CS 170 HW 13

Due **2019-12-04, at 10:00 pm**

1 Study Group

List the names and SIDs of the members in your study group.

2 Maximum Coverage

The Maximum Coverage Problem is an optimization problem defined as follows.

Input: Subsets $S_1, \ldots, S_m$ of a set $\mathcal{U}$, and a positive integer $k \leq m$.

Output: Find $T \subseteq \{1, \ldots, m\}$ of size k such that the size of $\bigcup_{i \in T} S_i$ is maximized (i.e., find k subsets whose union has maximum size.) We say that the elements in this union are *covered* by the subsets.

1. Briefly argue that, if we can solve the Maximum Coverage problem optimally, then we can also solve the Set Cover problem. (Note: this implies that Maximum Coverage is NP-hard.)

2. Let us examine the greedy algorithm which proceeds in k steps as follows: in each step, pick the subset that covers the maximum number of (currently) uncovered elements.

   (a) Give a counterexample for which the greedy algorithm does not give an optimal solution.

   (b) Show that, when $k = 2$, this algorithm is a 3/4-approximation algorithm for the problem (i.e. the two sets picked by the algorithm covers 3/4 as many elements as the optimal solution). *Hint:* It may be helpful to look at the hint from part (c).

   (c) Generalize the analysis above to show that, for every k, the greedy algorithm is an $\left(1 - (1 - 1/k)^k\right)$-approximation algorithm for Maximum Coverage.
   Hint: Let x_i denote the number of elements covered by the greedy solution after picking i subsets. Try to show that $x_{i+1} - x_i \geq (OPT - x_i)/k$ where OPT is the total number of elements covered by the optimal solution. You might find it useful to consider the quantity $y_i = OPT - x_i$ as well.

Solution:

1. If we can solve Maximum Coverage, then we can run the algorithm (on the same input for the Set Cover problem) and check if the output subsets cover every element. If they do, then these k subsets are a solution for Set Cover. Otherwise, we know that there is no solution of size k for Set Cover.

2. (a) Consider the case where there are three subsets $S_1 = \{1, 2, 3\}, S_2 = \{4, 5, 6\}, S_3 = \{2, 3, 4, 5\}$ and $k = 2$. Greedy would pick S_3 in the first iteration, at which point no matter which other set is picked the union is not the whole set $\{1, 2, 3, 4, 5, 6\}$.

(b) Let OPT denote the total number of elements covered by the optimal solution and let x denote the number of elements covered by the first subset in the greedy solution. Since OPT elements can be covered by two subsets, the first subset picked by the greedy algorithm must cover at least $OPT/2$ elements, i.e., $x \geq OPT/2$.

Let us now examine the second subset picked by the greedy algorithm. Since at least $OPT - x$ elements covered by the optimal solution are uncovered by the first subset picked by the greedy algorithm, the second subset picked must cover at least $(OPT - x)/2$ uncovered elements.

In conclusion, the greedy algorithm covers at least $x + (OPT - x)/2 = OPT/2 + x/2 \geq 3OPT/4$ elements.

(c) Let x_i be as defined as in the hint. After the i-th subset is picked by the greedy algorithm, at least $OPT - x_i$ elements covered by the optimal solution have not been covered by the greedy solution. Since these elements are covered in the optimal solution (which uses k subsets), there must be one subset that covers at least $(OPT - x_i)/k$ such elements. Hence, the greedily picked subset also covers at least $(OPT - x_i)/k$ additional elements. In other words, we have that $x_{i+1} \geq x_i + (OPT - x_i)/k$.

Let $y_i = OPT - x_i$. The above inequality can be written as $y_i(1 - 1/k) \geq y_{i+1}$. This implies that $y_k \leq y_0(1 - 1/k)^k = OPT(1 - 1/k)^k$, which implies that $x_k \geq OPT(1 - (1 - 1/k)^k)$. As a result, the greedy algorithm is an $(1 - (1 - 1/k)^k)$-approximation algorithm for Maximum Coverage.

