Modifying MCMC Metripolis Hastings/Random Walk for a new University Ranking Metric

Yiran Jia, Yiming Ding, Raymond Feng, Tiancheng Cai

April 18, 2018

1 Introduction

University ranking lists such as US News Top 100 Colleges serve as a resource for high school students and parents to gauge the relative prestige of a school. The metrics commonly used are along the lines of, number of academic publications, or perhaps Nobel laureates on the faculty. With this in mind, we set out to use PageRank inspired MCMC/Random Walk techniques to create a new metric for a university's prestige that is not used in any known college ranking scheme. The central assumption is: the prestige of a university is heavily based on the importance/impact of the alumni. After all, successful entrepreneurs, athletes, scientists, and politicians often donate heftily to their alma maters(Think the likes of Sehat Sutardja and Weili Dai, sound familiar?). The trouble now is, how do we put a number on a person's "importance"? There is no definite correct answer, but we chose to model a person's importance as proportional to the traffic of his/her Wikipedia page. The alma mater of this important figure then in turn also gains some importance.

2 Methods: Theory and Pseudocode

Each Wikipedia page [1] can be viewed as a node, and its edges are all the links on that Wikipedia page that lead to other Wikipedia pages. The idea is to perform a random walk. At each page, the next page is selected uniformly at random from all links on the current page. However there are a few issues:

- Unexpected pathologies lead to random walks going to a dead end or loop
- Choosing successors uniformly overemphasizes pages with more of links
- Will we converge to stationary?

[1] "Wikipedia page" will be used in place of "Wikipedia page of a person" for convenience. All Wikipedia pages, not excluding non-persons, can be used as well with a slight modification of the code, but we will stick to doing a random walk on only pages of people for continuity.

The first problem can be handled by restarting the random walk after a certain amount of steps, but the second and third problems are more egregious, and need to be addressed in a more involved way. We decided to apply ideas from the MCMC Metropolis Hastings algorithm, which has acceptance function and proposal density respectively as:

$$A(x,y) = min\{1, \frac{\pi(y)f(y,x)}{\pi(x)f(x,y)}\}, f(x,y) = \frac{1}{\sqrt{2\pi\sigma^2}}e^{-\frac{(x-y)^2}{2\sigma^2}}$$

However, we have to redefine these functions because unlike the case of modeling a Normal distribution, the proposal function does not really make sense for our application. To redefine the proposal density, we define some new values:

$$v(x) := \text{"views on page x"}$$

$$v_{total}(x) = \sum_{\text{link } l \in x} v(l) := \text{"sum of views of all links on page x"}$$

Then, we define the proposal density as follows:

$$f(x,y) := \frac{v(y)}{v_{total}(x)}$$

There are two options for the stationary, and we will explore both. We can assume an uniform stationary, which corresponds to all pages being selected at equal probabilities:

$$\pi(x) = \pi(y) \Rightarrow A(x,y) = \frac{v_{total}(x)}{v_{total}(y)}$$

Or we can have the stationary be proportional to the number of views on that page:

$$\frac{\pi(x)}{\pi(y)} = \frac{v(x)}{v(y)} \Rightarrow A(x,y) = \frac{v(x)v_{total}(x)}{v(y)v_{total}(y)}$$

$$\pi(s')A(s',s) = \pi(s)A(s,s')$$

However, there are some implementation details that need to be accounted for, as calculating $v_{total}(x)$ is computationally expensive. Instead, we estimate this value using the sample mean, an unbiased estimator of the mean. So instead of getting the total views of all links on a page, we sample a constant number, calculate the mean, and then multiply by the total number of links. In other words:

$$\overline{Y} = \frac{1}{k}\sum_{i=1}^{k} v(i) \Rightarrow v_{total}(x) \approx \sum_{\text{link } l \in x} \overline{Y} = (numlinks(x)) * \overline{Y}$$

For the actual implementation details, we used the Python Wikipedia API to get all necessary info. The pseudocode is then:

Algorithm 1 Hybrid Modified MCMC Random Walk

```
procedure MCMC
    url_start ← e.g. wiki of 'Forbes_Celebrity_100'
    curr_url ← url_start
    num_samples ← 20
    univ_count ← {Cal: 0, Stanford: 0, ...}
    for i in range(2000) do
        url_list ← all URLs on page curr_url
        if random.random() $< 0.9$ AND $len(url\_list) > 100$ then
            accepted = False
            while not accepted do
                sampled ← RANDOM_CHOICE(url_list)
                accept_prob ← A(curr, sampled)
                if random.random() $< min(1, accept\_prob)$ then
                    accepted = True, curr_url = sampled
            parse through curr_url, and increment all universities seen
        else curr_url = url_start
    return univ_count
```

