CME 305: Discrete Mathematics and Algorithms

Instructor: Reza Zadeh (rezab@stanford.edu)

HW#2 – Due at the beginning of class Thursday 02/05/15

1. **(Kleinberg Tardos 7.27)** Some of your friends with jobs out West decide they really need some extra time each day to sit in front of their laptops, and the morning commute from Woodside to Palo Alto seems like the only option. So they decide to carpool to work. Unfortunately, they all hate to drive, so they want to make sure that any carpool arrangement they agree upon is fair and doesn't overload any individual with too much driving. Some sort of simple round-robin scheme is out, because none of them goes to work every day, and so the subset of them in the car varies from day to day.

   Here's one way to define *fairness*. Let the people be labeled $S = \{p_1, \ldots, p_k\}$. We say that the *total driving obligation* of p_j over a set of days is the expected number of times that p_j would have driven, had a driver been chosen uniformly at random from among the people going to work each day. More concretely, suppose the carpool plan lasts for d days, and on the i^{th} day a subset $S_i \subseteq S$ of the people go to work. Then the above definition of the total driving obligation Δ_j for p_j can be written as $\Delta_j = \sum_{i:p_j \in S_i} \frac{1}{|S_i|}$. Ideally, we'd like to require that p_j drives at most Δ_j times; unfortunately, Δ_j may not be an integer.

   So let's say that a *driving schedule* is a choice of a driver for each day — that is, a sequence $p_{i_1}, p_{i_2}, \ldots, p_{i_d}$ with $p_{i_t} \in S_t$ — and that a *fair driving schedule* is one in which each p_j is chosen as the driver on at most $\lceil \Delta_j \rceil$ days.

   (a) Prove that for any sequence of sets $S_1, \ldots, S_d$, there exists a fair driving schedule.

   (b) Give an algorithm to compute a fair driving schedule with running time polynomial in k and d.

   Solution: We convert the problem into a network flow problem. First we construct a graph as follows. We denote the vertex p_i as the i-th driver. Moreover we denote the vertex q_j as the j-th day. If p_i can drive on the j-th day, we simply draw a directed edge from p_i to q_j of capacity 1. Finally we draw a source s which connects each p_i with capacity $\lceil \Delta_i \rceil$ and a sink which connects each q_j with capacity 1. It is easy to find that computing a fair driving schedule is equivalent to computing the maximum flow problem. The only thing we need to do is to prove that the value of the maximum flow is d.

   First of all, it is obvious that for any flow f, $|f| \leq d$. Thus if we are able to find a flow f with $|f| = d$, we are done. This is easy to achieve. Consider the following flow.

   $$f_{p_i q_j} = \frac{1}{|S_j|}, \qquad f_{sp_i} = \sum_{j:p_i \in S_j} \frac{1}{|S_j|} \leq \lceil \Delta_i \rceil, \qquad f_{q_j t} = 1.$$

   This flow satisfies all the constraints and have value n. Thus there exists a fair schedule. For computing it, we simply adopt the Ford algorithm.

2. Recall Karger's algorithm for the global min-cut problem. In this problem we modify the algorithm to improve its running time.

(a) Prove that if we stop the original Karger's algorithm when the remaining number of vertices is

$$\max\left\{\lceil 1+n/\sqrt{2}\rceil, 2\right\},$$

the probability that we have contracted an edge in the min-cut is less than 1/2. Lets call this procedure *Partial Karger*.

Solution: Denote A_k as the event that we do not contract an edge in the min cut. Suppose we stop at the k-th step, then:

$$\mathrm{P}(\cap_{i=1}^{k} A_i) \geq \frac{(n-k)(n-k-1)}{n(n-1)}$$

If we set $k = n - \left\lceil n/\sqrt(2)\right\rceil - 1$, we have:

$$\mathrm{P}(\cap_{i=1}^{n-\left\lceil n/\sqrt(2)\right\rceil-1} A_i) \geq \frac{\left\lceil n/\sqrt(2)\right\rceil (\left\lceil n/\sqrt(2)\right\rceil + 1)}{n(n-1)}$$
$$\geq \frac{n/\sqrt{2}(n/\sqrt{2}+1)}{n(n-1)} \geq \frac{1}{2}$$

(b) Now suppose we apply *Partial Karger* to two copies of G to produce graphs G_1 and G_2. We then recursively apply these steps to G_1 and G_2 and so on until each recursive call returns a graph on two vertices. If $r(n)$ is the running time of this process as a function of the number of vertices n of G, derive a recursive equation for $r(n)$ and solve it to obtain an explicit expression for the running time (you may use $O(\cdot)$ notation to simplify your recursive equation).

Solution: The operation cost for contracting a single edge is $O(n)$. Thus the operation cost for partial Kager is $O(n^2)$. By recursion we have:

$$r(n) = 2r(n/\sqrt{2}) + O(n^2)$$

By Master's Theorem we obtain $r(n) = O(n^2 \log n)$.

