COMS 4995 (Randomized Algorithms): Problem Set #1

Due by 11:59 PM on Wednesday, September 25, 2019

Instructions:

(1) Form a group of 1-3 students. You should turn in only one write-up for your entire group.

(2) Submission instructions: We are using Gradescope for the homework submissions. Go to www.gradescope.com to either login or create a new account. Use the course code 9D6V5E to register for this class. Only one group member needs to submit the assignment. When submitting, please remember to add all group member names in Gradescope. See the course Web site for detailed instructions.

(3) Please type your solutions if possible and we encourage you to use the LaTeX template provided on the course home page.

(4) Write convincingly but not excessively.

(5) Some of these problems are difficult, so your group may not solve them all to completion. In this case, you can write up what you've got (subject to (3), above): partial proofs, lemmas, high-level ideas, counterexamples, and so on.

(6) Except where otherwise noted, you may refer to your lecture notes and the specific supplementary readings listed on the course Web page *only*. You can also review any relevant materials from your undergraduate algorithms course. If you do use any approved sources, make you sure you cite them appropriately, and make sure that all your words are your own.

(7) You can discuss the problems verbally at a high level with other groups. And of course, you are encouraged to contact the course staff (via Piazza or office hours) for additional help.

(8) If you discuss solution approaches with anyone outside of your group, you must list their names on the front page of your write-up.

(9) Refer to the course Web page for the late day policy and the School of Engineering honor code.

Problem 1

(15 points) In the *weighted independent set (WIS)* problem, the input is an undirected graph $G = (V, E)$ and a nonnegative weight $w_v \geq 0$ for each vertex $v \in V$. The goal is to output an *independent set*—a subset $S \subseteq V$ of vertices that are mutually non-adjacent—with the maximum total weight $\sum_{v \in S} w_v$.[1] This problem is NP-hard, so we will consider a polynomial-time heuristic:

1. Order the vertices $v_1, v_2, \ldots, v_n$ uniformly at random (out of the $n!$ possible orderings, where $n = |V|$).
2. $S := \emptyset$
3. For $i = 1, 2, \ldots, n$:
   (a) If v_i is not adjacent to any vertex of S, $S := S \cup \{v_i\}$.

[1] For example, in a clique, only singletons are independent sets. On a path, taking every other vertex produces an independent set.

4. Return S.

Suppose the maximum degree of a vertex of G is Δ (where the degree of a vertex is its number of neighbors). Prove that the expected total weight of the independent set returned by the algorithm above is at least $\frac{1}{\Delta+1}$ times the total weight of an optimal (maximum-weight) independent set.

Problem 2

(15 points) In Lecture #2 we gave a $\frac{7}{8}$-approximation algorithm for the MAX 3SAT problem, where every clause is the OR of three literals corresponding to distinct variables (e.g., $\neg x_1 \vee x_2 \vee \neg x_3$). In this problem we'll consider the MAX SAT problem, where each clause is the OR of literals corresponding to any number of distinct variables (i.e., anywhere from 1 to all n variables).

(a) (4 points) Show that, for every SAT formula, a random assignment satisfies in expectation at least 50% of the clauses. Give an example of a SAT formula for which no truth assignment satisfies more than 50% of the clauses.

(b) (6 points) Consider a SAT formula that does not contain directly contradictory unit clauses—i.e., that does not contain both the clauses x_i and $\neg x_i$ for some variable x_i. Give a randomized algorithm that satisfies, in expectation, at least 60% of the clauses.

[Hint: if a variable appears in a unit clause, bias its assignment appropriately.]

(c) (5 points) Give a randomized algorithm for MAX SAT (with no assumptions about contradictory unit clauses) that satisfies, in expectation, at least $60\% \cdot OPT$ clauses, where OPT is the maximum number of clauses satisfied by any truth assignment. (This does not contradict the second part of (a), since OPT might be considerably smaller than the number of clauses.)

