CMS/CS/Ec/EE 144 – HW 4: PageRank & Centrality

Marco Yang

Collaborators: Nora Xiao, Jonathan Lin

1. **Approximately Central [30 points]**

   For large scale graphs that contains a million or even more nodes, some familiar centrality measures, such as betweenness centrality or closeness centrality, are prohibitively expensive to compute. For example, calculating the closeness centrality on a network of 24 million nodes using standard computing resources takes an astonishing 44,222 hours, or around 5 years, to compute. Even computing pagerank can be a difficult task in many cases. Therefore, it is often important to compute centrality using approximated methods. In this problem, you will try to use machine learning to train a model that approximates betweenness centrality on real datasets.

   We consider a situation with small training networks (potentially subgraphs) and a large target graph. Our objective is to train a neural network on the small graph and evaluate it on the large graph. Specifically, we consider a set of p2p networks using Gnutella. The dataset we use consisting of 9 networks ranging from 6,300 to 63,000 nodes. In order to train the model, we have to preprocess the dataset first. We use a Struc2Vec node embedding technique to represent the information of the graphs using generated graph vectors. Then the model can be trained using graph vectors and their corresponding ground truth. We recommend you to use Google Colab to solve this problem. We also provide a Colab notebook to give you a step-by-step guideline. To run this notebook, you need to copy it into your own google drive and upload the datasets we provided to your drive as well. Specific instructions are included in the notebook.

   Your task:

   - The provided notebook includes a fully functional model training pipeline: preprocessing graph data, embedding the graph information and training the model. We don't require you to modify any given function for this problem. After the training, the pipeline computes a Kendall-tau similarity score between the predicted centrality scores and their ground-truth (the exact betweenness function). We expect you to get a Kendall-tau score at least 0.7. In order to achieve this objective, you need to choose four pre-defined parameters wisely. We hope you can get familiar with this training pipeline by playing around the parameters. Turn in a description of your evaluations, including the best parameters you found and any connection you see to graph structure. (This description can be a brief paragraph on what you noticed with varying some subset of the following: embedding size, # of layers, # of folds for cross-validation, # of epochs, etc.)
   - Evaluate your trained model on a larger graph. In this exercise, you will need to write your own code to evaluate your trained model on p2p-Gnutella04 dataset. We provide some hints in the Colab notebook. Report the Kendall-tau score you get for the evaluation.

   Solution: Code is here:
   - notebook code
   - training code for hparam comparisons
   - visualization of hparam comparison

   First off, I don't think the number of folds in k-fold cross validation should have any effect on the actual Kendall-tau of the model on the training data. The number of folds should just be there to give you an estimate of how well a model generalizes, not impact performance. However, in the given code, we are evaluating the model trained on everything but the last holdout fold, so I changed it to train on all the data after it did the cross-validation. I also changed the model to train fully on all data after compute k-fold cross-validation scores.

After running some experiments with different hyperparameters, I came to the following conclusions:
- 5 epochs is enough for convergence (Figure 1).
- increasing embedding size to 256 and number of layers to 5 helps the most with reaching the 0.7 Kendall-tau mark (see notebook for top 10 hyperparameter config ranking)
- the optimal number of layers is around 16 (Figure 2).
  - increasing the number of layers to 16 with an embedding size of 256 achieved 0.52 Kendall-tau on test set with 0.68 Kendall-tau on train set (up to 0.7 with good random seeding)
  - also had the lowest validation MSE for 5-fold CV
- deeper neural networks require gradient clipping or else the weights will explode.
- graph seems somewhat "symmetric" up to 10 folds since validation MSE decreases compared to 5 folds. didn't feel like testing more than 10 folds since rerunning experiments takes a while and I'm kinda close to submission deadline (Figure 3).

One configuration that achieved 0.7 Kendall-tau was 256 embedding size, 5 layers, 20 epochs (see notebook). For some reason (maybe due to seeding?) the same config didn't reach 0.7 Kendall-tau in my training script, but it seemed pretty close (0.67).

I've only attached the relevant graphs for my conclusions, but all graphs of validation loss/ Kendall-tau vs each hyperparameter can be found in the notebook.

