ShanghaiTech University

CS240 Algorithm Design and Analysis

Spring 2023

Problem Set 5

Due: 23:59, May 24, 2023

1. Submit your solutions to the course Blackboard.
2. If you want to submit a handwritten version, scan it clearly.
3. Late homeworks submitted within 24 hours of the due date will be marked down 25%. Homeworks submitted more than 24 hours after the due date will not be accepted unless there is a valid reason, such as a medical or family emergency.
4. You are required to follow ShanghaiTech's academic honesty policies. You are allowed to discuss problems with other students, but you must write up your solutions by yourselves. You are not allowed to copy materials from other students or from online or published resources. Violating academic honesty can result in serious penalties.

Problem 1:

Suppose you want to estimate the fraction f of people who skateboard in ShanghaiTech. Assume that you can select a ShanghaiTech student uniformly at random and determine whether they skateboard. Also, assume you know a lower bound $0 < a < f$. Design a procedure for estimating f by some $\hat{f}$ such that $\Pr[|f - \hat{f}| > \varepsilon f] < \delta$, for any choice of constants $0 < a, \varepsilon, \delta < 1$. What is the smallest number of residents you must query?

Solution:

To estimate the fraction f of people who skateboard in ShanghaiTech, we can use a technique called sequential probability ratio test (SPRT). The SPRT allows us to make statistically reliable estimates using a minimal number of samples.

Here's the procedure to estimate f with the desired properties:

1. Initialize the parameters:

- Set $0 < a < f$, which is the known lower bound on f.
- Set $0 < \varepsilon, \delta < 1$, which are the desired tolerances.
- Set $n = 0$, the initial number of samples.
- Set $X = 0$, the initial count of skateboarders.
- Set $Y = 0$, the initial count of non-skateboarders.
- Set $S = 0$, the initial log-likelihood ratio.

2. Repeat the following steps until the stopping condition is met:

- Select a random resident from ShanghaiTech uniformly.
- Determine whether the selected resident skateboards.
- If the resident skateboards, increment X by 1. Otherwise, increment Y by 1.
- Increment n by 1.
- Update the log-likelihood ratio S using the observed outcome:
- If $X/Y > f/(1-f)$, set S to $S + \log\left(\frac{f}{1-f}\right)$.
- If $X/Y < f/(1-f)$, set S to $S + \log\left(\frac{1-f}{f}\right)$.
- If $X/Y = f/(1-f)$, set S to $S + \log\left(\frac{a(1-f)}{f(1-a)}\right)$.

3. Compute the estimated fraction $\hat{f}$ as $\hat{f} = \frac{X}{n}$.

4. Compute the test boundaries:

- Set $L = \frac{\log(1/\delta) + \varepsilon \log\left(\frac{1-a}{a}\right)}{\log((1-a)/a)}$.
- Set $U = \frac{\log(\delta) + \varepsilon \log\left(\frac{1-a}{a}\right)}{\log((1-a)/a)}$.

5. Check the stopping condition:

- If $S \leq L$, stop and output $\hat{f}$ as the estimate.

- If $S \geq U$, stop and output $\hat{f}$ as the estimate.

- Otherwise, go back to step 2.

The stopping condition ensures that the estimated fraction $\hat{f}$ satisfies $\Pr[|f - \hat{f}| > \varepsilon f] < \delta$.

The number of residents you must query depends on the desired tolerances ε and δ. The stopping condition guarantees that the SPRT terminates when either the lower or upper test boundary is crossed. The smallest number of residents you must query can be calculated using the test boundaries. The minimum number of residents, denoted as N, is given by:

$N = \lceil L^U \rceil$

This formula ensures that the estimated fraction $\hat{f}$ satisfies the desired tolerance condition.

Problem 2:

Suppose that there are n items which need to be placed in some bins. The capacity of each bin is 1. The volumes of the items may not be the same and all volumes are smaller than 1. We want to use the fewest number of bins to place all items. Design a 2-approximation algorithm to solve this problem.

Solution:

To design a 2-approximation algorithm for the problem of placing n items in bins with a capacity of 1 each, you can use the following approach:

1. Sort the items in non-increasing order based on their volumes.

2. Create an empty list of bins.

3. For each item in the sorted list: - Check if the item can fit into any existing bin without exceeding the capacity of 1. If it can, place the item in that bin. - If the item cannot fit into any existing bin, create a new bin and place the item in that bin.

4. Return the total number of bins used.

This algorithm works by greedily placing items into bins in a sorted order. The sorting step ensures that larger items are placed first, which reduces the chances of needing additional bins later.