Problem 3

(18 points) This problem considers two extensions of Karger's contraction algorithm (Lecture #3), one to approximate minimum cuts and one to r-way minimum cuts.

(a) (9 points) Let $\alpha \geq 1$ be a positive integer. A cut (A, B) of a graph $G = (V, E)$ is *α-approximate* if the number of edges crossing it is at most α times that crossing a minimum cut. Prove that for every fixed positive integer α, every n-vertex graph G has $O(n^{2\alpha})$ different α-approximate cuts. (The leading constant suppressed by the big-O notation can have arbitrary dependence on α; that is, we think of α as a fixed constant while $n \to \infty$.)

[Hint: stop the contraction algorithm when $2\alpha + 1$ vertices remain and then return a uniformly random cut of the contracted graph. What is the probability that this algorithm outputs a target α-approximate cut (A, B)?]

(b) (9 points) For a positive integer $r \geq 2$, an *r-way cut* of an undirected graph $G = (V, E)$ is a partition of the vertex set V into r non-empty subsets $A_1, A_2, \ldots, A_r$. An edge *crosses* an r-way cut if its endpoints lie in two distinct groups of the partition. Extend Karger's algorithm (and our analysis of it) to give, for every fixed integer $r \geq 2$, a randomized polynomial-time algorithm for computing an r-way cut with the minimum-possible number of crossing edges. (The running time of your algorithm can be exponential in r, but should be polynomial in n for every fixed r.) What upper bound does your analysis imply on the maximum number of minimum r-way cuts in an n-vertex graph?

Problem 4

(15 points) Karger's randomized minimum cut algorithm is unlikely to fail (i.e., to contract an edge crossing the desired minimum cut) in its earliest iterations, but it is increasing likely to fail in its later iterations (as

the number of vertices decreases). This suggests doing more independent trials of later iterations than of earlier ones; this is the key idea to speeding up the algorithm.[2]

Fix an n-vertex graph $G = (V, E)$.

(a) (4 points) Consider running two independent trials of Karger's contraction algorithm (with a total of $2n - 4$ contraction operations between them) and returning the better of the two results. Give a lower bound on the probability that the algorithm returns a minimum cut.

(b) (6 points) For a parameter k, suppose instead we run $n - k$ contraction iterations to reduce the input graph G to a k-vertex graph G'. Then, for another parameter ℓ, we make ℓ copies of G' and perform an independent trial of Karger's contraction algorithm on each (returning the best solution found). Determine the total number of edge contractions performed by this algorithm and bound from below the probability that it outputs a minimum cut.

(c) (5 points) Find optimal (or at least near-optimal) values of k and ℓ that maximize your lower bound on the probability of finding a minimum cut while using at most $2n - 4$ edge contractions. (Your success probability should be significantly higher than in part (a).)

Problem 5

(20 points) One can classify randomized algorithms for decision problems according to their correctness properties:

1. *Las Vegas*, or alternatively *zero-sided error.* Such an algorithm is correct with probability 1; only its running time is random. (Example: QuickSort.)

2. *Monte Carlo with one-sided error on "yes" instances.* Such an algorithm is always correct on "no" instances (i.e., no false positives) but errs with some probability on "yes" instances (i.e., can have false negatives). (Example: Karger's contraction algorithm for the problem "is there a cut with at most k crossing edges"?)

3. *Monte Carlo with one-sided error on "no" instances.* Such an algorithm is always correct on "yes" instances (i.e., no false negatives) but errs with some probability on "no" instances (i.e., can have false positives). (Example: the Miller-Rabin algorithm, for the problem "is the given integer prime"?)

4. *Monte Carlo with two-sided error.* Such an algorithm can err with some probability on both "yes" and "no" instances. (We haven't seen any examples yet in lecture, but there are many natural ones, for example for sampling/estimation problems.)