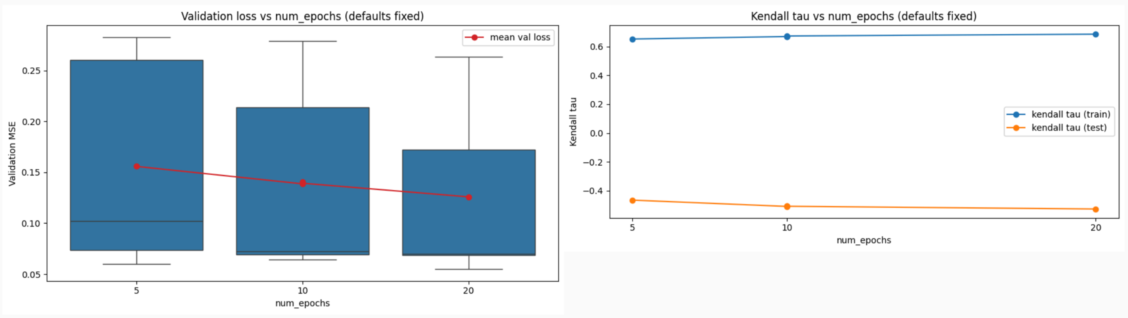

Figure 1: Metrics vs Epochs

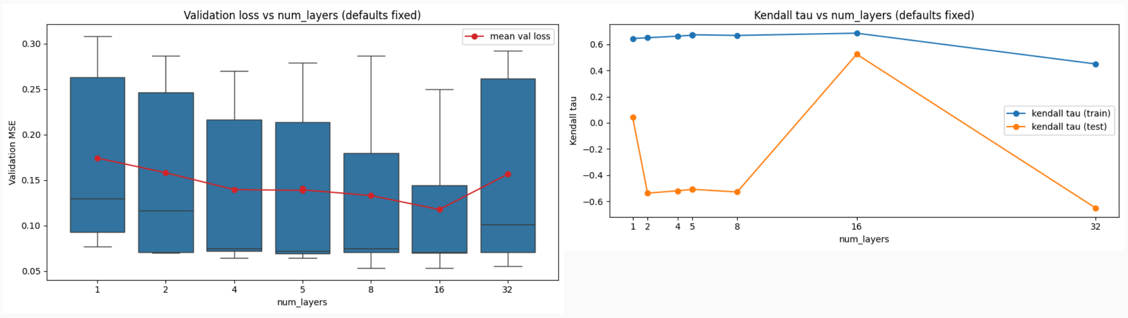

Figure 2: Metrics vs Number of layers

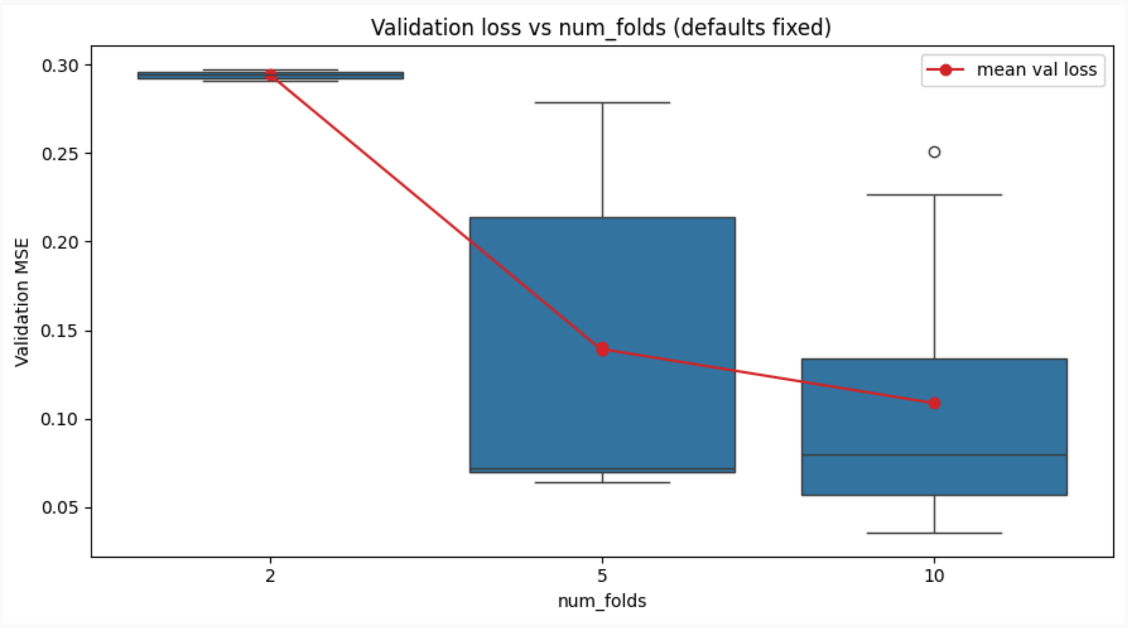

Figure 3: CV MSE vs Folds

2. **Bernoulli: Your Own Caltech Search Engine [40 points]**

In this problem, you will build a working search engine for the Caltech domain that uses PageRank to improve search results. This will give you hands-on experience with the PageRank algorithm we've studied in class, and let you see how it performs on a real web graph. You will implement the core PageRank algorithm, then use it to rank search results alongside traditional text-based relevance scoring. The starter code provides a complete web crawler, indexer, and search interface. All you need to do is implement the PageRank computation itself to help rank search results for relevance.

Your task:

(a) [20 points] Implement the `compute_pagerank()` function in `src/pagerank.py`. Your implementation should:
- Take as input a web graph (represented as a dictionary mapping URLs to lists of outgoing links) and the damping factor α
- Return a dictionary mapping each URL to its PageRank score
- Use the iterative power method discussed in class: $\pi = \lim_{n\to\infty} \pi_0 G^n$ where $G = \alpha P + \frac{1-\alpha}{n}(1_{n\times n})$
- Handle pages with no outgoing links appropriately
- Converge to a reasonable tolerance (e.g., when the maximum change in any PageRank score is less than 10^{-7})

The function signature and detailed specifications are provided in the starter code. Once implemented, run `compute_pagerank.py` to compute scores for your crawled web graph.

Solution: link to code.

(b) [20 points] Use your search engine! First, crawl the Caltech domain by running `crawl.py`. Then compute PageRank scores by running `compute_pagerank.py` and perform at least 5 different searches using `search.py`. In your submission, include:

- A brief description of your crawling results (number of pages indexed, any interesting observations)
