Colorado CSCI 5454: Algorithms
Homework 6

Instructor: Bo Waggoner
Due: October 10, 2019 at 11:59pm
Turn in electronically via Gradescope.

Remember to list people you worked with and any outside sources used. If none, write "none".

Problem 1 (6 points)

(Probability questions.)

Part a (1 point) Prove the following lemma from Max-3SAT in class: given a clause consisting of the OR of three distinct Boolean variables or their negations, setting the variables independently and uniformly at random gives a $\frac{7}{8}$ probability of satisfying the clause.

Solution. The only way for the clause to be false is if all three literals are false. Each literal is false independently with probability $\frac{1}{2}$ (note this is true regardless of if it is a negation of a variable or not), so the probability is $(\frac{1}{2})^3 = \frac{1}{8}$.

Part b (1 point) Suppose we have a coin that comes up heads with probability just p (think of these as trials with a very small success probability p). What is the probability of no heads, if we flip the coin k times independently?

Solution. Each trial is tails with probability $1 - p$, and they are independent, so the probability every trial is tails is $(1 - p)^k$.

Part c (1 point) What is the probability of getting at least one "success" (heads), if we flip the coin k times independently?

Solution. $1 - (1 - p)^k$, because this occurs if we do not have all-tails.

Part d (1 point) Show that, given any $\delta > 0$, if we flip the coin $\frac{1}{p} \ln\left(\frac{1}{\delta}\right)$ times independently, then we get at least one "success" (heads) with probability at least $1 - \delta$.

Hint: use the key fact that $1 + x \leq e^x$ for all $x \in \mathbb{R}$. Note this implies $1 - x \leq e^{-x}$ also!

Solution. We assume $\delta < 1$. Let $k = \frac{1}{p} \ln\left(\frac{1}{\delta}\right)$. The probability of all-tails is

$$\begin{aligned}(1-p)^k &\leq e^{-pk} \\ &= e^{-\ln(1/\delta)} \\ &= e^{\ln(\delta)} \\ &= \delta.\end{aligned}$$

So the probability of all-heads is $1 - (1-p)^k \geq 1 - \delta$.

Part e (2 points) In class, we saw that the contraction algorithm finds a min cut with probability at least $\frac{2}{n(n-1)}$. Prove that if we run the algorithm independently $\frac{n(n-1)}{2} \ln\left(\frac{1}{\delta}\right)$ times and take the minimum cut out of all its outputs, the probability of returning a min cut is at least $1 - \delta$.

Hint: use your answers to previous parts.

Solution. This reduces to the previous problem with $p = \frac{2}{n(n-1)}$. (There is a subtlety, which is: in the previous problem we had a probability p exactly of a success, whereas here we have a probability of at least p of a success. However, one can argue that this only increases the chance of some success, so $1 - \delta$ is still a lower bound. This was not needed for full credit.)

Problem 2 (6 points)

The *Max Cut* problem is as follows. We are given an unweighted, undirected graph $G = (V, E)$. We must output a *cut* U_1, U_2 of the graph (recall: how did we define a cut?).

The goal is to maximize the number of edges crossing the cut, i.e. edges with one vertex in U_1 and one in U_2. Achieving the exact optimal solution is known to be NP-hard.

(For example: in a bipartite graph $G = (U, V, E)$, the max cut would be U, V.)

Part a (4 points) Give a randomized algorithm for this problem and prove it has approximation ratio 0.5.

Hint 1 (algorithm): What's the simplest random cut you can construct?

Hint 2 (analysis): Let $X_{uv} = 1$ if edge (u, v) crosses the cut, 0 otherwise. Then the objective is $\mathbb{E}\left[\sum_{(u,v)\in E} X_{uv}\right]$.

Solution. Algorithm: for each vertex u, independently with probability 0.5 place it in U_1, otherwise place it in U_2.

Analysis: The algorithm's performance is $ALG = \sum_{(u,v)\in E} X_{uv}$, the number of edges crossing the cut. For each edge (u, v) there are four possibilities: each of u, v can be in U_1 or U_2. All four possibilities are equally likely, and in two of them, the edge crosses the cut (i.e. $u \in U_1, v \in U_2$ and $u \in U_2, v \in U_1$).