3 Multiway Cut

In the multiway cut problem, we are given a graph $G(V, E)$ with k special vertices $s_1, s_2 \ldots s_k$. Our goal is to find the smallest set of edges F which, when removed from the graph, disconnect the graph into at least k components, where each s_i is in a different component. When $k = 2$, this is exactly the min s-t cut problem, but if $k \geq 3$ the problem becomes NP-hard.

Consider the following algorithm: Let F_i be the set of edges in the minimum cut with s_i on one side and all other special vertices on the other side. Output F, the union of all F_i. Note that this is a multiway cut because removing F_i from G isolates s_i in its own component.

(a) Explain how each F_i can be found in polynomial time.

(b) Let F^* be the smallest multiway cut. Consider the components that removing F^* disconnects G into, and let C_i be the vertices in the component with s_i. Let F_i^* be the set of edges in F^* with exactly one endpoint in C_i. How many different F_i^* does each edge in F^* appear in? How do the sizes of F_i and F_i^* compare?

(c) Using your answer to the previous part, show that $|F| \leq 2|F^*|$. (Challenge: How could you modify this algorithm to output F such that $|F| \leq (2 - \frac{2}{k})|F^*|$?)

(As an aside, consider the minimum k-cut problem, where we want to find the smallest set of edges F whose removal disconnects the graph into at least k components. The following

greedy algorithm for minimum k-cut gets a $(2-\frac{2}{k})$-approximation: Initialize F to the empty set. While $G(V, E-F)$ has less than k components, find the minimum cut within each component of $G(V, E-F)$, and add the edges in the smallest of these cuts to F. Showing this is a $(2-\frac{2}{k})$-approximation is fairly difficult.)

Solution:

(a) Consider adding a vertex t to the graph and connecting t to all special vertices except s_i with infinite capacity edges. Then F_i is the minimum s_i-t cut, which we know how to find in polynomial time.

(b) Each edge in F^* appears in exactly two of the sets F_i^*.

Note that F_i^* is the set of edges in a cut which disconnects s_i from the other special vertices. Then by definition F_i has fewer edges than F_i^* since F_i is the minimum cut disconnecting s_i from all other special vertices.

(c) We combine the answers to the previous part and note that F's size is at most the total size of all F_i to get:

$$|F| \leq \sum_i |F_i| \leq \sum_i |F_i^*| = 2|F^*|$$

To get the $(2-\frac{2}{k})$-approximation, after computing all F_i, we instead output F as the union of all F_i except for the one with the most edges. Let this be F_j. This is still a multiway cut because each s_j is still disconnected from all other s_i. Then:

$$|F| \leq \sum_{i \neq j} |F_i| \leq (1-\frac{1}{k}) \sum_i |F_i| \leq (1-\frac{1}{k}) \sum_i |F_i^*| = (2-\frac{2}{k})|F^*|$$

4 Coffee Shops

A rectangular city is divided into a grid of $m \times n$ blocks. You would like to set up coffee shops so that for every block in the city, either there is a coffee shop within the block or there is one in a neighboring block. (There are up to 4 neighboring blocks for every block). It costs r_{ij} to rent space for a coffee shop in block ij.

Write an integer linear program to determine which blocks to set up the coffee shops at, so as to minimize the total rental costs.

(a) What are your variables, and what do they mean?

(b) What is the objective function?

(c) What are the constraints?

(d) Solving the linear program gets you a real-valued solution. How would you round the LP solution to obtain an integer solution to the problem? Describe the algorithm in at most two sentences.

(e) What is the approximation ratio obtained by your algorithm?

(f) Briefly justify the approximation ratio.

Solution:

(a) There is a variable for every block x_{ij}, i.e., $\{x_{ij} | i \in \{1, \ldots, m\}, j \in \{1, \ldots, n\}\}$. This variable corresponds to whether we put a coffee shop at that block or not.

(b) $\min \sum_{i=1}^{m} \sum_{j=1}^{n} r_{ij} x_{ij}$. Alternatively, $\min t$ is correct as well as long as the correct constraint is added.

(c) (i) $x_{ij} \geq 0$: This constraint just corresponds to saying that there either is or isn't a coffee shop at any block. $x_{ij} \in \{0, 1\}$ or $x_{ij} \in \mathbb{Z}_+$ is also correct.