- The top 10 pages by PageRank from your crawl
- Your 5 search queries and a discussion of the quality of results for each query. Do the results make sense? How does PageRank affect the ranking compared to pure text relevance?

Solution: I crawled the default 5000 pages. The order showed that pages with the same subdomain (e.g. a lab, catalog, etc…) would be crawled consecutively. This makes me feel like the crawled results could be heavily skewed to just a few labs or just a few subdomains, which probably isn't ideal if we want a little bit of everything.

The top 10 pages were

```
Top 10 pages by PageRank:
  1. 0.028467  https://www.caltech.edu
  2. 0.019154  https://www.caltech.edu/privacy-notice
  3. 0.017281  https://digitalaccessibility.caltech.edu
  4. 0.004371  https://thesis.library.caltech.edu
  5. 0.003022  https://directory.caltech.edu
  6. 0.002795  https://magazine.caltech.edu
  7. 0.002708  https://sites.caltech.edu
  8. 0.002657  https://digitalaccessibility.caltech.edu/accessibility-
guidelines-for-content-creators
  9. 0.002656  https://hr.caltech.edu/resources/notices-administrative-
guidelines/digital-accessibility
  10. 0.002475  https://www.caltech.edu/claimed-copyright-infringement
```

My first query was "courses 2025". The results were pretty good, but I did notice that the word relevance scores for the top 10 pages were all really similar. What decided the ranking was the PageRank scores. The main catalog page had a higher PageRank score than the 2025-2026, which ranked higher than the 2024-2025 page. Ideally, I feel like the 2025-2026 page ranks first, followed by the 2024-2025 page, followed by the main catalog, but I guess the main catalog hogs the "click traffic". Starting from the 6th result and onwards, it was a bunch of random stuff from the Keck Institute, probably because we didn't crawl enough pages, and like I said in results for the crawl, a few subdomains can easily hog all of the index.

