Cpt S 515 Homework #4

Instructor: Zhe Dang

Student: Sheng Guan

11593929

1. I have k, for some k, water tanks, $T1,\cdots,Tk$ (which are identical in size and shap), whose water levels are respectively denoted by nonnegative real variables $x1,\cdots,xk$. Without loss of generality, we assume that xi equals the amount of water that is currently in Ti. Initially, all the tanks are empty; $i.e.,xi = 0;\ 1 \leq i \leq k.$ I have m pumps $p1,\cdots,pm$, that pump water into tanks. More precisely, a pump instruction, say, PA, c1, c2 , where $A \subseteq \{T1,\cdots,Tk\}$, is to pump the same amount of water to each of the tank Ti with $i \in A$ (so water levels on other tanks not in A will not change), where the amount is anywhere between $c1$ and $c2$ (including $c1$ and $c2$, of course we have assumed $0 \leq c1 \leq c2$). For instance, $P\{T2,T5\},1.5,2.4$ means to pump simontaneously to T2 and T5 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 $P\{T2,T5\},1.5,2.4$ can result in 1.8 amount of water pumped into T2 and T5, respectively, and the second $P\{T2,T5\},1.5,2.4$ can result in 2.15 amount of water pumped into T2 and T5, 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 labeled 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 $Bad(x1,\cdots,xk)$:
$x1 = x2 + 1 = x3 + 2 \wedge x3 > x4 + 0.26.$
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 wa- ter pumped) for each such pump instruction and, as a result, the water levels $x1,\cdots,xk$ at the end of the sequence of pump

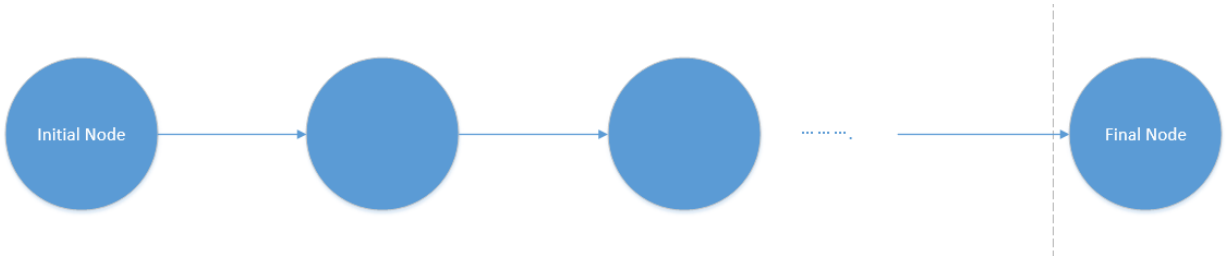
instruction satisfy $Bad(x1,\cdots,xk)$, then I call the walk is a bad walk. Such a walk intuitively says that there is an undesired execution of M.

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.)

Answer:

(1) We start with the simplest case first that the walk M does not contain any loop:

Assume we have a walk from the initial node to the final node and it looks like the below figure:



The algorithm-1 idea is: first, we use DFS to search such walks from the initial node to the final node and record all walks;

To simplify our analysis, we use the example Bad Walk definition proposed by the question to illustrate our ideas. Now we consider a walk satisfies:

x1 =x2 +1=x3 +2∧x3 >x4 +0.26

as a bad walk:

Worth to mention that this approach also applies when the bad walk definition is changed and our approach is a generalized one to solve the problem.

Then for each walk, we check all the edges related with T1, T2, T3, T4, and check whether the below linear programming constraints satisfy or not:

$$\sum_{i=0}^{n} x1 \; = 1 + \sum_{i=0}^{n} x2$$

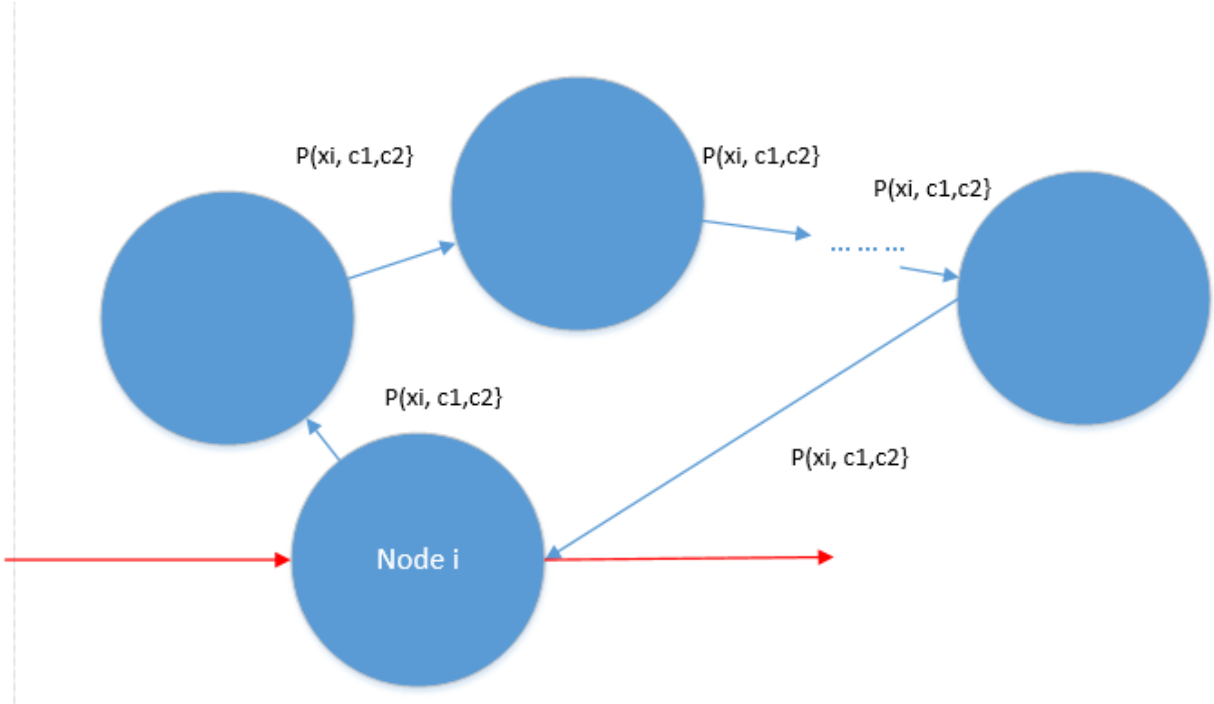
$$\sum_{i=0}^{n} x1 \; = \sum_{i=0}^{n} x3 \; + 2$$

$$\sum_{i=0}^{n} x3 \; > 0.26 + \sum_{i=0}^{n} x4$$

a) If satisfies, the corresponding walk is a bad walk, the algorithm will decide yes and print out this walk

b) Else, the algorithm will say there is no bad walk in M and decide no.

(2) The directed graph with several nodes that are included in one loop. To illustrate the idea, we need to define one loop first.



At this time, we can consider all other nodes and edges in this loop as One big self-circle edge on the node, and all P instructions can be summed into one instruction P'{xi, c'1,c'2}.

The algorithm Algorithm-2 now is based on Algorithm-1 but is more robust:

1) We use DFS to traverse the graph, and from the initial node to the Final node we will get one MAIN PATH(as red edges shown in the above figure) as a simple path M. At the same time we record loop along with this M if any. Continue the same procedure, in the end, we will get a set of M {M1, M2, M3,…,Mk} with corresponding loops if exist.
2) Use Algorithm-1 to search bad walks along with the MAIN PATH of Mi, (i=1,2,…,k). If Algorithm-1 decides yes, that means a bad walk existing without considering loop, Algorithm-2 stops, and decides yes; else, go to step 3).

3) we need to check:

for each walk Mi, check if there is one node in the MAIN PATH Mi that is also contained in a loop? Because there's only one loop that contains the node which lies in the MAIN PATH, we can use SCC algorithm to detect such loop. Then in one MAIN PATH Mi, there may contain several such one loops such as P1', P2', P3' … PK'.

If no, there is no bad walk in Mi;

Else, we need to check the new linear programming constraints satisfy or not of Mi:

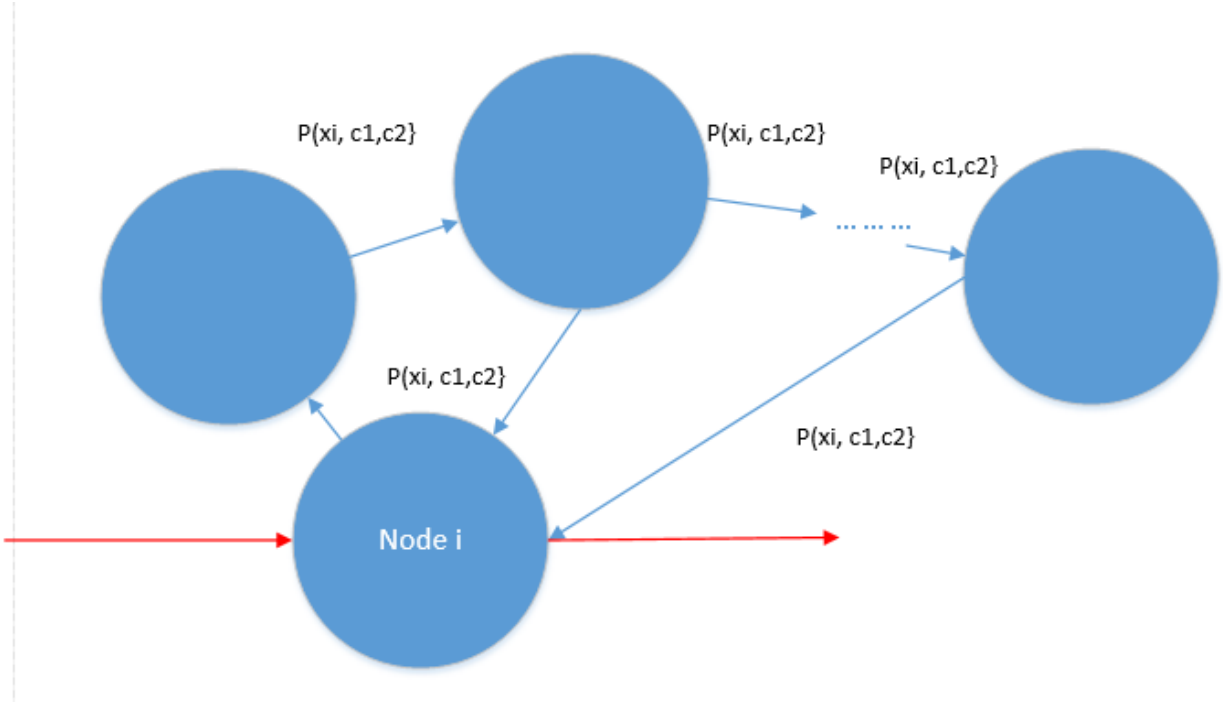
$$\sum_{i=0}^{n} x1\ in\ MAINPATH\ + t1 * (x1\ in\ P1') + \cdots + tk * (x1\ in\ Pk') = 1 + \sum_{i=0}^{n} x2\ in\ MAINPATH\ + t1 * (x2\ in\ P1') + \cdots + tk * (x2\ in\ Pk')$$

$$\sum_{i=0}^{n} x1\ in\ MAINPATH\ + t1 * (x1\ in\ P1') + \cdots + tk * (x1\ in\ Pk') = \sum_{i=0}^{n} (x3\ in\ MAINPATH) + 2 + t1 * (x3\ in\ P1') + \cdots + tk * (x3\ in\ Pk')$$

$$\sum_{i=0}^{n} x3\ in\ MAINPATH\ + t1 * (x3\ in\ P1') + \cdots + tk * (x3\ in\ P1') > 0.26 + \sum_{i=0}^{n} x4\ in\ MAINPATH\ + t1 * (x4\ in\ P1') + \cdots + tk * (x4\ in\ P1')$$

tk is the one loop k execution numbers, When the linear programming holds, there is a bad walk in Mi. After checking Mi (i=1,2,3,…,k), if all Mi don't satisfy the linear programming constraints, the Algorithm-2 decides no.

(3).The directed graph with several nodes that are included in nested loop. To illustrate the idea, here we use a two-nested loop as an example:



this time we use a two-level hashset to make sure that in the second level, the set DFS traversed is unique and every edge is traversed only once. In this way, we will get small cycles respectively.