(ii) For every $1 \leq i \leq m, 1 \leq j \leq n$:

$$x_{ij} + x_{(i+1),j} \mathbb{1}_{\{i+1 \leq m\}} + x_{(i-1),j} \mathbb{1}_{\{i-1 \geq 1\}} + x_{i,(j+1)} \mathbb{1}_{\{j+1 \leq n\}} + x_{i,(j-1)} \mathbb{1}_{\{j-1 \geq 1\}} \geq 1$$

This constraint corresponds to that for every block, there needs to be a coffee shop at that block or a neighboring block.

$\mathbb{1}_{\{i+1 \leq m\}}$ means "1 if $\{i+1 \leq m\}$, and 0 otherwise". It keeps track of the fact that we may not have all 4 neighbors on the edges, for instance.

(iii) If the objective was $\min t$, then the constraint $\sum_{i=1}^{m} \sum_{j=1}^{n} r_{ij} x_{ij} \leq t$ needs to be added.

(d) Round to 1 all variables which are greater than or equal to 1/5. Otherwise, round to 0. In other words, put a coffee shop on (i, j) iff $x_{i,j} \geq 0.2$.

(e) Using the rounding scheme in the previous part gives a 5-approximation.

(f) Notice that every constraint has at most 5 variables. So for every constraint, there exists at least one variable in the constraint which has value $\geq 1/5$ (not everyone is below average). The number of coffee shops in the rounded solution is at most $5 \cdot$ LP-OPT, (the number of $x_{i,j} \geq 0.2$ is at most $5 \cdot \sum_{i,j} x_{i,j}$). Since Integral-OPT $\geq$ LP-OPT (the LP is more general than the ILP), our rounding gives value at most 5 LP-OPT ≤ 5 Integral-OPT. So we get a 5-approximation.

5 Streaming for Voting

Consider the following scenario. Votes are being cast for a major election, but due to a lack of resources, only one computer is available to count the votes. Furthermore, this computer only has enough space to store one vote at a time, plus a single extra integer. Each vote is a single integer 0 or 1, denoting a vote for Candidate A and Candidate B respectively.

(a) Come up with an algorithm to determine whether candidate A or B won, or if there was a tie.

(b) Consider now an election with 3 candidates. Say there is a winner only if a candidate recieves more than 50 percent of the vote, otherwise there is no winner. If we're given another integer's worth of storage, come up with an algorithm to determine the winner if there is one. For simplicity, your algorithm can output any of the candidates in the case that there is no winner (not necessarily the one with the most votes). Votes are now numbered 0, 1, 2.

Solution:

(a) Initialize one of the integers i to 0. Use the other to store the incoming votes. For every vote for Candidate A, decrement i by one. For every vote to candidate B, incremenent i by 1.

If at the end i is negative, Candidate A won. If at the end i is positive, Candidate B won. If i is 0, there was a tie.

(b) Let m be a variable which will hold either 0, 1, or 2. Let i be a counter similar to in the previous part. For each element in the stream, do the following. If i is 0, set m equal to the value of the current vote, and set i to 1. Else if $i > 0$, and m is equal to the current vote in the stream, increment i. Else decrement i.

At the end, if there is a majority vote, m will contain it. However, if there is no majority vote, m will still contain some element of the stream.

We can see that this is the case with the following short inductive proof:

Base case: Our algorithm will definitely detect the majority element of a 1 element stream.

Inductive Hypothesis: Assume our algorithm works for a stream of n or fewer elements.

We have two cases. Either the first candidate's count drops to zero at some point during the streaming, or it doesn't.If the first candidate's count never drops to zero, then the first candidate must be the majority candidate.

If the first candidate's count does drop to zero (let's say after the xth vote), then we can say the following. Suppose we have not yet encountered the true majority element. Then it must be the majority of the rest of the stream. By the inductive hypothesis we will find it by the end of the algorithm. If we have encountered the true majority element already, we can say the following. This element could have appeared only up to $\frac{x}{2}$ times so far. Of the remaining $n - x$ elements, at least $(\frac{n}{2} + 1 - \frac{x}{2}) = \frac{n-x}{2} + 1$ must be the majority element. Thus the true majority element will also be the majority of the rest of the stream. We can then apply the inductive hypothesis again, completing the proof.