My second query was "honor code". The real Honor Code and Community Standards page was the second result, behind Quick Links for Students. Like before, the PageRank score seems to have done more harm than good by reducing the semantic signal and instead prioritizing pages with higher traffic.

My third query was computer vision, which failed massively because there weren't enough pages indexed. Also, the keyword matching matched "vision" with "division" in a few of the top ranking results.

My fourth query was "financial aid". This seems like a pretty crucial query, and once again PageRank pushed down the relevant pages to 2nd, 5th, and 6th, with "high traffic" pages like "Caltech at a Glance" and "Information for Graduate Students" taking the top results.

My fifth query was "cs 144". Everything ended up being a course catalog page ranked in order of the PageRank score. Nothing special.

3. **Pandemaniac Warm-Up [15 points]**

   In the coming week, you will be working on spreading an epidemic through a graph for Pandemaniac. You will do Pandemaniac in teams of 2 or 3 (no more than 4), so you should work to form your team this week and look out for the Piazza post asking for your group members and team name. In the meantime, this question is designed to help prepare you for this task.

   In Pandemaniac, the epidemics you are working with will work as follows. Each round begins with the teams selecting some subset of the nodes on the graph. Each team is assigned a different color, and any nodes claimed by a team are assigned that team's color. Each node may be claimed by at most one team, but nodes can also start as unclaimed, in which case they remain uncolored. These uncolored nodes may then be convinced to adopt a color based on the colors of their neighbors. After they (or any nodes) adopt a color, they still may later be convinced to switch their color if their neighbors switch. From the starting nodes, the epidemics spread iteratively. During each iteration, every node chooses its next color through a majority vote among itself and its direct neighbors. All colored direct neighbors of the node vote for their respective color with 1 vote; however, if the node itself is currently colored, it votes for its own color with 1.5 votes. If any color has a strict majority out of the total number of votes, then that becomes the next color for the node. If there is no strict majority, then the node keeps its current color or remains uncolored.

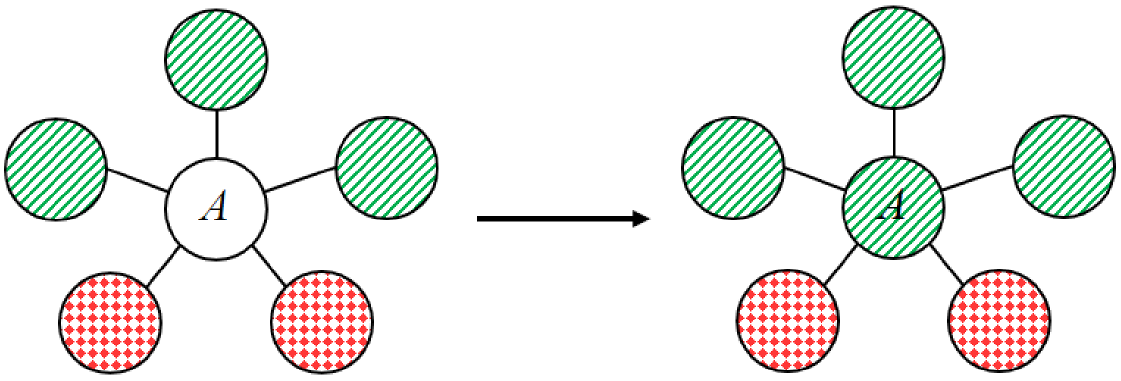

Figure 4: An uncolored node A converting to "striped" (since "spotted" gets 2 votes and "striped" gets 3 votes, which is a majority among the 5 votes).

