x such that $x_1 = 1$ and $x_k = 1/\sqrt{n-1}$ for $2 \leq k \leq n$. Then we have $Ax = \sqrt{n-1}x$ and $\lambda_1 = \sqrt{n-1}$. Using the claim above and the fact that A is rank 2, the spectrum of the star graph is: $\{\sqrt{n-1}, 0, ..., 0, -\sqrt{n-1}\}$.

4. Let $G = (V, E)$ be a connected, undirected graph. Let $H = (V, E')$ be a connected subgraph of G.

   (a) Show $\lambda_1(A(H)) \leq \lambda_1(A(G))$, where $\lambda_1(A(G))$ is the largest eigenvalue of the adjacency matrix associated with G.

   Hint: you may use the Perron-Frobenius thm, which says that a real square matrix with positive entries has a unique largest real eigenvalue and that the corresponding eigenvector can be chosen to have strictly positive components.

   Solution: Let A and A' be the adjacency matrices of G and H respectively. Let x and x' be the eigenvectors of A and A' corresponding to eigenvalues λ_1 and λ_1', so we have $Ax = \lambda_1 x$ and $A'x' = \lambda_1' x'$. Because A and A' are both non-negative matrices, by the Perron-Frobenius theorem, we have that both x and x' are non-negative as well.

   If H is a subgraph of G on the same vertices but with some edges deleted, we have:

$$\begin{aligned}\lambda_1' = \frac{(x')^T A' x'}{(x')^T x'} &= \frac{\sum_{i,j} x_i' a_{ij}' x_j'}{(x')^T x'} \\ &\leq \frac{\sum_{i,j} x_i' a_{ij} x_j'}{(x')^T x'} = \frac{(x')^T A x'}{(x')^T x'} \\ &\leq \sup_{x \neq 0} \frac{x^T A x}{x^T x} = \lambda_1\end{aligned}$$

   We get the first inequality above because x' is non-negative and $0 \leq a_{ij}' \leq a_{ij}$ for every i and j.

   (b) Show $\lambda_n(L(H)) \leq \lambda_n(L(G))$, where $\lambda_n(L(G))$ is the largest eigenvalue of the laplacian matrix associated with G.

   Solution Again, let L and L' be the laplacian matrices of G and H respectively. Let x and x' be the eigenvectors of these respective matrices with eigenvalues λ_n and λ_n'. We know $E' \subseteq E$, so we have the following:

$$\begin{aligned}\lambda_n' = \frac{(x')^T L' x'}{(x')^T x'} &= \frac{\sum_{(i,j)\in E'} (x_i' - x_j')^2}{(x')^T x'} \\ &\leq \frac{\sum_{(i,j)\in E} (x_i' - x_j')^2}{(x')^T x'} \\ &\leq \sup_{x \neq 0} \frac{x^T L x}{x^T x} = \lambda_n\end{aligned}$$

5. Recall the knapsack problem: there are n items each with some value $v_1, \dots, v_n > 0$ and weight $w_1, \dots, w_n > 0$ and a capacity $W > 0$. Suppose $W \geq w_i$ for all i and now consider the version of knapsack where one one of each item exists. Consider the following greedy algorithm: order all items in decreasing value/weight ratio (and relabel) such that $\frac{v_1}{w_1} \geq \frac{v_2}{w_2} \geq \cdots \geq \frac{v_n}{w_n}$, and take the first k items that fit in the knapsack such that the next item $(k+1)$ does not.

   (a) Show that this algorithm may be aribitrarily bad (unbounded approximation ratio).

   Solution: Consider the case with just two objects, $\{a_1, a_2\}$ such that $\frac{v_1}{w_1} > \frac{v_2}{w_2}$, but $w_2 = W$. Then greedy will pick a_1 and not have room for the second object. But the ratio of v_2 to v_1 may be made to be arbitrarily bad.

   (b) Consider the following modified algorithm: compute the greedy solution as before and find the item of maximum value v_{i^*}. Output the maximum of the greedy algorithm and v_{i^*}. Show that this new algorithm gives a $\frac{1}{2}$-approximation.