6 Approximate Median

Let $S = (x_1, x_2, \ldots, x_m)$ denote a stream of m elements. For simplicity, assume that the elements in the stream are unique. Define the *position* of an element to be $\text{pos}(x) = |\{y \in S | y \leq x\}|$. The ϵ-approximate median is then defined to be an element x such that:

$$\frac{m}{2} - \epsilon m \leq \text{pos}(x) \leq \frac{m}{2} + \epsilon m. \tag{1}$$

Provide an algorithm which returns a ϵ-approximate median with high probability. Provide a 3-part solution providing the algorithm, proof of correctness and the space complexity of the algorithm. Note that the optimal algorithm can solve the problem in space independent of the size of the stream.

Note: Your algorithm should take ϵ and the failure probability δ as parameters, and return a result that is ϵ-approximate with probability at least $1-\delta$. The proof of correctness should prove that your result is ϵ-approximate with this probability.

Hint: Try to provide a sampling based algorithm and argue that less than $1/2$ of the samples will be from the $(0, 1/2-\epsilon)$ percentile using a Hoeffding bound. Similarly, show that less than $1/2$ of the samples will be from the $(1/2+\epsilon, 1)$ percentile.

Hint: The following one-sided version Hoeffding bound might be useful for this (and other) problems: If $X_1, \ldots X_n$ are i.i.d Bernoulli with $\mathbb{E}(X_i) = p$, we have that:

$$\mathbb{P}\left(\frac{1}{t}\sum_{i=1}^{t} X_i - p \geq \epsilon\right) \leq \exp\left(-2\epsilon^2 t\right).$$

Solution: The algorithm that we will use is as follows: Sample t points from the stream uniformly at random using the Reservoir Sampling algorithm from class and return the median of the sampled t points. Setting $t = \frac{2}{\epsilon^2}\log\left(\frac{2}{\delta}\right)$ ensures that with probability atleast $1-\delta$, the sample median will be ϵ-approximate. Thus the space complexity of our algorithm is $t \cdot \log(|\Sigma|)$ where $|\Sigma|$ is the size of the sample space.
In order to argue about its correctness, let us partition the set S into three parts:

$$\begin{aligned} S_L &:= \{x | \text{pos}(x) \leq \frac{m}{2} - \epsilon m\} \\ S_M &:= \{\{x | \text{pos}(x) \geq \frac{m}{2} - \epsilon m \quad \& \quad \text{pos}(x) \leq \frac{m}{2} + \epsilon m\} \\ S_R &:= \{x | \text{pos}(x) \geq \frac{m}{2} + \epsilon m\}, \end{aligned}$$

where S_L denotes the set of points to the left of the approximate median and S_R denotes the set of points to the right of approximate median. Note that our algorithm will return a correct solution if the number of points from S_R and S_L is less than $t/2$ in the t samples that were collected. We will now show that these are low probability events. Let us start with S_L, the proof for S_R follows similarly.
Let Z_i denote a random variable such that $Z_i = 1$ if the i^{th} sample is from S_L and $Z_i = 0$ otherwise. We are interested in showing that $\Pr(\sum_{i=1}^{t} Z_i > t/2)$ is small. Note that $\mathbb{E}[Z_i] = 1/2 - \epsilon$ since the probability that a uniformly sampled element is in S_L is $1/2 - \epsilon$. Using the Hoeffding bound from class, we have:

$$\begin{aligned} \Pr\left(\sum_{i=1}^{t} Z_i > t/2\right) &= \Pr\left(\frac{1}{t}\sum_{i=1}^{t} Z_i - \left[\frac{1}{2} - \epsilon\right] > \epsilon\right) \\ &\leq \exp\left(-2\epsilon^2 t\right), \end{aligned}$$

and therefore setting the value of $t = \frac{1}{2\epsilon^2} \log\left(\frac{2}{\delta}\right)$ gives us that with probability atleast $1-\delta/2$, the number of elements from S_L will be less than $t/2$. We can argue similarly for S_R and combining those two completes our proof.

