CMSC 641 Design & Analysis of Algorithms Spring 2023
Section 01 Homework 13

Faisal Rasheed Khan

VB02734

vb02734@umbc.edu

1 Hoeffding's Inequality

Hoeffding's Inequality is a generalization of Chernoff Bounds and applies to independent bounded random variables, regardless of their distribution.
Hoeffding's Inequality: Let X1, X2, . . . , Xn be independent random variables such that a ≤ Xi ≤ b for all i. Let X = $\sum_{i=1}^{n} X_i$ and μ = E[X]. Then, for all δ > 0:
Prob [X ≥ (1 + δ)μ] ≤ $e^{-\frac{2\delta^2 \mu^2}{n(b-a)^2}}$

Suppose we roll a 6-sided die where the faces are labeled 1, 2, 3, 4, 5 and 6. The expected value of the single die roll is 3.5. The probability that a single die roll is ≥ 4 is 0.5. If we roll the die 100 times, the expected average value is still 3.5. However, the probability that the average value is ≥ 4 should drop exponentially. Use Hoeffding's Inequality to bound the probability that the average value from rolling a 6-sided die 100 times is ≥ 4.

Questions:

1. What are the values of a and b?

A. The values of a,b are minimum and maximum values of random variable X, i.e. a=1 and b=6

2. How should you define the random variables Xi?

A. The random variables X_i are defined over the values of rolls of a die i.e which can have the values {1,2,3,4,5,6}

3. What are the values of μ and δ?

A. μ = E[X]
for single die roll 3.5, for 100 rolls still its 3.5
μ = 3.5
δ, deviation from the expected value = 4 – 3.5
δ = 0.5

4. What bound do you get from Hoeffding's Inequality for the probability that the average of 100 die rolls is ≥ 4?

A. By Hoeffding's Inequality,
Prob [X ≥ (1 + δ)μ] ≤ $e^{-\frac{2\delta^2 \mu^2}{n(b-a)^2}}$

δ = 0.5, μ = 3.5, n=100, a=1, b=6
Prob [X ≥ (1 + 0.5)3.5] ≤ $e^{-\frac{2*0.5^2 3.5^2}{100(6-1)^2}}$

Prob [X ≥ 4] ≤ $e^{-\frac{2*0.5^2*3.5\^2}{100(6-1)\^2}}$
Prob [X ≥ 4] ≤ $e^{-2.45 * 10\^-3}$
Prob [X ≥ 4] ≤ 0.9975

2 Approximate graph coloring?

[Adapted from Kleinberg & Tardos]

Graph coloring is a decision problem. An undirected graph is either 3-colorable or not 3-colorable. Often there is more than one way to convert a decision problem into an optimization problem. For graph coloring, the usual way to express graph coloring as an optimization problem is to ask for a coloring of the given graph that uses as few colors as you can.

In this problem, we take an alternative approach — we keep the number of colors constant, say three. That is, we are always considering an assignment of three colors to the vertices of the graph. We say that an edge is satisfied if its endpoints are assigned different colors. If every edge is satisfied, then we have a 3-coloring of the graph. The optimization problem is to assign one of three colors to each vertex of the graph that maximizes the number of edges that are satisfied. Let's call this problem MaxColor.

Side Note: MaxColor is definitely the "cheating" version of graph coloring as an optimization problem. There are no good approximation algorithms for the normal version of graph coloring. For example, there are rather sophisticated algorithms that can color a graph using about $O(n^{1/5})$ colors under the assumption that the graph is 3-colorable. (That is, even knowing the graph is 3-colorable does not help you color the graph with few colors.)

Assignment:

1. Devise and describe a randomized approximation algorithm for MaxColor that achieves an expected approximation ratio of 1.5.

A. <u>Randomized Approximation Algorithm for MaxColor:</u>

- Initially start with any random assignment of vertex and assign any one color to it
- For the selected vertex assigned color make sure it has satisfied edges, i.e. the vertex which is colored and its edges to different vertices, both vertices which are connected with edge should have different color assignments.
- Update the color of vertex based on the satisfied edges color criteria.
- Repeat the steps till every vertex is colored

Claim: Vertices which are connected via same edge has different colors

Proof:
Suppose the algorithm outputs satisfying color assignment. We need to show that Vertices which are connected via same edge has different colors. Suppose we assign one color to one of the vertices, then according to our algorithm it doesn't assign same color to other vertex which is sharing the common edge. Therefore, Vertices which are connected via same edge has different colors.

The randomized nature of the algorithm allows it to explore different color assignments and iteratively improve the number of satisfied edges. On average, the algorithm is expected to achieve a solution that satisfies at least 1.5 times the number of edges as an optimal 3-coloring would.
c* is the optimum solution

$$\frac{|c|}{|c^*|} \leq \frac{\frac{3}{2}\, e}{e}$$

$$\frac{|c|}{|c^*|} \leq \frac{3}{2}$$

2. Provide an analysis of the expected number of edges that are satisfied by your algorithm.

A. The analysis of the expected number of satisfied edges by the algorithm, it involves considering the probabilities of each edge being satisfied during the random assignment process by assigning colors to vertices in different ways and iterations of updating the color assignments. Specifically, we can analyze the expected number of satisfied edges by considering the probability of each edge being satisfied at each step of the algorithm i.e. Vertices which are connected via same edge has different colors. By calculating the probabilities of satisfaction for all edges and summing them up, we can estimate the expected number of edges that will be satisfied by the algorithm.
Consider a random variable X for the satisfied edges.
For each edge in the graph, determine the probability of it being satisfied under the random assignment. This probability depends on the current color assignments of the vertices connected by the edge. Calculate the Expected value X for these probability for the satisfied edges.
The algorithm guarantees an expected approximation ratio of 1.5, meaning c ≥ (3/2) * c* where c* is the optimum solution

3. State and briefly justify the running time of your algorithm

A. The running time of the algorithm is $O(2^V * E)$, where V is the number of vertices and E is the number of edges in the graph.
Assigning colors to each vertex takes O(V) time, as we need to visit each vertex once. In order to assign colors to V vertices we iterate over the edges and also check satisfied edges by iterating over the edges, it takes O(E). But if the satisfying edges criteria is not met then we traceback and try with the remaining 2 colors and satisfy the edges.
Therefore, running time = $O(2^V * E)$

References:

https://archive.org/details/introduction-to-algorithms-by-thomas-h.-cormen-charles-e.-leiserson-ronald.pdf/page/427/mode/2up

https://archive.org/details/AlgorithmDesign1stEditionByJonKleinbergAndEvaTardos2005PDF/page/n201/mode/2up

21.1-Approximation-Algorithms-post.pdf - Google Drive

21.2-Approximating-TSP-post.pdf - Google Drive

22-Bin-Packing-post.pdf - Google Drive

23.2-Subset-Sum-post.pdf - Google Drive

23.1-Approximating-Set-Cover-v2-post.pdf - Google Drive

25-Randomized-MinCut-v1.1-post.pdf - Google Drive

24-Randomized-Algorithms-post.pdf - Google Drive