CME 305: Discrete Mathematics and Algorithms

Instructor: Reza Zadeh (rezab@stanford.edu)

HW#4 – Due at the beginning of class Thursday 03/12/15

1. Let $G = (V, E)$ be a c-edge connected graph. In other words, assume that the size of minimum cut in G is at least c. Construct a graph $G'(V, E')$ by sampling each edge of G with probability p independently at random and reweighing each edge with weight $1/p$. Suppose $c > \log n$, and ϵ is such that $\frac{\log(n)}{c\epsilon^2} \leq 1$. Show that as long as $p \geq \frac{\log(n)}{c\epsilon^2}$, with high probability the size of every cut in G' is within $(1 \pm \epsilon)$ of the cut in the original graph G.

   Solution: To see how this naive random sampling performs, we will sample each edge with the same probability p, and give weight $1/p$ to each edge in the sparse graph H. With these weights, each edge $e \in E$ will have expected contribution exactly 1 to any cut, and thus the expected weight of any cut in H will match that of G. It remains to see how many samples we need to have cut equivalence between G and H with high probability.

   Consider a particular cut $S \subseteq V$. If it has c edges crossing it in G, the expected weight of edges crossing it in the new graph H is also c. Denote the total weight of edges between S and $V \setminus S$ by $f_G(S) = c$, and we have the following concentration result due to Chernoff:

$$\text{P}[|f_H(S) - c| \geq \epsilon c] \leq 2e^{-\epsilon^2 pc/2}$$

   In particular, picking $p = 2\frac{t\log n}{\epsilon^2 c}$ (for t set a little later) will make the RHS of the above less than $2e^{-t\log n}$. To bound the probability there exists a bad cut, we apply union bound using Karger's cut-counting theorem, which says that if G has a min-cut of size c, then the number of cuts of value αc is at most $n^{2\alpha}$. Thus

$$\begin{aligned}\text{P}[\exists S \text{ s.t. } |f_H(S) - c| \geq \epsilon c] &\leq \sum_{\alpha=1}^{n^2} n^{2\alpha} 2e^{-\alpha\log(n)t} \\ &= 2\sum_{\alpha=1}^{n^2} e^{2\alpha\log(n) - \alpha\log(n)t} \\ &= 2\sum_{\alpha=1}^{n^2} e^{\alpha\log(n)(2-t)}\end{aligned}$$

   We pick $t = 5$, continuing:

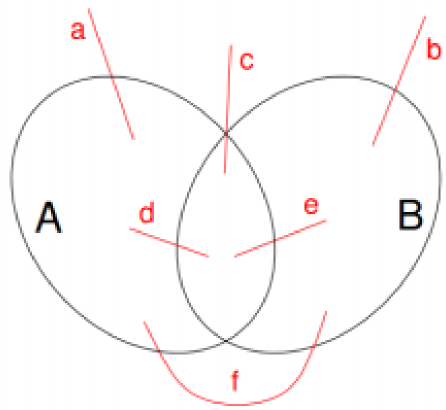
$$= 2\sum_{\alpha=1}^{n^2} e^{-3\alpha\log(n)} \leq 2\sum_{\alpha=1}^{n^2} \frac{1}{n^3} \leq \frac{2}{n}$$

   Which goes to zero. Note that increasing t by 1 increases by 1 the polynomial degree showing up in the denominator.

2. Let V be a finite set. A function $f: 2^V \to R$ is submodular iff for any $A, B \subseteq V$, we have

$$f(A \cap B) + f(A \cup B) \leq f(A) + f(B)$$

Now consider a graph with nodes V. For any set of vertices $S \subseteq V$ let $f(S)$ denote the number of edges $e = (u, v)$ such that $u \in S$ and $v \in V - S$. Prove that f is submodular.



Solution. To see this, notice that $f(A)+f(B) = a+b+2c+d+e+2f$, for any arbitrary A and B, and a, b, c, d, e, f are as shown in the figure. Here, a (for example) represents the total capacity of edges with one endpoint in A and the other in $V - (A \cup B)$. Also notice that $f(A \cap B) + f(A \cup B) = a + b + 2c + d + e$, and since all values are positive, we see that $f(A) + f(B) \geq f(A \cap B) + f(A \cup B)$, satisfying the definition. Thanks to [1] for the figure.

It is worth noting all submodular functions can be minimized in polynomial time, and that many discrete optimization problems can be recast as submodular optimization, with the Minimum Cut problem being a famous example.

3. A square integer matrix A is **unimodular** if and only if its determinant is -1 or 1. A matrix (not necessarily square) M is **totally unimodular** iff every square submatrix has determinant 1, -1, or 0, i.e. every non-singular square submatrix is unimodular.