3 Experiments

We tweaked the following variables when running the MCM algorithm on US News Top 50 Colleges:

- iterNum(integer): We set this by default to 2000. This took around 10 minutes on the Hive machines in Soda. Any less and the number of college hits would be insufficient.
- numSamples(integer): Default: 20. This is the number of links we would sample on a page. We took the sample mean of the views of these links, and then used this to estimate the total number of views on a page. The default value of 20 was a fair trade off between performance and accuracy. It would also be an option to use a percentage, as opposed to a constant number.
- restartRate(float): Default 0.9. This is the bias of the "coin flip" to decide whether or not to restart at a new page. The default value of 0.9 performed well, as an expected 10 pages of walking is ample.
- acceptanceFunc(boolean): The truth value of this variable determines the acceptance function used. The two options are described earlier in the paper. It turns out that the difference between the two acceptance functions did not matter too much, as the difference between uniform and non uniform priors rarely pushed the probability below 1. The proposal density function, however, is the same in both acceptance functions.

3.1 Consider links of celebrities only

We believe the quality of universities is best embodied by their alumni and thus we wish to rank colleges based on the impact of people the universities have educated. Therefore, in the first experiment we considered only pages of humans/celebrities, i.e. excluding pages of organizations, nouns and other entities.

In this experiment, we don't have a acceptance function, every link that links to a human is chosen uniformly at random and is visited once chosen.

The transition rule is as follows:
1. When on a page, order the links on the page randomly.
2. Go through the list.
3. If a page of a human is encountered before going through min(100, links on page) links, stop and visit that link. Otherwise, return to the starting page.
4. Record mentions of universities on the page.
5. Repeat.

The parameters were: iterNum = 2000, restartRate = 0.97.

3.2 Varying Acceptance Function

We experimented with two different assumptions for acceptance function (as explained in the theory and pseudocode section) – option one assuming uniform stationary distribution of all pages, option two assuming the stationary distribution of pages is proportional to the views of pages. Both assumptions produce reasonable result, although different from the other experiments, as included in graphs below.

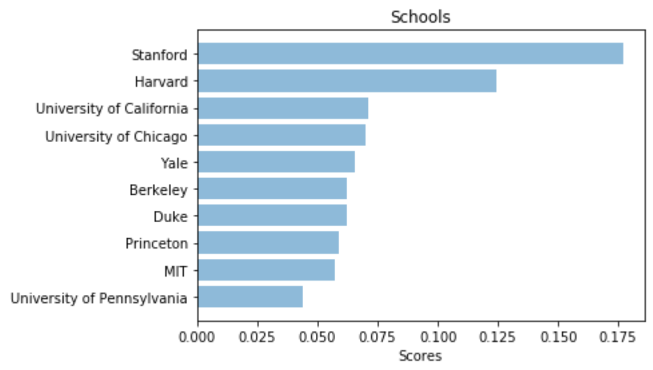
3.3 Varying Starting page

For both experiments described above, we varied starting page in order to eliminate and average out the bias caused by using a specific starting page. The starting pages used include: Wikipedia page of Elon Musk, Wikipedia page of Forbes Celebrity 100 and Wikipedia page of List of covers of Time magazine in 2010s.

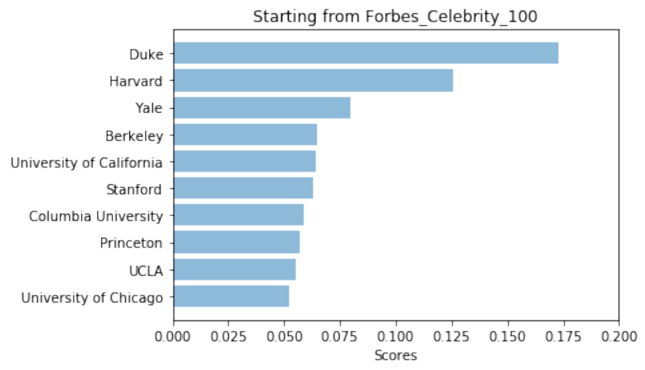
4 Results and Analysis (With Figures)