Now, let's analyze the approximation ratio of this algorithm. Let OPT be the optimal number of bins required to place all the items optimally.

At any point during the algorithm's execution, the number of bins used is always greater than or equal to the number of bins required by the optimal solution. This is because, in the worst case, each item may require a separate bin in the algorithm, whereas the optimal solution may find a way to pack multiple items into a single bin.

Therefore, the number of bins used by the algorithm is at most 2 * OPT. This establishes the 2-approximation ratio of the algorithm.

Note that this algorithm does not always achieve the optimal solution, but it guarantees that the number of bins used is at most twice the optimal solution.

Problem 3:

Consider the following simple model of gambling in the presence of bad odds. At the beginning, your net profit is 0. You play for a sequence of n rounds. In each round, your net profit increases by 1 with probability 1/3, and decreases by 1 with probability 2/3. Show that the expected number of steps for which your net profit is positive can be upper-bounded by an absolute constant, independent of the value of n.

Solution:

To analyze the expected number of steps for which your net profit is positive, let's define a random variable X to represent the number of steps until your net profit becomes zero. We want to show that the expected value of X can be upper-bounded by an absolute constant, regardless of the value of n.

In each round, your net profit increases by 1 with probability 1/3 and decreases by 1 with probability 2/3. This means that the probability of taking a step backward (from a positive net profit to zero) is higher than the probability of taking a step forward (from a zero net profit to positive).

Let's consider the probability of taking a step forward, i.e., transitioning from a zero net profit to a positive net profit. In order for this transition to occur, the previous step must have resulted in a decrease (a step backward) and the current step must result in an increase (a step forward). The probability of the previous step being a decrease is 2/3, and the probability of the current step being an increase is 1/3. Therefore, the probability of taking a step forward is (2/3) * (1/3) = 2/9.

Now, let's consider the probability of taking a step backward, i.e., transitioning from a positive net profit to zero. In order for this transition to occur, the current step must result in a decrease. The probability of the current step being a decrease is 2/3.

Since the probability of taking a step forward (2/9) is smaller than the probability of taking a step backward (2/3), we expect that, on average, it will take fewer steps to transition from a positive net profit to zero compared to transitioning from zero to a positive net profit.

Let's denote the expected value of X as E(X). We can express E(X) using the law of total expectation:

E(X) = (2/9) * 1 + (2/3) * (1 + E(X)),

where (2/9) * 1 represents the probability of taking a step forward and 1 is the number of steps taken in that case, and (2/3) * (1 + E(X)) represents the probability of taking a step backward and then an additional E(X) steps.

Simplifying the equation, we get:

$E(X) = (2/9) * 1 + (2/3) + (2/3) * E(X)$.

Rearranging the terms, we have:

$(1 - (2/3)) * E(X) = (2/9) * 1 + (2/3)$.

Simplifying further, we get:

$(1/3) * E(X) = (6/9) + (6/9)$.

$(1/3) * E(X) = (12/9)$.

$E(X) = 4$.

Therefore, the expected number of steps until your net profit becomes zero (i.e., the expected number of steps for which your net profit is positive) can be upper-bounded by an absolute constant of 4, independent of the value of n.

Problem 4:

Given a set of items, where the i'th item has size c_i ($c_i > 0$), we want to place them in as few bins as possible. Each bin has a capacity V, where $V \geq \max_i c_i$. To try to minimize the number of bins used, we take the following steps. We start with one active bin, then iterate through the items one by one and try to place the item into any active bin which can accommodate it. If an item cannot fit into any active bin, we open a new active bin. The algorithm is shown below. Prove that this algorithm is a 2-approximation, i.e. the number of bins it uses is at most twice the minimum possible number of bins required.

Algorithm:

Input : Set of items with sizes $c_1, c_2, \ldots, c_n$
Output: Number of bins used
Sort the items in non-decreasing order of their sizes: $c_1 \leq c_2 \leq \ldots \leq c_n$
for $i \leftarrow 1$ **to** n **do**
    for $j \leftarrow 1$ **to** k **do**
        if *item i fits into active bin j* **then**
            place item i in bin j
            break
        end
    end
    else
        open a new active bin $k+1$
        $k \leftarrow k+1$
        place item i in bin $k+1$;
    end
end
return *number of active bins k*

Solution:

To prove that the given algorithm is a 2-approximation, we need to compare the number of bins used by the algorithm with the minimum possible number of bins required.

Let's denote the number of bins used by the algorithm as k, and let OPT represent the minimum possible number of bins required.