Show that for a linear program with totally unimodular constraint matrix M and integral right-hand side c, all extreme points must be integral.

Solution: Suppose the LP has the form:

$$\begin{aligned} \max_x \quad & c^T x \\ \text{subject to} \quad & Ax \leq b \end{aligned}$$

Where A is totally unimodular and b is integral. Let v be a vertex solution. Since v is a vertex, several inequalities of $Ax \leq b$ are equalities. So therefore we can derive a submatrix A' of A (and a 'submatrix' b' of b) such that A' is a full rank square matrix and

[1] `http://www.cs.illinois.edu/class/sp10/cs598csc/Lectures/Lecture6.pdf`

$A'v = b'$. So $v = (A')^{-1}b'$. By Cramer's rule v is given by $v_i = \det(A'_i)/\det(A')$, where A'_i is the matrix where the ith column replaced by b'. Since A is totally unimodular, A' is totally unimodular. Note that we can write:

$$v_i = \det(A'_i)/\det(A') = (b_1\det(A'_1) - b_2\det(A'_2) + ...)/\det(A')$$

So since b is integer and A' is full rank and totally unimodular, v_i is integer.

4. We are given n jobs that each take one unit of processing time. All jobs are available at time 0, and job j has a profit of c_j and a deadline d_j. The profit for job j will only be earned if the job completes by time d_j. The problem is to find an ordering of the jobs that maximizes the total profit. First, prove that if a subset of the jobs can be completed on time, then they can also be completed on time if they are scheduled in the order of their deadlines. Now, let $E = \{1, \ldots, n\}$ and let

$$I = \{J \subseteq E : J \text{ can be completed on time } \}$$

Prove that $M = (E, I)$ is a matroid and describe how to find an optimal ordering for the jobs.

Solution: For $J \in I$, let $N_t(J)$ be the number of jobs in J whose deadline is t or earlier. Note that $N_0(J) = 0$ and $N_t(J) \leq t$, otherwise, there is no way to make a schedule with no late jobs, because there are more than t tasks to finish before time t; thus, there is no way to "get stuck" when scheduling the tasks in order of monotonically increasing deadlines.

Every subset of an independent set of tasks is certainly independent. To prove the exchange property, suppose that B and A are independent sets of tasks and $|B| > |A|$. Let k be the largest t such that $N_t(B) \leq N_t(A)$ (such a t exists, since $N_0(A) = N_0(B) = 0$). Since $N_n(B) = |B|$ and $N_n(A) = |A|$, but $|B| > |A|$, we must have that $k < n$ and that $N_j(B) > N_j(A)$ for all j in the range $k + 1 \leq j \leq n$. Therefore, B contains more tasks with deadline $k + 1$ than A does. Let job j in $B \setminus A$ have deadline $k + 1$. Let $A' = A \cup \{j\}$. For $0 \leq t \leq k$, we have $N_t(A') = N_t(A) \leq t$, since A is independent. For $k < t \leq n$, we have $N_t(A') \leq N_t(B) \leq t$, since B is independent. Therefore, A' is independent. Hence I is a matroid.

By applying the greedy algorithm (by sorting the price in decreasing order and adding jobs one by one and maintaining the set of selected jobs in I) on the matriod, we get the greedy algorithm to find the optimal schedule.

5. Given a list of personnel (n persons) and of list of k vacation periods, each period spanning several contiguous vacation days. Let D_j be the set of days included in the jth vacation period. You need to produce a schedule satisfying:

   - For a given parameter c, each tech support person should be assigned to work at most c vacation days total.
   - For each vacation period j, each person should be assigned to work at most one of the days during the period.

- Each vacation day should be assigned a single tech support person.
- For each person, only certain vacation periods are viable.

Describe a polynomial time algorithm to generate an assignment or output that no assignment exists. Prove correctness.

Solution: Let $V_1 = \{p_1, ..., p_n\}$, $V_2 = \{D_1, ..., D_k\}$ and $V_3 = \{y_{11}, ..., y_{1n_1}, ..., y_{k1}, ..., y_{kn_k}\}$, where V_1 is the list of personel, V_2 is the list of vacation periods and $y_{i1}, ..., y_{in_i}$ is the set of contiguous vacation days in D_i. By adding two nodes (s, t), we construct a network flow G as follows,

- $s \to p_i$, $c(s, p_i) = c$ for any i
- p_i connects to all the viable vacation periods with $c(p_i, D_j) = 1$
- $D_i \to y_{ij}$ for any i, j with $c(p_i, y_{ij}) = 1$
- $y_{ij} \to t$ with $c(y_{ij}, t) = 1$

where $c(*, *)$ is the capacity function. Assignment exists iff the maximum flow of G is $\sum_i n_i$.