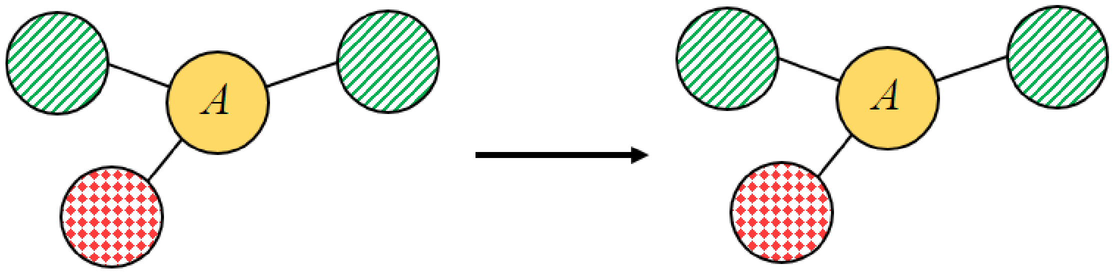

Figure 5: Node A doesn't change color because no color got the majority of votes ("striped" gets 2 votes, "dotted" gets 1 vote, "solid" gets 1.5 votes, none of them gets a majority among the 4.5 votes).

Your task: Is it necessary that a graph's epidemic colors always converge or stabilize? If yes, prove that this must always be true. Otherwise, give a counterexample in the form of a graph and initial coloring that do not converge.

Solution: The graph's colors don't always converge.

An example of a graph that stabilizes is a triangle with one node initially being red and the two other nodes being blue. After the first iteration, all nodes become blue, and nothing changes afterwards.

An example of a graph that never stabilizes is a loop graph of 2 colors with an even number of nodes, with consecutive nodes having different colors (e.g. blue, red, blue, red). Every node has 2 neighbors with the opposite color, so it will always switch to the opposite color during an iteration. For example, if a node is currently red, the node to the left and right are both blue, and so there are 1.5 votes for red and 2 votes for blue, in which case the node switches to blue. Vice versa if the node is already blue. After the iteration is over, we are left with a loop of alternating colors again, except every node has the opposite color from before. Clearly, this graph never stabilizes.

4. **Warmup with PageRank and stationary distributions [5 points]**

In class, we saw that PageRank can be viewed as calculating the stationary distribution of a transition matrix defined by a graph. In this problem we'll investigate the differences between various ways of calculating the stationary distribution. In particular, we have seen two ways of calculating the stationary distribution of a transition matrix P. One is the iterative method $\pi = \lim_{n\to\infty} \pi_0 P^n$ for some initial π_0, and the other one is to solve the equation $\pi = \pi P$.

For the following probability transition matrices, first use $\pi = \pi P$ to get the stationary distribution, and then show whether or not $\pi_0 P^n$ converges as $n \to \infty$. If it converges, show that $\pi = \lim_{n\to\infty} \pi_0 P^n$ does not depend on π_0 and interpret what this means about the stationary distribution.

Hint: you may want to diagonalize P to handle P^n easily. Feel free to use Mathematica/MATLAB to diagonalize P.

(a)
$$P = \begin{pmatrix} \frac{2}{5} & \frac{3}{10} & \frac{3}{10} \\ \frac{1}{5} & \frac{3}{5} & \frac{1}{5} \\ \frac{7}{10} & \frac{1}{10} & \frac{1}{5} \end{pmatrix}$$

Solution: From Wolfram Alpha, the diagonalization is $P = UDU^{-1}$, where

$$U \approx \begin{pmatrix} 1 & -0.472555 & 0.359347 \\ 1 & -0.141606 & -1.06594 \\ 1 & 1 & 1 \end{pmatrix}$$

$$D \approx \begin{pmatrix} 1 & 0 & 0 \\ 0 & -0.144949 & 0 \\ 0 & 0 & 0.344949 \end{pmatrix}$$

$$U^{-1} \approx \begin{pmatrix} 0.4 & 0.36 & 0.24 \\ -0.894022 & 0.277238 & 0.616784 \\ 0.494022 & -0.637238 & 0.143216 \end{pmatrix}.$$

