Contents

Tarik Pecaninovic 26946831
Question 2

1 2 a)

In this section we compute the transition matrix for the case when n is equal to 4. We first note that the only case of interest is when there are two vertices of each type; in every other case there are no interesting dynamics (either every state is absorbing, or there are no absorbing states.) Next, for ease of discussion, we classify each of the states. As a state can be considered to be four vertices where two have been filled in (as in the diagrams given in the assignment spec sheet,) we conclude that there are n choose four (meaning six in this case) states. I have enumerated each state using diagrams below and given them corresponding names. 'U' stands for up (as the two filled in vertices are on the top), 'D' stands for down (as the two filled in vertices are on the bottom), 'L' and 'R' stand for left and right respectively due to similar reasong, and finally 'M' and 'N' stand for 'Main diagonal' and 'Other Diagonal'.
The method used to compute the transition probabilities is best explained by use of example. There are two cases. First consider that we are currently in the state 'U'. Observe that every vertex is 'happy', hence we are in an absorbing state and the probability of going from 'U' to 'U' is one and every other transition probability is zero. Next assume that we are in the state 'M'. Consider that we are attempting to transition to the state 'L'. For this to occur, first either the bottom left or bottom right vertex must be 'chosen', then the other. Then as this swap is considered 'good', the swap would occur. The probability of choosing the 'correct' initial vertex is 1/4 and the probability of choosing the 'correct' second vertex is 1/3, hence the probability of choosing both vertices correctly is 1/12. But, as a change in order would result in the same swap we have the probability is given by $2/12 = 1/6$. Thus the probability of transitioning from 'M' to 'L' is 1/6. Now consider that we are attempting to transition to the state 'N'. Observe that for this to happen we would have to swap both of the filled in vertices to other positions, which is not possible in one transition. Hence the probability of transitioning from 'M' to 'N' is 0. Repeating the above reasoings

for each state yields the transition matrix shown below. \

To find the canonical form of the matrix found for the case n equal to four we use the script '2a_ canonicalMatrix.py.' It yields the matrix shown below. \

The following describes the technique I used to approximate the absorption times using Montecarlo Approximation. We observe that a state can be modelled by an array where the blank type vertex is represented by a '0' and a filled in type vertex is represented by a '1'. For example the array [1,1,0,0] represents the state 'U'. Using this representation we can model the transition of states in discrete time. At a given time we randomly select two vertices of the current state, decide on whether we should swap the two vertices, and then swap accordingly. Now using this model we can approximate the absorption time of the Markov Chain. We simply give the model a transient state and let it run until it hits an absorbing state. We repeat this a desired amount of times, and then take an average of the times it took to get to the absorbing state. The script '2a _MontecarloAbosrption.py' implements this method. \

The numerical approximation of the abosprtion times of the Markov Chain for n equal to four and five was computed by first finding the corresponding Fundamental Matricies (using numpy) and then summing appropriately. Note that the transition matrix for the case n equals four was found above, but the case for n equals five had to be found by hand. The code implementing this is in the script '2a_ numericalAbsorption.py'.

2 2 b)

The construction of the transition matrix for the case n equals four is described as it was above but with the additional reasoning added that depending a swap that would occur has probability equal to the previous multiplied by one minus epsilon, and a swap that wouldn't occur now has probability equal to the probability of selecting the two vertices multiplied by epsilon. The numerical approximation of the stationary solution was computed in two ways. First was by finding the eigenvector corresponding to the eigenvalue of one. The second was by taking a large power of the transition matrix and inspecting a row. Both of these were implemented using numpy. The following describes the idea and the implementation of the Montecarlo approximation.

We know that, if P is the transition matrix of a Markov Chain, the rows of P^k converge to the stationary distribution as k goes to infinity. Hence if we use a very large value of k we have an approximate stationary solution. For the following assume that k is fixed. We know that the i,j-th entry of P^k tells us the probability of starting in the state i and ending in the state j in k steps. Moreover, since the Markov Chain we have is Ergodic, the starting state i does not matter (i.e., the probability is independent of i). Finally note that the probability of starting at state i and ending at state j in k steps is given by the sums of the probabilities of the different paths from i to j. Using this knowledge we can approximate the stationary solution using a Montecarlo technique. Using the above model, we start at an arbitrary state and run the model for k steps–keeping track of the probabilty of the current path–and finally summing them together. If we iterate this process and then take the average, we end up with an approximate stationary solution. The script 2b_ nEquals4Ergodic.py implements this method and also the numeric approximation described above. We observe that in this model, in the long run we want to be in states U,D,L, or R. Meaning we can partition our cycle into two intervals, one filled with type 1 and the other with type 0. \

The analysis above was repeated for a linear structure by modifying the decision function in the code. The script is named 2b_ linearstucture.py. We observe that in the long run we want to end up in states L of R. Meaning if we set our space be linear we have our space being partitioned into the two types (the two types try to stay away from each other, similar to the cycle model). Moreover, as there are less states which partition our space like this, the probability of moving to these states in the long run is much higher than the previous model. \

Some reasonable extensions to this model include: we could change the condition under which we allow two vertices to swap, change the definition of a vertex being 'happy', change the number of vertices we swap at a time, change the structure of the graph (e.g., adding more edges). First, observe that this last statement increases the computational complexity of the model, but also makes it easier for every vertex to be 'happy'. Observe that with the first statement, any changes to the definition would hinder the model as now we are allowing swaps which would 'worsen' the state. Observe that with the second statement, as currently every vertex has either one or two neighbours, the only changes one could make to the definition would make it so that there will always exist a vertex which cannot be happy. Observe that the only effect the third statement makes to the model is that it is essentially

like doing multiple transitions at once, which adds no meaningful addition to the model. From this we conclude that the current model is at a 'good level' of realistic vs tractable.