By Ford-Fulkerson algorithm, the maximum flow of G can be solved in polynomial time.

6. Let G be a graph with n nodes and an independent set of size $2n/3$. Give a polynomial time algorithm to find an independent set of size $n/3$ or greater - find a 1/2-approximation to the independent set in this graph.

   Solution: We do this by converting the problem to an instance of VERTEX COVER, applying an approximation algorithm we know for this problem, and finally realize that the vertex cover found by our approximation corresponds to an independent set of at least the required size.

   Let S be the independent set of size $2n/3$ in the graph. Consider the set $T = V - S$, the complement of S in G. For every edge (u, v) in G, we see that either u or v must lie in T– if neither u nor v was in T then both u and v would be in S, implying that our independent st S contains the edge (u, v). Thus, we see that T is a vertex cover of the graph, which has size $n - 2n/3 = n/3$.

   With this in mind, consider the problem of approximating the minimum vertex cover in G. We recall that we can achieve a $2-$approximation for this problem via the linear programming relaxation covered in class. Thus, if the optimal vertex cover has size OPT_{VC}, we can find a vertex cover of size at most $2OPT_{VC}$. But we see from above that G contains a vertex cover of size $n/3$. Thus, we have $OPT_{VC} \leq n/3$, and so applying the LP-relaxation algorithm to G will afford a vertex cover of at most $2n/3$ nodes. Let this found vertex cover be T'.

   Finally, consider the set $S' = V - T'$, the complement of T' in G. For every edge (u, v) in G, we see that one of u and v must lie outside of S', as otherwise both u and v would lie in S' and thus neither u nor v would lie in T'. Thus, in this case we would have that the edge (u, v) would have neither of its endpoints inside of T', a contradiction. So, for every edge (u, v) in the graph, S' cannot contain both u and v: thus, S' is an independent set. As $|T'| \leq 2n/3$, we have $|S'| = |V| - |T'| \geq n - 2n/3 = n/3$– and so we have found an independent set of size at least $n/3$, as desired.

7. The *directed* cut size is the number of outgoing edges from a cut S. The directed MAX-CUT problem asks for the cut with maximum directed cut size. Give a 1/4 approximation algorithm for this problem.

   Solution: Consider the following modification to the greedy algorithm for undirected MAX-CUT covered in class: Initialize two sets $A = V, B = \varnothing$, and consider the cuts defined by A and B. If there exists a vertex v such that moving it from one set to the other would strictly increase the cut size of A **plus** the cut size of B, move it, and continue doing this until no such vertex v can be found. Compute the cuts sizes of A and B, and return the larger of the two. This runs in polynomial time as it costs $O(m)$ time to compute the value of a given cut, we do this at most n times to find a satisfactory vertex v, and since the maximum cut value is m and each swap we perform is guaranteed to increase the cut size by at least 1, we do at most m swaps before returning our approximate max cut. Thus, this algorithm runs in time $O(nm^2)$.

   We will now show it achieves the desired approximation ratio. We note trivially that $OPT \leq m$ for this problem. This will serve as our handle on OPT. Let $\delta_{X,Y}$ be the number of edges crossing out of X into Y. With this, we see that the cut size of X is just $\delta_{X,V-X}$. Now, as $B = V - A$ by the algorithm, we see that the size of the cut defined by A is $\delta_{A,B}$, and the size of the cut defined by B is $\delta_{B,A}$. We claim that $\delta_{A,B} + \delta_{B,A} \geq 2\delta_{A,A}$. To prove this, we consider what happens when we take some node $v \in A$ and move it into B. Let $\delta_{in}^{X}(v)$ be the number of edges pointing *into* v from some set X and let $\delta_{out}^{X}(v)$ be the number of edges pointing *out* of v into X. With this, we see that when we move v from A to B, the cut defined by A loses $\delta_{out}^{B}(v)$ edges (the edges originally pointing out of v now point within B and do not cross the cut) but gains $\delta_{in}^{A}(v)$ (the edges pointing into v from A will now cross the cut). Similarly, if we move v from A to B the cut from B to A will gain $\delta_{out}^{A}(v)$ edges but lose $\delta_{in}^{B}(v)$ edges. Thus moving v will change the sum of the cut sizes by $\delta_{in}^{A}(v) + \delta_{out}^{A}(v) - \delta_{out}^{B}(v) - \delta_{in}^{B}(v)$.

