# Asmt 6: Graphs

Gopal Menon
Turn in through Canvas by 5pm:
Monday, May 1

## 1 Finding $q*$ (50 points)

**A: (20 points):** Run each method (with $t = 1024$, $q_0 = [1, 0, 0, \ldots, 0]^T$ and $t_0 = 100$ when needed) and report the answers.

Table 1: Value of $q*$ vector

| Matrix Power | State Propagation | Random Walk | Eigen Analysis |
|---|---|---|---|
| $\begin{bmatrix}0.035758\\0.057212\\0.058092\\0.079217\\0.085818\\0.066014\\0.157905\\0.171636\\0.137309\\0.151040\end{bmatrix}$ | $\begin{bmatrix}0.035758\\0.057212\\0.058092\\0.079217\\0.085818\\0.066014\\0.157905\\0.171636\\0.137309\\0.151040\end{bmatrix}$ | $\begin{bmatrix}0.00000\\0.50000\\0.50000\\0.00000\\0.00000\\0.00000\\0.00000\\0.00000\\0.00000\\0.00000\end{bmatrix}$ | $\begin{bmatrix}0.035758\\0.057212\\0.058092\\0.079217\\0.085818\\0.066014\\0.157905\\0.171636\\0.137309\\0.151040\end{bmatrix}$ |

**B: (10 points):** Rerun the *Matrix Power* and *State Propagation* techniques with $q_0 = [0.1, 0.1, \ldots, 0.1]^T$. For what value of $t$ is required to get as close to the true answer as the older initial state?

Table 2: Matrix Power

| $t = 64$ | $t = 128$ | $t = 256$ | $t = 512$ |
|---|---|---|---|
| $\begin{bmatrix}0.035711\\0.057177\\0.058120\\0.079316\\0.085925\\0.066098\\0.158001\\0.171422\\0.137112\\0.151117\end{bmatrix}$ | $\begin{bmatrix}0.035756\\0.057215\\0.058091\\0.079218\\0.085819\\0.066013\\0.157925\\0.171612\\0.137289\\0.151062\end{bmatrix}$ | $\begin{bmatrix}0.035757\\0.057212\\0.058092\\0.079217\\0.085818\\0.066014\\0.157906\\0.171636\\0.137308\\0.151040\end{bmatrix}$ | $\begin{bmatrix}0.035758\\0.057212\\0.058092\\0.079217\\0.085818\\0.066014\\0.157905\\0.171636\\0.137309\\0.151040\end{bmatrix}$ |

With $t = 512$, the value of $q*$ is equal to the value obtained with the older initial state $q_0$ for Matrix Power. The Matrix Power function was implemented using $M^{2i} = M^i \times M^i$ and so it can only be run for powers of 2. This is shown in table 2.

The State Propagation method obtained the value of $q*$ is equal to the value obtained with the older initial state $q_0$ with $t = 285$. This is shown in table 3.

Table 3: State Propagation

| $t = 282$ | $t = 283$ | $t = 284$ | $t = 285$ |
|---|---|---|---|
| $\begin{bmatrix}0.035757\\0.057212\\0.058092\\0.079217\\0.085818\\0.066014\\0.157905\\0.171636\\0.137309\\0.151040\end{bmatrix}$ | $\begin{bmatrix}0.035758\\0.057212\\0.058092\\0.079217\\0.085818\\0.066014\\0.157905\\0.171636\\0.137309\\0.151039\end{bmatrix}$ | $\begin{bmatrix}0.035757\\0.057212\\0.058092\\0.079217\\0.085818\\0.066014\\0.157905\\0.171636\\0.137309\\0.151040\end{bmatrix}$ | $\begin{bmatrix}0.035758\\0.057212\\0.058092\\0.079217\\0.085818\\0.066014\\0.157905\\0.171636\\0.137309\\0.151040\end{bmatrix}$ |

**C: (12 points):** Explain at least one **Pro** and one **Con** of each approach. The **Pro** should explain a situation when it is the best option to use. The **Con** should explain why another approach may be better for some situation.

The Matrix Power and State Propagation approaches are mathematically the same. The reason is that $q* = M^t q_0$ and $q_{i+1} = M \times q_i$ done $t$ times are equivalent since the latter method results in $M$ being raised to the power $t$.