The stationary distribution is the left eigenvector associated with the eigenvalue, which is the first row of U^{-1}:

$$\pi = (0.4\ \ 0.36\ \ 0.24).$$

To find whether or not $\lim_{n\to\infty}$ converges, we first find

$$\begin{aligned}
\lim_{n\to\infty} \pi_0 P^n &= \lim_{n\to\infty} \pi_0 (UDU^{-1})^n \\
&= \lim_{n\to\infty} \pi_0 U D^n U^{-1} \\
&= \lim_{n\to\infty} \pi_0 U \begin{pmatrix} 1^n & 0 & 0 \\ 0 & (-0.144949)^n & 0 \\ 0 & 0 & 0.344949^n \end{pmatrix} U^{-1} \\
&= \pi_0 U \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} U^{-1} \\
&= \pi_0 \begin{pmatrix} 1 & -0.472555 & 0.359347 \\ 1 & -0.141606 & -1.06594 \\ 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} 0.4 & 0.36 & 0.24 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \\
&= \pi_0 \begin{pmatrix} 0.4 & 0.36 & 0.24 \\ 0.4 & 0.36 & 0.24 \\ 0.4 & 0.36 & 0.24 \end{pmatrix} \\
&= (\pi_{01}\ \ \pi_{02}\ \ \pi_{03}) \begin{pmatrix} 0.4 & 0.36 & 0.24 \\ 0.4 & 0.36 & 0.24 \\ 0.4 & 0.36 & 0.24 \end{pmatrix} \\
&= (0.4(\pi_{01}+\pi_{02}+\pi_{03})\ \ 0.36(\pi_{01}+\pi_{02}+\pi_{03})\ \ 0.24(\pi_{01}+\pi_{02}+\pi_{03})) \\
&= (\pi_{01}+\pi_{02}+\pi_{03})\pi.
\end{aligned}$$

Thus, no matter what π_0 is (besides the null vector), it always converges to a scalar multiple of π, and the stationary distribution is always achievable via iteration.

(b)

$$P = \begin{pmatrix} 0 & \frac{5}{8} & 0 & \frac{3}{8} \\ 1 & 0 & 0 & 0 \\ 0 & \frac{3}{8} & 0 & \frac{5}{8} \\ \frac{3}{4} & 0 & \frac{1}{4} & 0 \end{pmatrix}$$

Solution: The diagonalization of P (according to Wolfram Alpha Mathematica) is

$$P = UDU^{-1}$$

where

$$U \approx \frac{1}{6}\begin{pmatrix} -6 & 1 & -1 & 6 \\ 6 & -4 & -4 & 6 \\ -6 & -9 & 9 & 6 \\ 6 & 6 & 6 & 6 \end{pmatrix}$$

$$D \approx \frac{1}{4}\begin{pmatrix} -4 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 4 \end{pmatrix}$$

$$U^{-1} \approx \frac{1}{20}\begin{pmatrix} -9 & 6 & -1 & 4 \\ 6 & -6 & -6 & 6 \\ -6 & -6 & 6 & 6 \\ 9 & 6 & 1 & 4 \end{pmatrix}.$$

The stationary distribution is the left eigenvector corresponding to the eigenvalue 1 (third row of U^{-1}):

$$\pi = \begin{pmatrix} -6 & -6 & 6 & 6 \end{pmatrix}$$

Since π contains negative and positive values, it can't be normalized as a probability distribution. Thus, there is no stationary distribution.

Like before, for convergence, we try to find D^{∞}:

$$\lim_{n\to\infty} (\frac{1}{4}\begin{pmatrix} -4 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 4 \end{pmatrix} = \begin{pmatrix} (-1)^n & 0 & 0 & 0 \\ 0 & \left(-\frac{1}{4}\right)^n & 0 & 0 \\ 0 & 0 & \left(\frac{1}{4}\right)^n & 0 \\ 0 & 0 & 0 & 1^n \end{pmatrix}.$$

Clearly, the top left value $(-1)^n$ doesn't converge. Thus, the iterative process doesn't converge.

5. **Training to be a farmer [10 points]**

A typical way to raise the PageRank of a page is to use "link farms", i.e., a collection of "fake" pages that point to yours in order to improve its PageRank. Our goal in this problem is to do a little analysis of the design of link farms, and how their structure affects PageRank calculations.