To establish the 2-approximation guarantee, we'll consider two cases:

Case 1: $k \leq 2 \cdot OPT$ In this case, the number of bins used by the algorithm

is already within the desired approximation factor. Therefore, we can conclude that the algorithm is a 2-approximation.

Case 2: $k > 2 \cdot OPT$ In this case, we have more than twice the number of bins used by the algorithm compared to the optimal solution. We'll show that this situation is not possible, leading to a contradiction.

Since the algorithm uses k bins, each with a capacity of V, the total capacity of all bins used by the algorithm is $k \cdot V$. Moreover, as stated in the problem, $V \geq \max_i c_i$, meaning that the largest item can fit in a single bin.

Since $k > 2 \cdot OPT$, there must be at least $k/2$ bins used by the algorithm that are not used in the optimal solution. Let's consider these $k/2$ bins. Since each bin has a capacity of V, the total capacity of these bins is $(k/2) \cdot V$.

Now, let's consider the $k/2$ largest items that were placed in these additional bins. Each of these items has a size of at least $c_{n-k/2+1}$. Since the algorithm sorts the items in non-decreasing order, we know that $c_{n-k/2+1}$ is at least as large as the $k/2$ smallest items used in the optimal solution.

Therefore, the total size of the $k/2$ largest items in the algorithm's additional bins is at least the total size of the $k/2$ smallest items in the optimal solution.

However, this contradicts the fact that the total capacity of the algorithm's additional bins is $(k/2) \cdot V$, which should be greater than or equal to the total size of the $k/2$ smallest items in the optimal solution.

Thus, we have reached a contradiction, which means that k cannot be greater than $2 \cdot OPT$.

In both cases, we have either $k \leq 2 \cdot OPT$ or $k > 2 \cdot OPT$ resulting in a contradiction. Therefore, the algorithm is a 2-approximation.

Note: The proof assumes that the algorithm correctly determines whether an item can fit into an active bin or not. Additionally, it assumes that the algorithm places items into bins in a valid manner, obeying the bin capacities. These assumptions are crucial for the correctness of the proof.

Problem 5:

Suppose that for some decision problem, we have an algorithm which on any instance computes the correct answer with probability at least 4/5. We wish to reduce the probability of error by running the algorithm n times on the same input using independent randomness between trials and taking the most common result. Using Chernoff bounds, give an upper bound on the probability that this new algorithm produces an incorrect result.

Solution:

To apply Chernoff bounds in this scenario, let's assume that our algorithm produces an incorrect result with probability p. Since the algorithm computes the correct answer with probability at least 4/5 on any instance, we have $p \leq 1/5$.

Let X be a random variable that represents the number of incorrect results obtained when running the algorithm n times. X follows a binomial distribution with parameters n and p.

The probability of getting k incorrect results out of n trials is given by the binomial probability formula:

$$P(X = k) = \binom{n}{k} p^k (1-p)^{n-k}$$

We want to find an upper bound on the probability that the new algorithm produces an incorrect result. This can be expressed as the probability that the number of incorrect results, X, exceeds $n/2$, since we are taking the most common result.

$$P(X > n/2) = P(X \geq \lceil n/2 \rceil) = P(X \geq \lceil 0.5n \rceil)$$

To apply Chernoff bounds, we need to find the expectation of X, denoted as μ. For a binomial distribution, the expectation is given by:

$$\mu = np$$

Now, we can use Chernoff bounds to obtain an upper bound on $P(X > n/2)$. Chernoff bounds provide exponential tail bounds for sums of independent random variables. In this case, we have n independent trials.

Using Chernoff bound with the parameter $\delta > 0$, we have:

$$P(X \geq (1+\delta)\mu) \leq e^{-\frac{\delta^2 \mu}{3}}$$

In our case, we want to find an upper bound on $P(X > n/2)$, so $(1+\delta)\mu$ corresponds to $n/2$. Let's solve for δ:

$$(1+\delta)\mu = \frac{n}{2} \implies (1+\delta) = \frac{n}{2\mu} = \frac{2n}{np} = \frac{2}{p}$$

Substituting $p = 1/5$, we get:

$$(1+\delta) = \frac{2}{1/5} = 10$$

Now we can use the Chernoff bound formula to obtain an upper bound on $P(X > n/2)$:

$$P(X > n/2) \leq P(X \geq (1+\delta)\mu) \leq e^{-\frac{\delta^2\mu}{3}} = e^{-\frac{10^2 \cdot np}{3}} = e^{-\frac{100n}{15}} = e^{-\frac{20n}{3}}$$

Therefore, the upper bound on the probability that the new algorithm produces an incorrect result is $e^{-\frac{20n}{3}}$.