The algorithm Algorithm-3 now is based on Algorithm-2 but is more robust:

The only difference is we need to detect small cycles in the nested loops and store the information into a data structure. After applying SCC algorithm to allocate the large cycle in the above figure, this time we use a two-level hashset to make sure that in the second level, the set DFS traversed is unique and every edge is traversed only once. In this way, we will get two loops – Loop P, Loop Q separately. Similarly we can generalize this if one MAIN PATH contains k such nested loops, the linear programming constraints will become:

$$\sum_{i=0}^{n} x1 \ + \sum_{k=1}^{k} x1 \ tk + \sum_{k=1}^{k} x1 \ mk = 1 + \sum_{i=0}^{n} x2 \ + \sum_{k=1}^{k} x2 \ tk + \sum_{k=1}^{k} x2 \ mk$$

$$\sum_{i=0}^{n} x1 \ + \sum_{k=1}^{k} x1 \ tk + \sum_{k=1}^{k} x1 \ mk \ = 2 + \sum_{i=0}^{n} x3 + \sum_{k=1}^{k} x3 \ tk + \sum_{k=1}^{k} x3 \ mk$$

$$\sum_{i=0}^{n} x3 + \sum_{k=1}^{k} x3 \ tk + \sum_{k=1}^{k} x3 \ mk \ > 0.26 + \sum_{i=0}^{n} x4 + \sum_{k=1}^{k} x4 \ tk + \sum_{k=1}^{k} x4 \ mk$$

tk is the one small loop k execution numbers, mk is the other small loop k execution numbers.

This solution can be easily generalized to n-nested loops

When the linear programming holds, there is a bad walk in Mi, the algorithm decides Yes. After checking Mi (i=1,2,3,…,k), if all Mi don't satisfy the linear programming constraints, the Algorithm-3 decides no.

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

$$a1, a2, \cdots, a1000$$

where each ai 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.

Answer:

The idea to solve this problem is to use the variance to represent the “unfairness”

According to the question, the roll of the dice is not a fair event, which means there is no guarantee that each side of the dice will have the same chance of appearing in 1/6 in 1000 tests. We can use the frequency representation:

$$\mu = \frac{\sum_{i=1}^{1000} a_i}{1000};$$

$$\sigma^2 = \frac{\sum_{i=1}^{1000} (a_i - \mu)^2}{1000}$$

Then in the “fair” case, the outcomes of 1000 tests should be evenly distributed over six possibilities

$$\overline{\mu} = \frac{1 \times \frac{1000}{6} + 2 \times \frac{1000}{6} + 3 \times \frac{1000}{6} + 4 \times \frac{1000}{6} + 5 \times \frac{1000}{6} + 6 \times \frac{1000}{6}}{1000};$$

$$\overline{\sigma^2} = \frac{\sum_{i=1}^{1000} (a_i - \overline{\mu})^2}{1000}$$

We can use the variance difference ratio to represent the “unfairness” in the end:

$$\theta = \frac{\left|\sigma^2 - \overline{\sigma^2}\right|}{\sigma^2}$$

The greater the value of θ, the greater the degree of “unfairness”.

In the algorithm, we can precompute $\overline{\mu}$ and use $a1, a2, \cdots, a1000$ as input to get the $\sigma^2, \mu, \overline{\sigma^2}$. In the end, we output θ.

However, later I noticed Prof. Zhe Dang gives us the hint to use Kullback-Leibler Divergence to compare two probability distributions.

The ideal distribution of the rolling dice event is a uniform distribution.

And of course we can get our observed distribution of the rolling dice count.

$$H = -\sum_{i=1}^{N} p(x_i) \cdot \log p(x_i)$$

The information gain simply gives the theoretical lower bound on the number of bits we

need, we have a way to quantify exactly how much information is in our data. Now that we can quantify this, we want to quantify how much information is lost when we substitute our observed distribution for a parameterized approximation.

$$D_{KL}(p||q) = \sum_{i=1}^{N} p(x_i) \cdot (\log p(x_i) - \log q(x_i))$$

Essentially, what we're looking at with the KL divergence is the expectation of the log difference between the probability of data in the observed distribution with the approximating distribution (uniform distribution in our case). Note that the KL Divergence is not symmetric, here we use:

$$D_{kl}(\text{Observed} \,||\, \text{Uniform})$$

to decide how “unfair” the dice that the kid tosses is

3. In below, a sequence is a sequence of event symbols where each symbol is drawn from a known finite alphabet. For a sequence $\alpha = a1 \cdots ak$ that is drawn from a known finite set S of sequences, one may think it as a sequence of random variables $x1 \cdots xk$ taking values $xi = ai$, 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 likelihood on the process being i. i. d.

Answer:

Input: a set S, which contains a finite sequences, and these sequences are with the same length n.

Goal: Output the sequence α = a1 · · · ak drawn from the S and output the likelihood on the process being i.i.d.