1. **Matrix Power:**

   - **Pros:** This method is easy to implement and works and is the best method for small matrices.
   - **Cons:** It can be very compute intensive for large matrices. Also, when recursion is used, each recursion level will need to store half the matrix at the previous level. We cannot predict what power the matrix will need to be raised to in order to find the solution. The State Propagation method will be better suited for large matrices.

2. **State Propagation:**

   - **Pros:** This will work for large graphs as we only need to do many matrix multiplications of the transition matrix with the state vector. In the case of the world wide web, where this is the best method to use, 50 to 75 multiplications reportedly reach a steady steady state. Since the matrix for the world wide web is sparse, the matrix can be efficiently stored. Also the work can be split up by using the MapReduce framework.
   - **Cons:** We cannot predict how many matrix to state vector multiplications we will need to do in order to find the solution. The Eigen Analysis method will give us the answer in one step.

3. **Random Walk:** I did not understand how the Random Walk method is supposed to work. So I'm leaving this method out from the pros and cons analysis.

4. **Eigen Analysis:**

   - **Pros:** The Eigen Analysis method will give us the exact answer and is the best method to use when we need an exact and not an approximate result. Once we get $q*$, if we multiply the transition matrix again by $q*$, we should get $q*$ again.

$$M \times q* = q*$$
$$= \lambda \frac{q*}{\|q*\|_1}$$

where $\lambda$ is some constant that keeps the equality relation true. This means that $q*$ is the principal left eigen vector of $M$.

- **Cons:** This method is only suitable for small matrices where it is easy to do the computation. Even though the Eigen Analysis method will give us the exact answer, it may not always be possible to use it in the case of a graph for the world wide web since the matrix size will be very large. For the world wide web, the State Propagation method is best.

**C: (12 points):** Is the Markov chain *ergodic*? Explain why or why not.

The Markov chain seems to be ergodic as the state probability vector is converging for values of $t$ shown in tables 2 and 3 above. If the Markov chain had not been ergodic $M* \neq M^n$ as $n \to \infty$.

Since the final state vector does not have a zero in any position, it means that there is a probability that a random walker on the graph can end up at any state. This means that the Markov chain is ergodic.

**D: (8 points):** Each matrix row and column represents a node of the graph, label these from 1 to 10 starting from the top and from the left. What nodes can be reached from node 4 in one step, and with what probabilities?

Table 4: Nodes that can be reached from node 4 in one step

| Node | Probability |
|---|---|
| 3 | 0.3 |
| 5 | 0.3 |
| 6 | 0.4 |

## 2 BONUS: Taxation (5 points)

Repeat the trials in part **1.A** above using taxation $\beta = 0.85$ so at each step, with probability $1 - \beta$, any state jumps to a random node. It is useful to see how the outcome changes with respect to the results from Question 1. Recall that this output is the *PageRank* vector of the graph represented by $M$.
Briefly explain (no more than 2 sentences) what you needed to do in order to alter the process in question 1 to apply this taxation.

If $M$ is the state transition matrix and $P$ is a matrix with the same shape as $M$ where each element has a value of $\frac{1}{m}$, where $m$ is the number of nodes in the graph, then the matrix $M$ used in **1.A** can be replaced by the matrix $\beta M + (1 - \beta)P$. Table 5 shows the values obtained by using the new matrix instead of the one used in **1.A**.

Eigen Analysis returned a vector with negative values and the vector shown has the negative values removed. In the general case it is possible to have an eigen vector with negative values, but in this case negative values are not valid probabilities. Also the Random Walk method did not give the correct result. I'm not sure why as I did not understand why this method would work.

Table 5: Value of vector

| Matrix Power | State Propagation | Random Walk | Eigen Analysis |
|---|---|---|---|
| $\begin{bmatrix}0.058527\\0.079006\\0.082942\\0.109283\\0.117222\\0.092012\\0.123235\\0.126784\\0.101247\\0.109742\end{bmatrix}$ | $\begin{bmatrix}0.058527\\0.079006\\0.082942\\0.109283\\0.117222\\0.092012\\0.123235\\0.126784\\0.101247\\0.109742\end{bmatrix}$ | $\begin{bmatrix}0\\0\\0\\0\\0\\0\\0\\0\\0\\1\end{bmatrix}$ | $\begin{bmatrix}0.058527\\0.079006\\0.082942\\0.109283\\0.117222\\0.092012\\0.123235\\0.126784\\0.101247\\0.109742\end{bmatrix}$ |