These four types of algorithms give rise to four different complexity classes, each a formalization of "problems solvable by a randomized algorithm in polynomial time." Recall from your study of NP-completeness that a *decision problem* is specified by a language $L \subseteq \{0,1\}^*$. Inputs x that belong to L are called "yes instances," those not in L are called "no instances."

1. (Polynomial-time Las Vegas algorithms.) A decision problem belongs to the complexity class $\mathcal{ZPP}$ if there is an algorithm A with expected running time polynomial in the input size and with

$$\begin{aligned} \mathbf{Pr}[A \text{ is correct on } x] &= 1 \quad \text{if } x \in L \\ \mathbf{Pr}[A \text{ is correct on } x] &= 1 \quad \text{if } x \notin L. \end{aligned}$$

2. (Polynomial-time Monte Carlo algorithms with one-sided error on "yes" instances.) A decision problem belongs to the complexity class $\mathcal{RP}$ if there is an algorithm A with worst-case running time polynomial in the input size and with

$$\begin{aligned} \mathbf{Pr}[A \text{ is correct on } x] &\geq \tfrac{1}{2} \quad \text{if } x \in L \\ \mathbf{Pr}[A \text{ is correct on } x] &= 1 \quad \text{if } x \notin L. \end{aligned}$$

[2]We'll ignore implementation details and use edge contractions as our notion of a "primitive operation."

3. (Polynomial-time Monte Carlo algorithms with one-sided error on "no" instances.) A decision problem belongs to the complexity class co-$\mathcal{RP}$ if there is an algorithm A with worst-case running time polynomial in the input size and with

$$\begin{array}{ll}\mathbf{Pr}[A \text{ is correct on } x] = 1 & \text{if } x \in L \\ \mathbf{Pr}[A \text{ is correct on } x] \geq \frac{1}{2} & \text{if } x \notin L.\end{array}$$

4. (Polynomial-time Monte Carlo algorithms with two-sided error.) A decision problem belongs to the complexity class $\mathcal{BPP}$ if there is an algorithm A with worst-case running time polynomial in the input size and with

$$\begin{array}{ll}\mathbf{Pr}[A \text{ is correct on } x] \geq \frac{2}{3} & \text{if } x \in L \\ \mathbf{Pr}[A \text{ is correct on } x] \geq \frac{2}{3} & \text{if } x \notin L.\end{array}$$

(a) (4 points) Prove that $\mathcal{ZPP}$ is contained in each of the other three classes. (I.e., given a Las Vegas algorithm for a problem with expected polynomial running time, show how to extract from it algorithms of the other three types.)

[Hint: start by using Markov's inequality (see Exercise #3) to bound the probability that the running time of a $\mathcal{ZPP}$ algorithm is more than a constant factor larger than its expectation.]

(b) (4 points) Prove that $\mathcal{ZPP} = \mathcal{RP} \cap \text{co-}\mathcal{RP}$. (I.e., in light of (a), show how to extract a Las Vegas algorithm with expected polynomial running time from Monte Carlo algorithms with each type of one-sided error.)

(c) (4 points) Consider a randomized algorithm A for a decision problem L that runs in worst-case polynomial time and "knows when it's right," meaning that it outputs "yes" only when $x \in L$, outputs "no" only when $x \notin L$, and additionally might output "fail" (whether or not $x \in L$). Suppose that, for every (yes or no) input, A outputs "fail" with probability at most $\frac{1}{2}$. Prove that $L \in \mathcal{ZPP}$. (I.e., convert A into a Las Vegas algorithm for the problem.)

(d) (4 points) Suppose we modify the definition of $\mathcal{RP}$ by replacing the "$\frac{1}{2}$" by "$\frac{1}{|x|^{10}}$," where $|x|$ denotes the size of input x. Prove that the corresponding complexity class remains the same (i.e., a decision problem L belongs to $\mathcal{RP}$ under the new definition if and only if it does under the original definition).