   Solution: Let k be the index of the first item that is not accepted by our greedy algorithm. Then we can show $v_1 + v_2 + \cdots + v_{k-1} + \alpha v_k \geq OPT$, where $\alpha = \frac{W - (w_1 + \cdots + w_{k-1}}{w_k}$, i.e. the fraction of the $k-th$ item that we can fit into the sack to completely fill it. To see this fact, we can write knapsack as an integer program and relax it to an LP.

$$\begin{aligned} \max \quad & \sum_{i=1}^{n} v_i x_i \\ \text{subject to} \quad & \sum_{i=1}^{n} w_i x_i \leq W \\ & 0 \leq x_i \leq 1, \text{ for all } i \end{aligned}$$

   Then $x_1 = \cdots = x_{k-1} = 1$, $x_k = \alpha$, $x_j = 0$ for $j > k$ is feasible for the above LP and in fact optimal (work out on own). So we know at least one of $v_1 + \cdots + v_{k-1}$ and αv_k must be greater that $OPT/2$. The first is our output from greedy and the second can be bounded by the following $\alpha v_k < v_k \leq v_{i^*}$, so our simple modification is a $\frac{1}{2}$-approximation.

6. The SETCOVER problem is as follows: Given a set E of elements and a collection $S_1, \dots, S_n$ of subsets of E, what is the minimum number of these sets whose union equals E?

   Let $f(e)$ be the number of sets in our collection of subsets that contain $e \in E$. Let $f = \max_{e \in E} f(e)$. Give a f-approximation algorithm to this problem.

   Note that you can also come up with a $\log n$-approximation algorithm that does not depend on f. If you're interested, try doing that as well.

Solution: We will write the problem as an IP first:

$$\begin{aligned} \min \quad & \sum_{i=1}^{n} x_i \\ \text{subject to} \quad & \sum_{j:e\in S_j} x_j \geq 1, \ \forall e \in E \\ & x_i \in \{0,1\}, \ 1 \leq i \leq n \end{aligned}$$

Notice that when $f = 2$, this is the same as vertex cover. That is, each set S_v is the set of edges that v covers - and each edge is covered by exactly 2 nodes. Now we may relax the integral constraint to give an LP – $0 \leq x_i \leq 1$ now – just as we did for the approximation algorithm for vertex cover.

We know the min computed from the LP will be less than or equal to OPT because we have just increased the feasible region by relaxing. Now we must round our solution. Let x^* be the LP solution and consider the following rounding:

$$y_i = \begin{cases} 1 \text{ if } x_i^* \geq \frac{1}{f}, \\ 0 \text{ otherwise.} \end{cases}$$

First see that $\sum_i y_i \leq f \sum_i x_i^* \leq f \cdot OPT$. Now we must show that our collection of sets given by y covers all elements in E. Because each element of E is in at most f sets from our collection, each constraint of the form $\sum_{j:e\in S_j} x_j \geq 1$ has at most f terms in the sum, so for each of these constraints, at least one of the x_i^* in the sum is at least $\frac{1}{f}$. So for each constraint, we are rounding up at least one term. Therefore, for each element in E, y must contain at least one set that covers it, and we are done.

Note that for the $\log n$ approximation algorithm, we can just use the greedy algorithm - try doing this on your own.

7. You're consulting for an e-commerce site that receives a large number of visitors each day. For each visitor i, where $i \in \{1, 2, \ldots, n\}$, the site has assigned a value v_i, representing the expected revenue that can be obtained from this customer.

   Each visitor i is shown one of m possible ads $A_1, A_2, \ldots, A_m$ as he or she enters the site. The site wants a selection of one ad for each customer so that *each* ad is seen, overall, by a set of customers of reasonably large total weight. Thus, given a selection of one ad for each customer, we will define the *spread* of this selection to be the minimum, over $j = 1, 2, \ldots, m$, of the total weight of all customers who were shown ad A_j.

   Example: Suppose there are six customers with values $3, 4, 12, 2, 4, 6$, and there are $m = 3$ ads. Then, in this instance, one could achieve a spread of 9 by showing ad A_1 to customers $1, 2, 4$, ad A_2 to customer 3, and ad A_3 to customers 5 and 6.

   The ultimate goal is to find a selection of an ad for each customer that maximizes the spread. Unfortunately, this optimization problem is NP-hard (you don't have to prove this). So instead, we will try to approximate it.