(c) Show that the algorithm in part (b) produces $O(n^2)$ contracted graphs on two vertices each. Prove that the probability that at least one of them contains a global min-cut is at least $1/\log(n)$ up to a multiplicative constant.

Hint: Think of the recursion as a binary tree with paths leading to the $O(n^2)$ leaves representing the two-vertex contracted graphs.

Solution: By using partial Karger's algorithm, we obtain graphs G_1, G_2 from the original graph G. Here G_1, G_2 have $\left\lceil n/\sqrt(2)\right\rceil$ vertices. We continue using partial Karger's algorithm, so that G_1, G_2 keep branching. In the end we get a binary tree. The height of the tree is $\log_{\sqrt{2}} n$. Thus the total number of leaves is $O(2^{\log_{\sqrt{2}} n} = O(n^2)$. Now we proceed to prove that the probability that at least one of the leaves contains a global min cut is greater than $c/\log n$. We denote such probability as $f(n)$. Moreover we denote p as the probability in part (a). We know from part (a) that $p < 1/2$.

We consider $1 - f(n)$, which is the probability that none of the leaves contains a global min cut. Since the algorithm G branches to G_1 and G_2. By independence we only consider the probability that none of G_1's leaves contains a global min cut. There are two cases that can make this happen. (1) We contracted an edge in the min cut when we derive G_1 from G using Karger's algorithm, which has probability p. (2) We did not contract an edge in the min cut when we derive G_1 from G, but we unfortunately contracted a min-cut-edge in the following steps, which has probability $(1-p)(1-f(nsqrt2)$. Thus we have:

$$\begin{aligned} 1 - f(n) &= (p + (1-p)(1 - f(n/\sqrt{2})))^2 = (1 - (1-p)f(n/\sqrt{2}))^2 \\ &\le (1 - \frac{1}{2}f(n/\sqrt{2}))^2 = 1 - f(n/\sqrt{2}) + \frac{1}{4}f^2(n/\sqrt{2}) \end{aligned}$$

Thus, $f(n) \ge f(n/\sqrt{2}) - \frac{1}{4}f^2(n/\sqrt{2})$. We prove by induction that $f(n) \ge c/\text{log}n$ for some small c. Here we take the logarithm under base $\sqrt{2}$. Suppose this holds for $k \ge n$, then for $k = n$, we have:

$$f(n) \ge f(n/\sqrt{2}) - \frac{1}{4}f^2(n/\sqrt{2}) \ge \frac{c}{\text{log}n - 1/2} - \frac{c^2}{4(\text{log}n - 1/2)^2}$$

We want to prove that the right hand side is less than $c/\text{log}n$, that is:

$$\frac{c}{\text{log}n - 1/2} - \frac{c^2}{4(\text{log}n - 1/2)^2} \le \frac{c}{\text{log}n}$$

This is equivalent to $\frac{2}{\text{log}n} \le c$. If we choose c to be less than 2, then this indeed holds. Thus we completed the induction process.

(d) Compare the running time of the above algorithm to Karger's original given the same probability of failure.

Solution: For the partial Karger's algorithm, the success probability is $c/\text{log}n$. Thus we need to run it $\text{log}n$ times to achieve constant success rate. The total run time is $O(n^2\text{log}n)$ times $O(\text{log}n)$, which is $O(n^2\text{log}^2 n)$ time. For traditional Karger's algorithm, the total run time is $O(n^2 m\text{log}n)$. Obviously, partial Karger's algorithms is significantly smaller.

3. An independent set in a graph is a set of vertices with no edges connecting them. Let G be a graph with $nd/2$ edges ($d > 1$), and consider the following probabilistic experiment for finding an independent set in G: delete each vertex of G (and all its incident edges) independently with probability $1 - 1/d$.

(a) Compute the expected number of vertices and edges that remain after the deletion process. Now imagine deleting one endpoints of each remaining edge.

Solution. Each node survives with probability $1/d$. Thus the expected number of nodes is n/d. Each edge survives with probability $1/d^2$ (both its ends must survive independently). Thus the expected number of edges is $nd/2 \times 1/d^2 = n/2d$.

(b) From this, infer that there is an independent set with at least $n/2d$ vertices in any graph with on n vertices with $nd/2$ edges.

Solution. After applying this sampling, we create an independent set as follows: for each edge in the resulting graph, delete one of the endpoints. After doing this for each edge, none of the remaining vertices are connected by any edges, i.e. form an independent set. If $G' = (V', E')$ is the graph we obtain after sampling, we expect the size of the independent set to be

$$E[\text{size of independent set}] = E[|V'| - |E'|] = n/d - n/2d = n/2d$$

Since there will be at least one outcome with a value equal to (or greater than) the expectation, by the probabilistic method there must be an independent set of size $\geq n/2d$.