(e) (4 points) Suppose we modify the definition of $\mathcal{BPP}$ by replacing the "$\frac{2}{3}$" by "$\frac{1}{2}$." Would the resulting complexity class be the same as $\mathcal{BPP}$? Explain your answer.

Problem 6

(17 points) Recall that, for the Max Cut problem (Lecture #2), a random cut gives a $\frac{1}{2}$-approximation in expectation. (Specifically, the expected weight of the edges crossing a random cut is 50% times the sum of the weights of all of the edges.) This problem outlines one way to derandomize that algorithm. The plan is to identify a polynomial-size set S of cuts (depending on n but not on the specific graph) such that, for every n-vertex graph, at least one of the cuts of S is a $\frac{1}{2}$-approximation. Given such a set S, we can obtain a deterministic $\frac{1}{2}$-approximation algorithm for Max Cut by enumerating all of the cuts of S and returning the best of them.

For a positive integer k, consider the following mapping from k-bit strings to n-bit strings, where $n = 2^k - 1$. Coordinates of the output correspond to the $2^k - 1$ non-empty subsets of $\{1, 2, \ldots, k\}$. A k-bit string $b_1 b_2 \ldots b_k$ is mapped to an n-bit string where the output bit corresponding to a non-empty set S is set to $\sum_{i \in S} b_i \bmod 2$. (I.e., to the "sum mod 2," or equivalently the "XOR" or "parity," of the input bits in the coordinates specified by S.) For example, when $k = 3$ (and $n = 7$):

Input	{1}	{2}	{3}	{1,2}	{1,3}	{2,3}	{1,2,3}
000	0	0	0	0	0	0	0
001	0	0	1	0	1	1	1
010	0	1	0	1	0	1	1
011	0	1	1	1	1	0	0
100	1	0	0	1	1	0	1
101	1	0	1	1	0	1	0
110	1	1	0	0	1	1	0
111	1	1	1	0	0	0	1

Observe that in each pair of columns, each of the 4 possibilities (00, 01, 10, and 11) occurs exactly twice. Thus, if we choose a row uniformly at random, for every $1 \le i < j \le 7$,

$$\mathbf{Pr}[(X_i, X_j) = (0,0)] = \mathbf{Pr}[(X_i, X_j) = (1,0)] = \mathbf{Pr}[(X_i, X_j) = (0,1)] = \mathbf{Pr}[(X_i, X_j) = (1,1)] = \frac{1}{4}, \qquad (1)$$

where X_i and X_j denote the values of the entries of the chosen row in the ith and jth coordinates. In other words, the random variables $(X_1, \ldots, X_7)$ are *pairwise* (a.k.a. *2-wise*) *independent.* I.e., from the perspective of any pair of random variables, it's as if the output string was chosen uniformly at random from the $2^7 = 128$ possibilities (as opposed to the 8 possibilities it was actually chosen from).

(a) (6 points) Prove that the construction above works in general. That is, for an arbitrary positive integer k and $n = 2^k - 1$, consider the map λ_k given by

$$b_1 b_2 \ldots b_k \mapsto x_{S_1} x_{S_2} \ldots x_{S_n}$$

with

$$x_S = \sum_{i \in S} b_i \bmod 2,$$

where the S_i's range over the $2^k - 1$ non-empty subsets of $\{1, 2, \ldots, k\}$. Prove that if a k-bit string $\mathbf{b}$ is chosen uniformly at random, then the output bits $(X_1, \ldots, X_n)$ of $\lambda_k(\mathbf{b})$ are pairwise independent, meaning that (1) holds for every $1 \le i < j \le n$.

(b) (6 points) Use the construction in (a) to give a deterministic $\frac{1}{2}$-approximation algorithm for the Max Cut problem.

[Hint: to prove a guarantee of $\frac{1}{2}$ for a uniformly random cut, how much randomness do you really need?]

(c) (5 points) Can you use the construction in (a) to derandomize the $\frac{7}{8}$-approximation algorithm for MAX 3SAT from Lecture #2? Explain your answer.