Advanced Algorithms: Homework 4

Ivani Patel

November 12, 2022

1. I have k, for some k, water tanks, $T_1, ..., T_k$ (which are identical in size and shap), whose water levels are respectively denoted by nonnegative real variables $x_1, ..., x_k$. Without loss of generality, we assume that xi equals the amount of water that is currently in T_i. Initially, all the tanks are empty; i.e. $x_i = 0; 1 \leq i \leq k$. I have m pumps $p_1, ..., p_m$, that pump water into tanks. More precisely, a pump instruction, say, P_{A,c_1,c_2}, where $A \subseteq T_1, ..., T_k$, is to pump the same amount of water to each of the tank T_i with $i \in A$ (so water levels on other tanks not in A will not change), where the amount is anywhere between c_1 and c_2 (including c_1 and c_2, of course we have assumed $0 \leq c_1 \leq c_2$). For instance,$P_{\{T_2,T_5\},1.5,2.4}$, means to pump simontaneously to T_2 and T_5 the same amount of water. However, the amount can be anywhere between 1.5 and 2.4. Suppose that we execute the instruction twice, say:

$$P_{\{T2,T5\},1.5,2.4};$$
$$P_{\{T2,T5\},1.5,2.4}.$$

The first $P_{\{T_2,T_5\},1.5,2.4}$, can result in 1.8 amount of water pumped into T_2 and T_5 , respectively, and the second $P_{\{T_2,T_5\},1.5,2.4}$, can result in 2.15 amount of water pumped into T_2 and T_5, respectively. That is, the amount of water can be arbitrary chosen inside the range specified in the instruction, while the choice is independent between instructions.

Now, let M be a finite state controller which is specified by a directed graph where each edge is labeled with a pump instruction. Different edges may be label with the same pump instruction and may also be labeled with different pump instructions. There is an initial node and a final node in M. Consider the following condition Bad$(x_1, ..., x_k)$:

$$x_1 = x_2 + 1 = x_3 + 2 \wedge x_3 > x_4 + 0.26.$$

A walk in M is a path from the initial to the final. I collect the sequence of pump instructions on the walk. If I carefully assign an amount (of water pumped) for each such pump instruction and, as a result, the water levels $x_1, ..., x_k$ at the end of the sequence of pump instruction satisfy Bad$(x_1, ..., x_k)$, then I call the walk is a bad walk. Such a walk intuitively says that there is an undesired execution of M.

Design an algorithm that decides whether M has a bad walk. (Hint: first draw an example M where there is no loop and see what you can get. Then, draw an M that is with a loop and see what you get. Then, draw an M that is with two nested loops and see what you get, and so on.)

   (a) **Step 1**: In depth first search, find all the simple paths in the directed graph specified by the finite state controller M by marking all the nodes that we visit to ensure that we do not traverse cycles.

   Step 2:If there is a simple path from the first to the last node which satisfies the bad condition, then there is a bad walk in the graph.

   Step 3:Find all the SSC's using Tarjan's algorithm and find the simple cycles from the obtained SSC's.

Step 4: Find the output = min(1 + 2 + ... + n) based on all the given constraints using Linear Programming using Simplex method.

Step 5: If the output is zero, then there is a bad walk in the graph. If the output is not zero then there is no bad walk in the graph.

2. The word bit comes from Shannon's work in measuring the randomness in a fair coin. However, such randomness measurement requires a probability distribution of the random variable in consideration. Suppose that a kid tosses a dice for 1000 times and hence he obtains a sequence of 1000 outcomes

$$a_1, a_2, , a_{1000}$$

where each a_i is one of the six possible outcomes. Notice that a dice may not be fair at all; i.e., the probability of each outcome is not necessarily $\frac{1}{6}$. Based on the sequence only, can you design an algorithm to decide how "unfair" the dice that the kid tosses is.

(a) **Step 1**: We will build a table which will have the occurrence of each value of dice. For each throw of dice we enter the value which has occurred. And perform this for 1000 times.

If in the first throw 4 occurred the table entry will be as given.

1	2	3	4	5	6
0	0	0	1	0	0

Step 2: Now we will count each column of the table and store the value in $d_1, d_2, ..., d_6$.

Let D be the total number of rolls. i.e $D = d_1 + d_2 + ...d_6$.