4. Prove that a graph can only have at most $\binom{n}{2}$ different cuts that realize the global minimum cut value.

Solution: Suppose we run Karger's min cut algorithm we saw in class. Let x_i be the event that the algorithm returns the i^{th} global min cut. Suppose there are s different min cuts, then the probabilities of realizing each in the algorithm will be disjoint (all end with different sets of nodes at the conclusion of the algorithm). We saw in class that for a given global min cut, the probability of returning that cut is $\geq \frac{2}{n(n-1)}$. So we have:

$$\text{P[returning a global min cut]} = \text{P}[\cup_{i=1}^{s} x_i] = \sum_{i=1}^{s} \text{P}[x_i] \geq s\frac{2}{n(n-1)}$$

We also have that the probability of returning a global min cut is ≤ 1, so we need the above to be upper bounded by 1, which means $s \leq \frac{n(n-1)}{2}$.

5. Exhibit a graph $G = (V, E)$ where there are an exponential (in $|V| = n$, the number of nodes) number of minimum cuts between a particular pair of vertices. Do this by constructing a family of graphs parameterized on n and give a pair of vertices s, t such that there are exponentially many minimum cuts between s and t.

Solution. For $n = 3$, we simply have a path of length 2 between the two ends s and t. For $n > 3$, each new vertex will be connected to s and t (and nothing else) creating an additional path of length 2 between s and t.

For general n, to separate s and t, we must cut one edge along every edge-disjoint path between them. There are $n - 2$ paths between s and t, each of length 2. So we have $n - 2$ binary choices, giving 2^{n-2} different minimum cuts.

6. Exhibit a directed graph that has cover time exponentially large in the number of nodes. Contrast this with the cover time of undirected graphs discussed in class.

Solution: Consider the digraph consisting of a directed cycle $(1, 2, ..., n, 1)$, and edges $(j, 1)$, from vertices $j = 2, ..., n - 1$. Starting from vertex 1, the expected time for a random walk to reach vertex n is $3 \times 2^{n-2} - 2$; in fact, let T be the hitting time at state n. $\mathrm{E}[T|x_0 = n] = 0$, $\mathrm{E}[T|x_0 = 1] = \mathrm{E}[T|x_0 = 2] + 1$, $\mathrm{E}[T|x_0 = i] = (1/2)\mathrm{E}[T|x_0 = 1] + (1/2)\mathrm{E}[T|x_0 = i + 1] + 1$ for $i = 2, ..., n - 1$. Solving this linear system, we get $\mathrm{E}[T|x_0 = 1] = 3 \times 2^{n-2} - 2$, $\mathrm{E}[T|x_0 = i] = 3(2^{n-2} - 2^{i-2})$ for $i = 2, ..., n$.

7. You are the sole proprietor of the company *Widgets Inc.*, a supplier of quality widgets to a large corporation in your country. You have just received a big order of w widgets from the large corporation, and are preparing a shipment from your facility, s, to their headquarters, t. Due to various idiosyncrasies in your country's postal service, the fastest way to deliver packages is to route them by hand and choose which postal hubs the package will stop in before it reaches its final destination. In addition, occasionally a hub's computer will crash and massively delay all shipments exiting the city. Since you have been burned by this before, you decide that you want to minimize the number of widgets that are sent through any given city. However, since this is such a lucrative contract for you, you also want to send as many widgets as you possibly can. Thus, if we model our country's postal service as a graph G with a set of hubs and cities V and a set of connections between hubs E, and assume that every edge in our graph has infinite capacity (in other words, we can send as many widgets across a given connection as we like), we would like to find the smallest number k such that we can route all w widgets from s to t while ensuring that no city handles more than k widgets. Find a polynomial time algorithm to do so.

   Solution. To solve this problem, we note that if we had a polynomial time algorithm to find a satisfactory routing for fixed k (or return that none exists), we could find the smallest k with a satisfactory routing by binary searching for k. Thus, we solve the problem of finding a satisfactory routing for fixed k. We modify our graph slightly. For every hub h, define two new nodes h_{in} and h_{out}. Have every connection pointing into h now point into h_{in} and have every connection ponting out of h point out of h_{out}. Finally, add an edge of weight k between h_{in} and h_{out}. We do this because the weight k edge effectively enforces that hub h handles at most k widgets: if we tried to route $k + 1$ widgets into h_{in}, only k of them would be allowed to exit h_{out}. After doing this, we compute a maximum flow between s and t. If the maximum flow has size less than w, then we immediately know that no satisfactory routing plan can exist. If instead the maximum flow is greater than w, we can simply choose to route our widgets along this flow and get all w widgets to the target node t and ensure that no hub sees more than k widgets, as desired. This runs in polynomial time since solving maximum flows clearly takes polynomial time and we only do a polynomial amount of extra work to convert the flow into a solution. To find the smallest such k, we note that k is bounded by w (we are only routing w widgets anyway), and thus binary searching for the smallest k only requires $log(w)$ calls to this algorithm. Thus, the final procedure still takes polynomial time.