Consider the web graph. It contains n pages, labeled 1 through n. Of course, n is very large. As mentioned in class, we use the notation $G = \alpha P + \frac{1-\alpha}{n}(1_{n\times n})$ for the transition matrix. Let r_i denote the PageRank of page i, and $r = (r_1, r_2, ..., r_n)$ denote the vector of PageRanks of all pages.

Note: For a page that has k outgoing links, we put $\frac{1}{k}$ for the corresponding entries of P. However, when a webpage has no outgoing links, we add a 1 as the corresponding diagonal element of P for making its row-sum one. Note that this makes G a valid transition probability matrix.

(a) You now create a new web page X (thus adding a node to the web graph). X has neither in-links, nor out-links. Let $\tilde{r} = (\tilde{r}_1, \tilde{r}_2, ..., \tilde{r}_n)$ denote the vector of new PageRanks of the n old web pages, and x denote the new PageRank of page X. In other words, $(\tilde{r}_1, \tilde{r}_2, ..., \tilde{r}_n, x)$ is the

PageRank vector of the new web graph. Write $\tilde{r}$ and x in terms of r. Comment on how the PageRanks of the older pages changed due to the addition of the new page (remember n is a very large number).

Hint: Use the stationary equations to calculate PageRank, not the iterative approach.

Solution: We know that the PageRank algorithm solves:

$$r_i = \alpha \sum_{j \in I(i)} \frac{r_j}{L(j)} + \frac{1-\alpha}{n}$$

where $I(i)$ is the set of inbound edges to i.

After updating G to include X, we know that X doesn't affect $I(i)$ or $L(j)$ in the summation in r_i for $i < n+1$ at all since there are no inbound/outbound links to X. The only thing that changed was $n \to n+1$, so

$$\tilde{r}_i = \alpha \sum_{j \in I(i)} \frac{\tilde{r}_j}{L(j)} + \frac{1-\alpha}{n+1}.$$

Since for large n, $n \approx n+1$, we can just say $\tilde{r}_i = r_i \Rightarrow \tilde{r} = r$.

Plugging in the fact that X has no inbound edges besides its self loop,

$$x = \alpha x + \frac{1-\alpha}{n+1}$$
$$x = \frac{1}{n+1}.$$

(b) Unsatisfied with the PageRank of your page X, you create another page Y (with no in-links) that links to X. What are the PageRanks of all the $n+2$ pages now? Does the PageRank of X improve?

Solution: Like before, since X and Y have no edges to the rest of the web pages and we can approximate $n \approx n+2$ for large numbers, $\tilde{r} \approx r$.

The pagerank of Y (no inbound edges at all since it points to X) is

$$y = \alpha \cdot 0 + \frac{1-\alpha}{n+2}$$
$$y = \frac{1-\alpha}{n+2}.$$

The pagerank of X now that it has one inbound edge is

$$
\begin{aligned}
x &= \alpha(x+y) + \frac{1-\alpha}{n+2} \\
x(1-\alpha) &= \alpha y + \frac{1-\alpha}{n+2} \\
x &= \frac{\alpha y}{1-\alpha} + \frac{1}{n+2} \\
x &= \frac{\alpha}{1-\alpha} \cdot \frac{1-\alpha}{n+2} + \frac{1}{n+2} \\
x &= \frac{\alpha+1}{n+2}.
\end{aligned}
$$

(c) Still unsatisfied, you create a third page Z. How should you set up the links on your three pages so as to maximize the PageRank of X? Justify your answer.

Solution: If we consider the iterative process of computing PageRank, intuitively, we are sending an equal amount of the current PageRanks of every page to its neighbors after scaling it down by a factor of α. Since our goal is to maximize the PageRank of X, we know that we should directly connect Y, Z to X because otherwise the initial PageRanks of Y, Z will be reduced by a factor of α^d before reaching X, where d is the distance from the other pages to X. We also shouldn't create links between Y and Z because that would halve the amount of PageRank being directly transferred to Z.

Now, we just need to figure out whether or not to create links from X back to Y or Z. Using the same intuition as before, if we lose $1 - \alpha$ of the score with each extra edge, we wouldn't want to route X back to Y or Z ever. Thus, our optimal structure is just $Y \to X, Z \to X$.