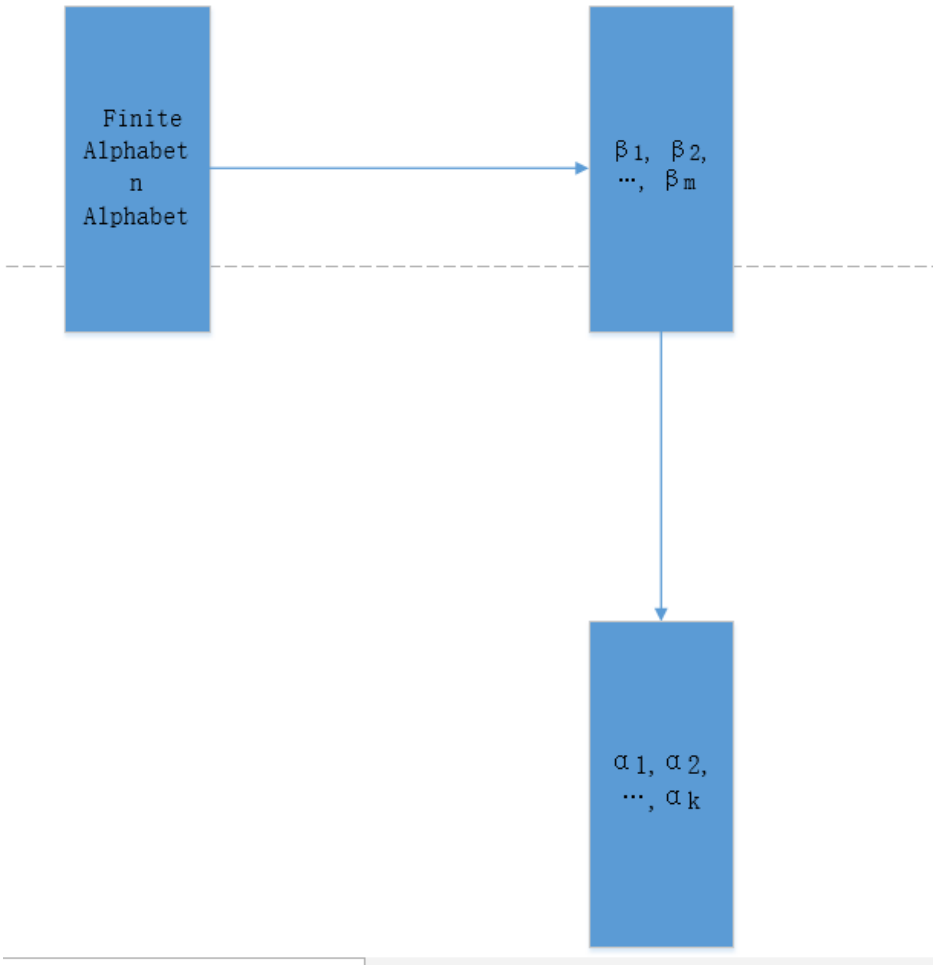
According to the question, we start formulating the problem that we have a set of sequences in S={β1, β2, …, βm}, each βi is with the same length n. (i=1,2,3,…,m).

Then to get a sequence α = a1 · · · ak that is drawn from a known finite set S of sequences, from my best understanding, this question could mean both scenarios:

Scenario 1: choose ai uniformly from βi, (i=1,2,…,k)

Scenario 2: choose ai uniformly from all βj' s ith element, (j=1,2,…m) (i=1,2,…,k)
I personally tend to select scenario 1 to discuss since in this question we don't have value m in S to discuss. If we choose scenario 2, we have one more parameter that is unknown to consider. However, the analysis is pretty similar for both scenarios.

Because the event symbols in set S are dependent events, we need to show that the likelihood ai is independently drawn and all ai obey some kind of identical distribution.



In scenario 1, for each βi, the n elements is uniformly drawn from the Finite Alphabet, the probability is $\frac{1}{N}$. If we choose ai uniformly from βi, then each ai has the probability to be chosen as
$\frac{1}{n}$. To prove the independence, we need to show that for any i ≠j, ai∈ α, aj∈ α, the

conditional probability equals to the probability. We can certainly compute the P(ai = M) and P(ai=M|aj=1,…,N) separately.
If we have P(ai=M|aj=1,…,N) = P(ai=M), then any ai and aj in α are independent. Otherwise, they don't satisfy independence property. Similarly, in scenario 2, we still can compute the P(ai = M) and P(ai=M|aj=1,…,N) separately, if P(ai=M|aj=1,…,N) = P(ai=M), any ai and aj in α are independent. Otherwise, they are not independent.
For all ai satisfied the P(ai |aj)=P(ai) talked above, we add 1 to count. The independent probability of process is count/k.