Step 3: Find the expected number of times each side should come up i.e. The total number of rolls divided by number of sides.

$$d_{exp} = \frac{D}{6}$$

Step 4: Using Chi-Square test we get

$$\chi_k^2 = \frac{(d_k - d_{exp})^2}{d_{exp}}$$

Where k = 1 to 6. Get sum of all the values.

$$\chi = \chi_1^2 + \chi_2^2 + ... + \chi_6^2$$

DF	0.995	0.99	0.975	0.95	0.9	0.1	0.05	0.025	0.01	0.005
1	---	---	0.001	0.004	0.016	2.706	3.841	5.024	6.635	7.879
2	0.010	0.020	0.051	0.103	0.211	4.605	5.991	7.378	9.210	10.597
3	0.072	0.115	0.216	0.352	0.584	6.251	7.815	9.348	11.345	12.838
4	0.207	0.297	0.484	0.711	1.064	7.779	9.488	11.143	13.277	14.860
5	0.412	0.554	0.831	1.145	1.610	9.236	11.070	12.833	15.086	16.750
6	0.676	0.872	1.237	1.635	2.204	10.645	12.592	14.449	16.812	18.548

Figure 1: Chi-Square distribution table

Step 5: From Chi-Square distribution table we will find value of

$$df = k - 1$$

$$= 6 - 1 = 5$$

Step 6:Generally we pick P value as 0.05 which is 95% confidence level which in this case is 11.070.

So if the value of $\chi^2 \leq 11.070$, then we consider the dice is fair. If $\chi^2 > 11.070$, the dice is unfair. To measure how unfair the dice is we calculate $|\chi - 11.070|$

3. In below, a sequence is a sequence of event symbols where each symbol is drawn from a known finite alphabet. For a sequence $\alpha = a_1, ..., a_k$. that is drawn from a known finite set S of sequences, one may think it as a sequence of random variables $x_1, ..., x_k$. taking values $x_i = a_i$, for each i. We assume that the lengths of the sequences in the set S are the same, say n. In mathematics, the sequence of random variables is called a stochastic process and the process may not be i.i.d at all (independent and identical distribution). Design an algorithm that takes input S and outputs the likehood on the process being i.i.d.

   (a) **Step 1**: Encode each symbol from alphabet as unique non binary number. Lets take there are m sequences in set S. Join sequences in pair so now we have $\frac{m^2}{2}$ pairs. Take a variable P = 0 to store the number of i.i.d. pairs.

   Step 2: We have a new set S'. Now all the sequences in S' are presented as number sequences.Calculating average values for each sequences we get:

$$\bar{x}_1 = \frac{\sum_{i=1}^{n} x_{1,n}}{n}$$

$$.$$

$$.$$

$$.$$

$$\bar{x}_m = \frac{\sum_{i=1}^{n} x_{m,n}}{n}$$

   Pick one pair in S'. Hypothesize those two sequences which are identically distributed. Apply permutation test. Choose the value p = 0.05.

   If the hypothesize is not true, repeat Step 2, choose other sequences pair. If the hypothesize in Step 2 is true, hypothesize are independent.

   Step 3: Now we apply Chi-Square test. If p value is less than 0.05 hypothesize is not true, repeat Step 2, choose other sequences pairs. If p value is larger than 0.05 hypothesize is true. P = P + 1.

   Step 4: After going through all pairs i S', the likehood on the process being i.i.d. can be computed by

$$likehood = \frac{2P}{m^2}$$

4. Let G_1 and G_2 be two directed graphs and v_1, u_1 be two nodes in G_1 and v_2, u_2 be two nodes in G_2. Suppose that from v_1 to u_1, there are infinitely many paths in G_1 and that from v_2 to u_2, there are infinitely many paths in G_2 as well. Design an algorithm deciding that the number of paths from v_1 to u_1 in G_1 is "more than" the number of paths from v_2 to u_2 in G_2, even though both numbers are infinite (but countable).

(a) **Step 1**: Lets take G_1 and G_2 are SSC's respectively. And we take adjacent matrix of G_1 as M_1 and the adjacent matrix of G_2 as M_2.

Step 2: Take the largest eigenvalues of M_1 and M_2 as λ_1 and λ_2, which are also known as Perron numbers.

Step 3: M_1^n represents the total number of walks with length n in M_1, which can be approximated by

$$M_1^n = \lambda_1^n * v_{\lambda_1} * u_{\lambda_1}^T$$

where λ_1 is the Perron number of M_1, v_{λ_1} is the left eigenvector of λ_1, $u_{\lambda_1}^T$ is the right eigenvector of λ_1.

Step 4: In G_1 the total number of walks v_1 and u_1. taking them as walks from node i to node j with length of n can be approximated by

$$M_1^n[i,j] = \frac{v_i * u_j}{||u||} * \lambda_1^n * v_{\lambda_1} * u_{\lambda_1}^T$$

where $||u|| = \sum_k u_k$ and v_i, u_j are the components in vectors v, u.

Step 5: Sum up all $M_1^n[i,j]$ for every length n, i to node j with length less equal to n is S_1,

$$S_1 = \sum_{i=1}^{n} M_1^n[i,j]$$

Step 6: In the same way we can get S_2 for the total number of walks from node a to node b which is given as v_2, u_2 with the length less equal to n in the G_2.

Step 7: If $S_1 - S_2 > 0$, we can decide the number of paths from i to j in G_1 is more than the number of paths from node a to node b in G_2 with less than equal to n.