(a) Give a polynomial-time algorithm that approximates the maximum spread to within a factor 2. That is, if the maximum spread is s, then your algorithm should produce a selection of one ad for each customer that has spread at least $s/2$. In designing your algorithm, you may assume that no single customer has a value that is significantly above the average; specifically, if $\bar{v} = \sum_{i=1}^{n} v_i$ denotes the total value of all customers, then you may assume that no single customer has a value exceeding $\bar{v}/(2m)$.

Solution: First we have the following bound on the optimal solution: $OPT \leq \frac{\bar{v}}{m}$, otherwise the sum of expected revenue assigned to adds would exceed the sum of expected revenues of customers.

Consider the greedy algorithm. We arbitrarily order the customers, and for each customer we assign him/her to the ad with the minimum weight currently assigned to it (breaking ties arbitrarily).

At the end of the algorithm, there exists some ad A_k such that the weight of $A_k \geq \frac{\bar{v}}{m}$. Let $w(A_k)$ denote the weight of customers shown ad A_k. Now consider the last customer shown ad A_k, say v_j without loss of generality. At this time, the weight of A_k was minimum among all ads. The weight of A_k at this time was:

$$\begin{aligned} w(A_k) - v_j &\leq \frac{\bar{v}}{m} - \frac{\bar{v}}{2m} \\ &= \frac{\bar{v}}{2m} \end{aligned}$$

where we have used the facts that $w(A_k) \geq \frac{\bar{v}}{m}$ and $v_i \leq \frac{\bar{v}}{2m}$ for all i. Therefore, at this time, all ads have weight $\geq \frac{\bar{v}}{2m}$, so output of algorithm is at least this as well. And we have the following: $ALG \geq \frac{\bar{v}}{2m} \geq OPT/2$.

Notice that this is very similar to the minimum makespan problem except now instead of minimizing the max load we are maximizing the minimum load - the analysis should be pretty familiar.

(b) Give an example of an instance on which the algorithm you designed in part (a) does not find an optimal solution (that is, one of maximum spread). Say what the optimal solution is in your sample instance, and what your algorithm finds.

Solution: Consider the simple choice of $m = 2$, $n = 3$ and the values of v are $v_1 = 1$, $v_2 = 1$, $v_3 = 2$. Suppose we considered the v's in order, then our assignment would be v_1, v_3 to A_1 and v_2 to A_2 with spread 1. But optimal achieves a spread of 2.

8. Consider the following problems. Show that each is NP-complete.

(a) HITTING SET: Given a family of sets $\{S_1, S_2, \ldots, S_n\}$ and an integer b, is there a set H with b or fewer elements such that H intersects all sets in the family?

(b) LONGEST CYCLE: Given a graph and integer k, is there a cycle with no repeated nodes of length at least k?

(c) MAX BISECTION: Given a graph $G = (V, E)$ and integer k, does a bisection exist (i.e. $|S| = |V \setminus S| = \frac{|V|}{2}$) such that the cutsize of S is at least k?

(d) SUBGRAPH ISOMORPHISM: Given two undirected graphs G and H, is H a subgraph of G? That is, if H has nodes $v_1, \ldots, v_n$, can you find distinct nodes $u_1, \ldots, u_n$ in G such that (u_i, u_j) is an edge in G whenever (v_i, v_j) is an edge in H?

Solution: Remember to show that a problem is NP-complete, we must show that it is in NP and that it is NP-hard by reducing a known NP-hard problem to our problem.

For all the above problems, it's not difficult to see that they can be checked (given a yes certificate) in polynomial time - make sure you understand why. I will know give hints for each reduction.

For HITTING SET, we reduce VERTEX COVER to this problem. Given an input of VERTEX COVER, we create sets such that each set contains the two nodes for an edge for each edge, then we want to find the k elements (nodes) that cover all the sets (edges).

For LONGEST CYCLE, we use HAMILTONIAN CYCLE and $k = |V|$.

For MAX BISECTION, we solve MAX CUT using an algorithm for this problem. Suppose $G = (V, E)$ and k is an input to MAX CUT. Create a new graph G' with $|V|$ new "dummy" nodes (isolated vertices), so MAX BISECTION on this new graph is equivalent to MAX CUT on G.

For SUBGRAPH ISOMORPHISM, we can just reduce CLIQUE to this problem. CLIQUE takes as input a graph G and integer k and asks if G contains a clique of size K (note that this is NP-hard because it is exactly INDEPENDENT SET on the compliment graph). We can easily use SUBGRAPH ISOMORPHISM to solve this problem then.