So $\Pr[X_{uv} = 1] = 0.5$ for all edges (u, v). This implies $\mathbb{E}[X_{uv}] = 0.5(1) + 0.5(0) = 0.5$. So

$$\begin{aligned}\mathbb{E}[ALG] &= \mathbb{E}[\sum_{(u,v)\in E} X_{uv}] \\ &= \sum_{(u,v)\in E} \mathbb{E}[X_{uv}] \\ &= \sum_{(u,v)\in E} 0.5 \\ &= 0.5|E|.\end{aligned}$$

Meanwhile (and this is crucial), the optimal maximum number of edges is at most $|E|$, i.e. all the edges. So $\mathbb{E}[ALG] \geq 0.5 \cdot OPT$.

Part b (2 points) In the weighted Max Cut problem, the input additionally has edge weights w_{uv}. The goal is to maximize the total weight of edges crossing the cut.

Briefly argue that your algorithm above also gives a 0.5 approximation ratio for weighted Max Cut.

Solution. (Note: The problem should have stated all edge weights are assumed positive.) For each edge, we now get w_{uv} with half probability, and 0 otherwise. So if we let $X_{uv} = w_{uv}$ if the edge crosses the cut, then $\mathbb{E}[X_{uv}] = 0.5w_{uv}$, and we get $\mathbb{E}[ALG] = 0.5 \sum_{(u,v)\in E} w_{uv}$. Meanwhile, OPT can get at most all the edges, so $\mathbb{E}[ALG] \geq 0.5 \cdot OPT$.

Part c (Bonus: 1 point) (Derandomization.) Give a deterministic 0.5-approximation-ratio algorithm for Max Cut.

Hint: Start by taking the first randomized decision of your algorithm and making it deterministically instead, while leaving the rest of the algorithm randomized. Argue that this weakly increases the expected size of the cut. Then repeat for each remaining decisions.

Problem 3 (10 points)

In the *weighted vertex cover problem*, the input is an undirected graph $G = (V, E)$ and a list of positive vertex weights: w_v for each $v \in V$.

The goal is to output a vertex cover while minimizing *total weight*.

Recall that a *vertex cover* is a set of vertices, $S \subseteq V$, such that for all edges $(u, v) \in E$, at least one of $\{u, v\}$ is in S. The *total weight* of a vertex cover is $w(S) := \sum_{v\in S} w_v$.

Part a (4 points) Recall from the previous homework the following algorithm: find a maximal matching M in the graph, then let S equal the set of all endpoints of edges in M.

Does this algorithm give a constant-factor approximation to the *weighted* vertex cover problem? If so, prove it (and give the factor). If not, prove why not (e.g. using counterexamples).

Solution. It does not guarantee any constant factor. To prove this, let C be any positive constant. We will give an example where the algorithm's performance is more than $C \cdot OPT$.

Consider a graph on two vertices u, v with one edge between them and $w_u = 1, w_v = C$. Here $ALG = 1 + C$ but $OPT = 1$, so $\frac{ALG}{OPT} = 1 + C \geq C$.

Part b (4 points) Now consider this randomized algorithm: for each edge in the graph, if it is not yet covered, pick one of its endpoints uniformly at random and add it to S.

Does this algorithm give a constant-factor approximation? If so, prove it (and give the factor). If not, prove why not (e.g. using counterexamples).

Solution. It does not give any constant factor. To prove this, let C be any positive constant. We will give an example where the randomized algorithm's **expected** performance is more than $C \cdot OPT$.

Consider a graph on two vertices u, v with one edge between them and $w_u = 1$, $w_v = 2C$. Here $\mathbb{E}[ALG] = 0.5(1) + 0.5(2C) = C + 0.5$, but $OPT = 1$. So $\frac{\mathbb{E}[ALG]}{OPT} = C + 0.5 \geq C$.

Part c (2 points) Briefly compare your answers above to the randomized algorithm described in class. (For algorithms: which is better and why? For counterexamples: why does the algorithm from class succeed?)

Solution. (This answer deals with the examples above; student answers may vary, but the key point is the algorithm in class is biased toward adding vertices with smaller weight, in such a way that the expected total weight is small.)

On the graphs above, the randomized algorithm in class will with most of the probability only add u (i.e. probability $\frac{C}{1+C}$ in the first example). So compared to part (a), it only takes one of the vertices instead of both, and it prefers to take the smaller one. Compared to part (b), it takes the larger vertex with much less probability.

Part d (Bonus: 1 point) Illustrate your answers above by writing a program that generates graphs of various sizes and runs the above three algorithms on them. For each kind of graph, create a plot where average performance (total weight of the vertex cover) of each algorithms is plotted as a function of n, the number of vertices.