8. Compute the cover time of a Hamiltonian cycle with n vertices.

Solution. First of all $C(G) \leq 2m(n-1) = 2n(n-1)$, so $C(G)$ is $O(n^2)$. Now we compute the maximum resistance in the graph. For two points that are x and $n-x$ edges away from each other, the resistance between these two points are $R = \frac{1}{1/x+1/(n-x)} = x(1-x/n)$. The maximum of this function is achieved at $x = [n/2]$ and $R_{max} = \theta(n)$. Then $C(G) \geq mR_{max}$ implies $C(G)$ is $\Omega(n^2)$. Hence $C(G)$ is $\Theta(n^2)$.

9. Suppose we have a $2n \times 2n$ ($n \geq 2$) table where each cell is filled with an integer in $\{1, 2, 3, ..., 2n^2\}$. Moreover, each integer shows up exactly twice. Show that one can pick $2n$ cells that satisfy all the following conditions: (1). all the numbers written in these cells are distinct; (2). in each row exactly one cell is picked out; (3) in each column exactly one cell is picked out.

   Solution. Take a random permutation π of $\{1, 2, 3, ..., 2n\}$, pick out $\pi(i)$-th cell from row i. Any $2n$ cells chosen this way satisfy condition (2) and (3). We show that the probability of these cells satisfying condition (1) is positive: For any $j \in \{1, 2, 3, ..., 2n^2\}$, the probability of picking out two j's in our $2n$ cells is: 0 if the two j's are in the same row or column; $\frac{1}{2n} \times \frac{1}{2n-1}$ if the two j's are in different rows and columns. By union bound, the probability of satisfying condition (1) $\geq 1 - 2n^2 \times \frac{1}{2n} \times \frac{1}{2n-1} > 0$ when $n \geq 2$. Therefore there is a way of choosing $2n$ cells that satisfies all the three required conditions. □

10. After your success selling widgets, you have grown your business and now sell widgets of many different sizes and shapes. Shipping these widgets requires special boxes which you have custom built by a supplier. Unfortunately, having the supplier ship you these boxes is rather expensive, due to a large customs tax imposed on every package he sends you. You notice that some of the smaller of the boxes you need fit inside some of the larger ones, and seeing an opportunity to save money, you ask your supplier to ship smaller boxes inside of larger ones. He agrees, as long as no two boxes lie side-by-side inside of a larger one– in other words, no two boxes A and B lie inside box C if neither A contains B nor B contains A. (Otherwise, the two inner boxes might damage each other during shipping.) Assuming you need n different boxes and you know which boxes fit inside others, find a polynomial time algorithm to compute the minimum number of separate shipments needed, and compute the algorithm's running time.

    Solution. We note that if we have n boxes to ship and k boxes are nested inside others, we only have to ship $n-k$ different packages. Thus, an equivalent formulation of the problem is to maximize the number of boxes that we can fit inside others. With this, we construct a graph. Create two sets U and V, and for each box i, create the nodes u_i in U and v_i in V. Now, for every pair of boxes i and j such that box j fits inside box i, add an edge of capacity 1 from u_i to v_j. Create a source node s and a target node t, and connect s to every node in U and every node in V to t with capacity 1 edges. We run the Ford-Fulkerson algorithm on this graph. This algorithm will find us a maximal bipartite matching from U to V by taking the edges from U to V in our returned flow. We convert this to a scheme to nest boxes as follows: for every edge connecting u_i to v_j in the maximum bipartite matching, put box j inside box i. We see that box i is put inside another box if and only if some edge in our maximum matching points into it. Since a maximum matching clearly maximizes the number of such boxes, this algorithm will give us a nesting of boxes which maximizes the number of boxes which are put inside of others. By our initial observation, this algorithm thus minimizes the number of needed packages, as desired.

To analyze runtime, we first note that the runtime of the Ford-Fulkerson algorithm is $O(mF)$, where m is the number of edges and F is the size of the maximum flow. Since U and V both have size n, there are clearly at most n^2 edges between them. Since there are n edges connecting each of s and t, we have that $m = O(n^2)$. Now, we note that the set t corresponds to a $s - t$ cut of size n. By min-cut max-flow, we conclude that the maximum $s-t$ flow is bounded by n. Thus, the runtime of the Ford-Fulkerson algorithm for this problem is $O(mF) = O(n^3)$. Finally, since our step to convert the maximum matching to a nesting of boxes looks at at most every edge in the matching, this step clearly takes $O(n)$, and thus the entire algorithm runs in $O(n^3)$ time.