7 Project

Recall the RUDRATA CYCLE problem: given a graph, does there exist a cycle that passes through each vertex exactly once? In this problem, we will construct a reduction from RUDRATA CYCLE to DTH (Drive the TAs Home).

Note: Use the version of DTH as stated in the spec, i.e. it takes the driver $\frac{2}{3}$ of the energy of a TA to travel the same unit of distance.

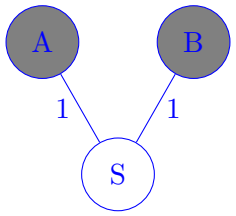
(a) Create a DTH instance on 3 vertices where there is an optimal solution in which Professor Rao's car does not move. Prove that this solution is optimal for your instance.

(b) Use the gadget you created in part (a) to give a reduction from Rudrata Cycle to DTH.

Hint: Show that there exists a Rudrata Cycle in the original graph if and only if there exists a solution to DTH with cost $\frac{8}{3}n$ in a modified graph. Specifically, the only solutions in your instance of DTH with cost $\frac{8}{3}n$ should correspond to Rudrata cycles in the original graph.

(c) Based on what you've shown, what can you conclude about the complexity classes DTH is in?

Solution:



(a) S is the starting vertex. Gray vertices are homes. All edge weights are 1.

Having the two TAs walk from S to either A or B would incur a cost of 2, whereas driving from S to the other two vertices and back would have cost $\frac{8}{3}$, so the optimal solution is to keep the car at S and have TAs walk to A and B.

(b) **Description of Reduction:**

Given a graph G as an instance of the Rudrata Cycle problem, convert to a graph G' by adding 2 vertices u, w for each vertex v in G, and add the edges (u, v), (w, v), each with weight 1. All edge weights in G' should be 1. Make each of the extra vertices u and w a TA's home, for a total of $2|V|$ homes. Let n be the number of vertices in G. There is a Rudrata Cycle in G if and only if there is a Drive the TAs Home solution in G' with

cost $\frac{8}{3}n$.

If there is a solution to Drive the TAs home on the modified graph whose cost is $\frac{8}{3}n$, then there exists a Rudrata Cycle in the original graph:

Since the weights of the additional edges are 1, the cost for any student to get from one of the original vertices v to an additional vertex u is 1, and the cost for the car to drive along (v, u) and back along (u, v) is $\frac{4}{3}$. Since there are two homes per vertex v, the cost for the driver to go to both homes from v is $\frac{8}{3}$. If the students instead walked from v to their respective homes, the cost would be 2, which is less than $\frac{8}{3}$. Thus, the car will never go to any homes.

Now we will consider what happens if we drop off the two students at another vertex v' that was in the original graph G but is not connected to their homes. Without loss of generality, we can assume that the students still get dropped off together as they must go through v to get to their homes. Thus the additional cost of dropping off the students at v' is 2 times the number of edges from v' to v, for a total of at least 2. Therefore the total cost of the solution is at least $\frac{2}{3}k + 2n + 2(n - k)$, where k is the number of original vertices (vertices from the original graph G) visited by the driver. The first term corresponds to the cost of the driver, the second is the cost of two students walking home from every original vertex, and the third is a lower bound on the total additional cost of students dropped off at a vertex $v' \neq v$. This is larger than $\frac{8}{3}n$ for all values of k less than n. The minimum cost of a solution to Drive the TAs Home on graph G' is $\frac{8}{3}n$, which occurs if there is a Rudrata path in the original graph G (revisiting vertices would only add to the total cost).

If there is a Rudrata Cycle in the original graph, then there exists a solution to Drive the TAs home whose cost is $\frac{8}{3}n$:

Treat the Rudrata cycle as the path of the car in the transformed graph. Every two students will get off at their respective vertex v, and thus add a cost of $2n$, as shown below. The cost that the driver incurs is $\frac{2}{3}n$. Thus the total cost of the solution to Drive the TAs Home is $2n + \frac{2}{3}n = \frac{8}{3}n$.

(c) NP-Hard.