To prove the identical distribution, we need to collect α = a1 · · · ak first. Then we can select a bunch of known distribution simulators to fit these data points. We have a total combination of choices N regarding a1 · · · ak selection since the alphabet is finite. Then for particular α, we can sequentially put α into distribution analyzers and see whether they obey this distribution or not. Then for each α we can get a satisfied count 1 if α obeys one kind of distribution or 0 if α doesn't obey any distribution. Since α is a finite set, in the end we will get

$$\sum_{i=1} 1\{\alpha \text{ obeys certain distribution}\} = M.$$

In the end, the likelihood of identical distribution is $\frac{M}{N}$.
The total likelihood is $\frac{count}{k} \times \frac{M}{N}$
Another measurement inspired by paper -- *Lossiness of communication channels modeled by transducers,* we can try to compute the information rate of event symbols in S. Here we use L(Si) denote the number of event symbols in a sequence βi (i=1,2,…,m). The information rate $\gamma(\mathrm{Si}) = \lim \frac{logL\ (S_i)}{n}$, since alphabet is finite, we always can have an upper bound of this $\gamma(\mathrm{Si})$.
Then the total likelihood in this measurement representation is $\frac{count}{k} \times \frac{1}{m} \sum_{i=1}^{m} \gamma(Si)$

4. Let G1 and G2 be two directed graphs and v1, u1 be two nodes in G1 and v2, u2 be two nodes in G2. Suppose that from v1 to u1, there are infinitely many paths in G1 and that from v2 to u2, there are infinitely many paths in G2 as well. Design an algorithm

deciding that the number of paths from v1 to u1 in G1 is “more than" the number of paths from v2 to u2 in G2, even though both numbers are infinite (but countable).

Answer:

We can use adjacency matrix to represent G1 and G2. We first locate v1 and u1 in adjacency matrix to prune this matrix and only reserves related information to get A1. Similarly, we can get A2.

A1 is diagonalizable if there is a basis v1, v2, …,vn such that D[A1] - is diagonal. { v1, v2, …,vn} is a basis of Eigenvectors. Similarly for A2.

There is a positive real number r, called the Perron–Frobenius eigenvalue (Perron root), such that r is an eigenvalue of A1 and any other eigenvalue λ (possibly, complex) is strictly smaller than r in absolute value, $|\lambda| < r$. Thus, the spectral radius p(A1) is equal to r.

Similarly, we have p(A2)= r2.

In the paper “AN ALGORITHM FOR COMPUTING THE PERRON ROOT OF A NONNEGATIVE IRREDUCIBLE MATRIX” by PRAKASH CHANCHANA gives the algorithm to find the Perron–Frobenius eigenvalue.

The algorithm detail can be found in the below picture:

1. Compute the LU factorization of

$$(\lambda^{(i)} I - A) = L^{(i)} U^{(i)},$$

and solve for $\tilde{x}^{(i)}$

$$L^{(i)} U^{(i)} \tilde{x}^{(i)} = x^{(i)}$$

2. Use the same LU factors solve for $x^{(i+1)}$

$$L^{(i)} U^{(i)} x^{(i+1)} = \tilde{x}^{(i)}.$$

3. Compute

$$\bar{\lambda}^{(i)} = \lambda^{(i)} - \min_j \left(\frac{\tilde{x}_j^{(i)}}{(B^{(i)} \tilde{x}^{(i)})_j}\right)$$

and

$$\underline{\lambda}^{(i)} = \lambda^{(i)} - \max_j \left(\frac{\tilde{x}_j^{(i)}}{(B^{(i)} \tilde{x}^{(i)})_j}\right)$$

where $1 \leq j \leq n$; Note that the quantity $(B^{(i)} \tilde{x}^{(i)})_j$ is $x_j^{(i+1)}$.

4. Set

$$\lambda^{(i+1)} = \bar{\lambda}^{(i)}.$$

5. Compute

$$error^{(i)} = (\bar{\lambda}^{(i)} - \underline{\lambda}^{(i)}) / \bar{\lambda}^{(i)}.$$

6. If $error > tol$ go back to step 1; otherwise, the Perron root of A is $\lambda^{(i+1)}$.

Then we can compare p(A1) and p(A2) to decide whether that the number of paths from v1 to u1 in G1 is “more than" the number of paths from v2 to u2