4.1 Consider links of celebrities only

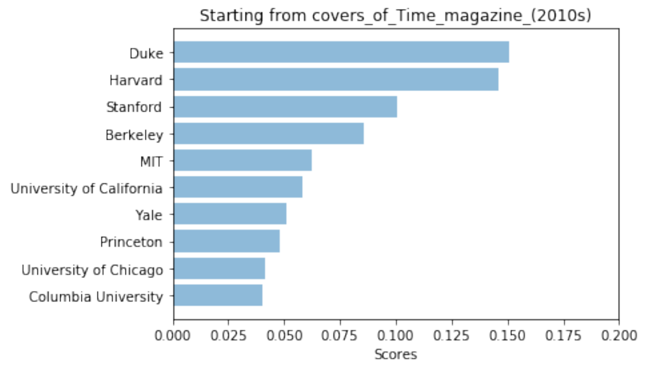
- Running from Elon Musk



- Running from Forbes 100

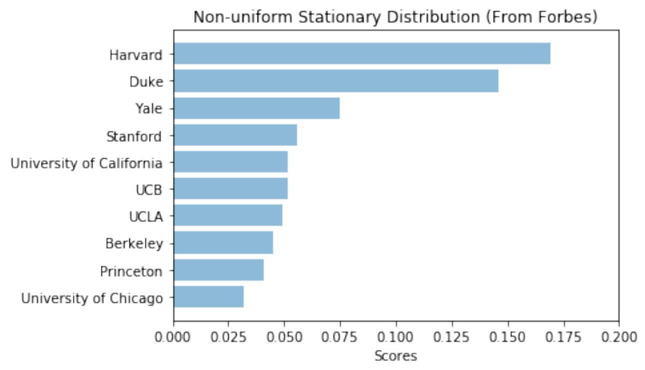


- Running from covers of Times Magazine

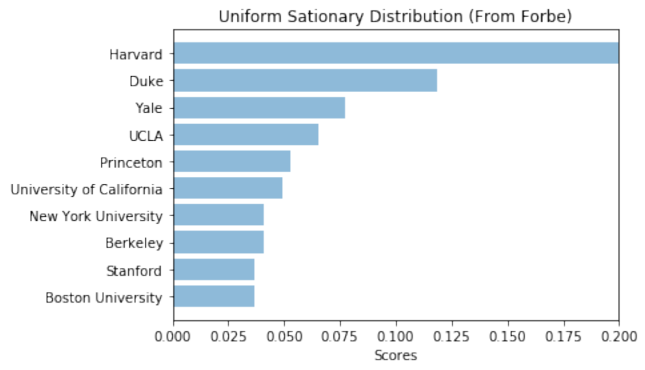


4.2 Varying acceptance function

- Uniform Stationary Distribution Assumption



- Non-Uniform Stationary Distribution Assumption



4.3 Analysis

With different assumptions on acceptance function and on the algorithm and running 2000 iterations, while the rank shuffles, the top 10 universities are constantly in the following list:
Stanford, Harvard, Berkeley, Duke, Yale, Columbia, MIT, UCLA, Princeton, University of Chicago, New York University, University of Pennsylvania
which shows the consistency of our ranking method.

5 Discussion w/ Limitations

- Handling pathologies
  - There may be cases of infinite loops that stumble the algorithm
  - We solve this by randomly restarting at an arbitrary page, which artificially adds more links in our graph model.
  - When we randomly restart, we choose a new Wikipedia page at random. Sampling from a reasonable distribution could yield better results.

- Optimizations
  - Web scraping: The main bottleneck with our function right now is the bandwidth bottleneck. An ideal system would be to scrape Wikipedia to completely define the graph structure, and then perform the algorithm. All the work would be done up front by the web scraping.
  - Caching: Wikipedia has around 5.6 million articles, and roughly 1 million of those articles are those of people. Scraping Wikipedia and then caching the views per page as well as the link structure should not take more than a few days.
  - The two above improvements would greatly improve the speed of our implementation.
- Limitations of Our Assumption

  We assumed that the ranking of a university should be proportional to the mentions it got on Wikipedia pages and the number of celebrity alumni it has. However, the two quantities mentioned above may also be influenced by the history and the size of the university.

6 Git Gist Link to Code

1. MCMC with acceptence function: https://gist.github.com/yiranjia/779da7b25c4ded7b0e75938b1396d482
2. MCMC human only: https://gist.github.com/yiranjia/b1563a6875a82dd6032dbb59fdafbbd9