   Since we know that moving single nodes across the cut cannot increase this sum of cut sizes, we must have that $\delta_{in}^{A}(v) + \delta_{out}^{A}(v) - \delta_{out}^{B}(v) - \delta_{in}^{B}v \leq 0$, for every $v \in A$. If we sum these inequalities over all v in A, we see that the first two terms will each count the number of edges internal to A (or, $\delta_{A,A}$), the third term counts the number of edges crossing from A to B (or $\delta_{A,B}$), and the fourth term counts the number of edges crossing from B to A (or $\delta_{B,A}$). With this we see that the sets A and B found by our algorithm must satisfy $2\delta_{A,A} \leq \delta_{A,B} + \delta_{B,A}$.

   By the same reasoning, we also see that $\delta_{B,A} + \delta_{A,B} \geq 2\delta_{B,B}$. Since every edge in the graph is directed, we see that regardless of the cut found the edge is counted in exactly one of $\delta_{A,A}, \delta_{A,B}, \delta_{B,A}, \delta_{B,B}$. Thus, $\delta_{A,A} + \delta_{A,B} + \delta_{B,A} + \delta_{B,B} = m$. Finally, since our outputted cut will have size $max(\delta_{A,B}, \delta_{B,A})$, we have that $APX = max(\delta_{A,B}, \delta_{B,A}) \geq (\delta_{A,B} + \delta_{B,A})/2 \geq (\delta_{A,A} + \delta_{A,B} + \delta_{B,A} + \delta_{B,B})/4 = m/4 \geq OPT/4$– our algorithm finds a 1/4-approximation as desired.

(**Note:** A more intricate greedy algorithm achieves a .5 approximation for this problem, and the original Goemans-Williamson paper also provides an algorithm with a .796... approximation ratio. The current best algorithm for the directed MAX-CUT problem achieves a .859... approximation, using extensions to the semidefinite programming technique of Goemans-Williamson.)

8. Online social networks carry a huge potential for online advertising. After a recent controversy, a popular social networking platform does not allow advertisers to target the users individually. However, it is allowed to run ads on user communities.

   Formally, let X be the set of all users on a social network, and $S_1, S_2, \ldots, S_m$ be subsets of X, where each S_i represents a user community. Notice that a user can belong to several communities. Suppose the advertiser can afford placing ads on at most k communities. The goal is to show the ads to as many users as possible, i.e. to find $S_{i_1}, S_{i_2}, \ldots, S_{i_k}$ such that $|\cup_{j=1}^{k} S_{i_j}|$ is maximized.

   Unfortunately, this problem is NP-hard and therefore we are interested in designing efficient approximation algorithms to solve it. Consider the following greedy approach: pick the k communities one at a time, and in each iteration pick the community that contains the largest number of users that have not been covered yet. In other words, choose the community that maximizes the current coverage. Show that this greedy approach yields at least $1 - (1 - 1/k)^k > 1 - 1/e$ fraction of the optimal solution.

   Hint: Let x_i denote the number of new elements covered by the algorithm in the i-th set that it picks. Also, let $y_i = \sum_{j=1}^{i} x_j$, and $z_i = OPT - y_i$. Show $x_{i+1} \geq z_i/k$ and prove by induction that $z_i \leq (1 - 1/k)^i OPT$.

   Solution: Optimal solution covers OPT elements at k iterations. That means, at each iteration there should be some sets whose size is greater than or equal to $1/k$ of the remaining uncovered elements, i.e., z_i/k. If we were choosing the optimal sets each time, we know that at each iteration, we would be able to choose a new set that has at least $1/k$ of the uncovered elements. So during the greedy algorithm, when we are choosing the next set with the maximum number of uncovered elements, there must be some set with at least $1/k$ of the uncovered elements in OPT we choose, so we have $x_{i+1} \geq z_i/k$.

   In the first step, we have $x_1 \geq OPT/k$ (using the same arguments above). Note $y_1 = x_1$, we have

$$OPT - y_1 = OPT - x_1 \leq OPT - OPT/k = (1 - 1/k)OPT$$

   Now, for inductive hypothesis assume $z_i \leq (1 - 1/k)^i OPT$ is true, for $i + 1$,

$$z_{i+1} = z_i - x_{i+1} \leq z_i - z_i/k = z_i(1 - 1/k) = (1 - 1/k)^{i+1} OPT$$

   Note that $y_k = \sum_{i=1}^{k} x_i = OPT - z_k \leq OPT - (1 - 1/k)^k OPT \leq (1 - 1/e)OPT$.