

# Selected Topics in Frontiers of Statistics First Assignment

12111620 Yixuan Ding

March 19, 2024

## Graph Generation

As the question mentioned: First generate a undirected graph with 100 nodes, the smallest degree of each node is 2, the largest is 20. Also, the degree distribution of the graph is:  $p(k) = k^{-2.5}$ . After that, randomly select edges and turn them into directed edges, making the graph being directed.

As for code implementation, first define a function named *generate\_degree\_sequence* to generate a sequence of nodes whose degrees according with demand, then also define a function named *generate\_directed\_graph* for final graph generation.

The network visualization is shown as:

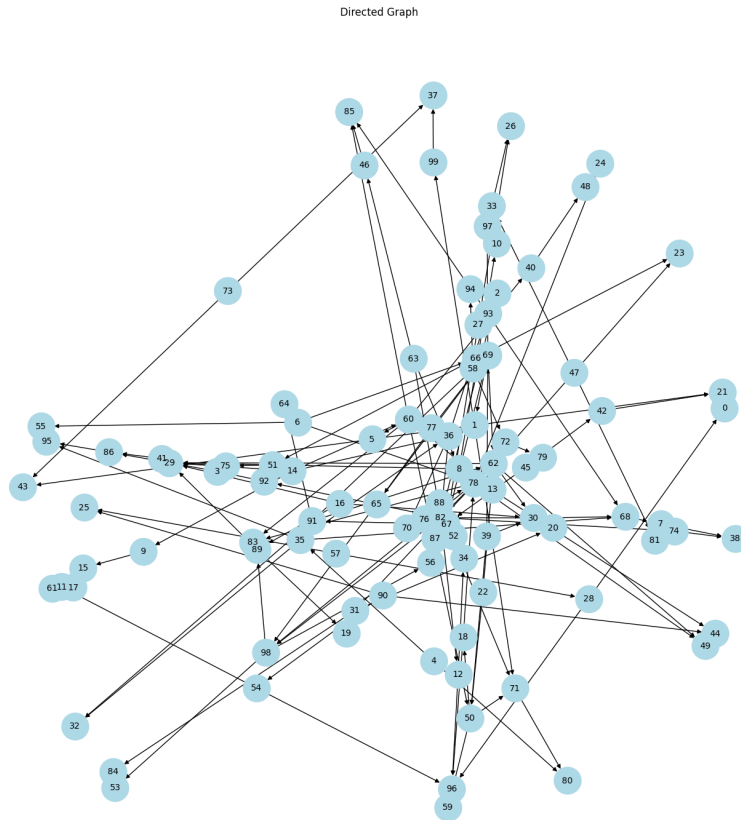


Figure 1: Network Visualization

## PageRank Algorithm Implementation

For the first problem, PageRank algorithm is implemented into graph generated before, with  $\beta = 0.85$ . Below is the implemented detail, which also define in the code script *calculate\_pagerank*:

---

**Algorithm 1** PageRank

---

```
N ← Number of nodes in graph
Initialize the pagerank of each node  $\frac{1}{N}$ 
for __ in range(max_iterations) do
  for node in graph.nodes() do
    sum_pr ← 0
    if graph has neighbor then
      for node in graph's neighbors do
        if node's neighbor is null then
          Continue
        end if
        sum_pr += node's pagerank / number of node's neighbors
      end for
    else
      sum_pc ← 0
    end if
    new pagerank of node ←  $(1-\beta)/N + \beta * \text{sum\_pr}$ 
  end for
  if Converge then
    break
  end if
  pagerank ← new_pagerank
end for
```

---

```

PageRank值:
Node 14: 0.017040120920571705
Node 82: 0.015293414665075418
Node 60: 0.013084040281199227
Node 54: 0.011274999999999999
Node 62: 0.01047899255775792
Node 9: 0.01
Node 15: 0.01
Node 11: 0.01
Node 61: 0.01
Node 17: 0.01
Node 36: 0.009978285882848575
Node 98: 0.009900929695972635
Node 91: 0.009778030660956217
Node 16: 0.009491605120881976
Node 78: 0.009310392994993292
Node 58: 0.009155200212174825
Node 88: 0.007716946717129669
Node 39: 0.007689995035863737
Node 63: 0.007397600034886073
Node 69: 0.0073616124689449905
Node 8: 0.007138687500000002
Node 6: 0.007117853916042166

```

Figure 2: Part of Pagerank Outcome

Conduct the algorithm, then we can get the pagerank for each node, after that, discover the relationship of pagerank and degrees for nodes, which has a correlation coefficient of 0.503. The relationship is shown by the figure below:

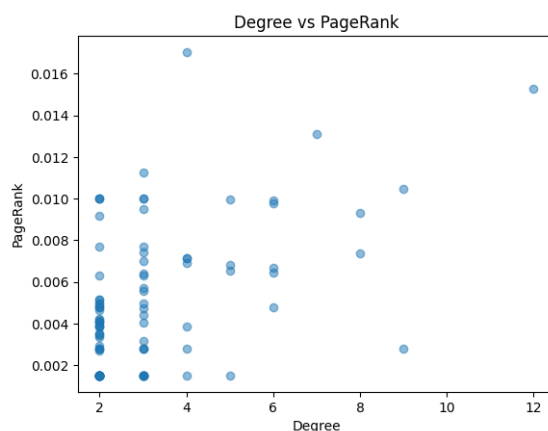


Figure 3: Relationship of Pagerank and Degree of nodes

## Spam Farm Design

Based on the graph generated before, design a spam farm to simulate an attack to the origin network to boosting the pagerank of the target page. The structure of spamfarm consists of three aspects:

- Accessible Nodes: Composed by the original node of directed graph, which represents the common webpage(harmless)
- Target Page: Page that we want to boost its pagerank by means of adding spam page
- Spam Pages: Harmful pages in the network, aiming for boosting pagerank of target page; need to be detected and removed.

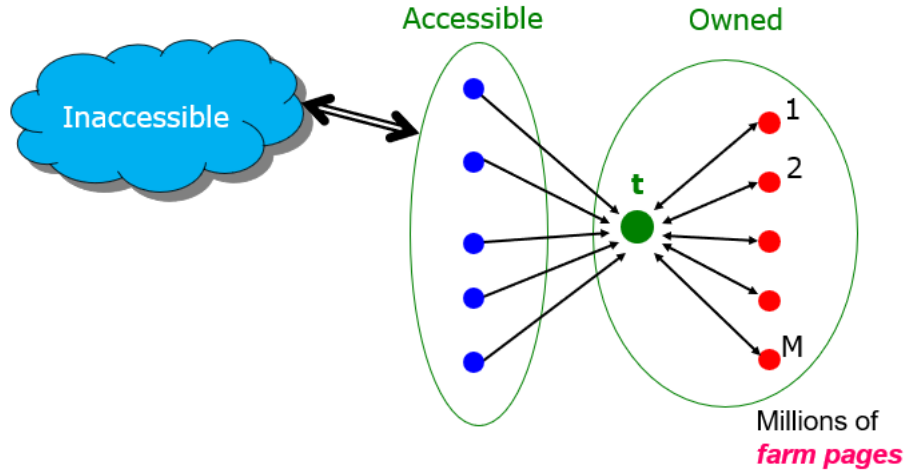


Figure 4: Design of Spamfarm

To discover the relationship of target page's pagerank and the number of spam pages, a figure is shown below:

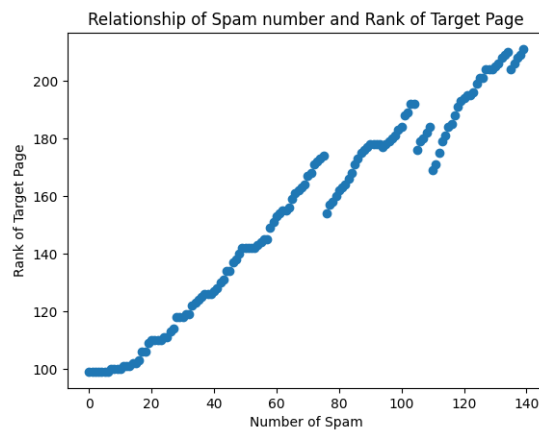


Figure 5: Relationship of Spam Number and Pagerank of Target Page

It is easy to find that as the number of spam pages growing, the pagerank of target page will become higher.

## TrustRank Algorithm Implementation

In order to solve the spam farm situation, trustrank algorithm is introduced and implemented in the following report. Randomly, select 5% percent of the accessible node to be trust node, according to which trustrank of each node is calculated and analyzed.

---

**Algorithm 2** TrustRank

---

```
seedsizesize  $\leftarrow$  5% amount of number of accessible nodes
Trustvalue  $\leftarrow \frac{1}{seedsizesize}$ 
Initialize the trustrank of each node  $\triangleright$  Randomly Select trust nodes from accessible nodes
for  $\_$  in range(max\_iterations) do
  for node in graph.nodes() do
     $sum\_pr \leftarrow 0$ 
    if graph has neighbor then
      for node in graph's neighbors do
        if node's neighbor is null then
          Continue
        end if
       $sum\_pr += \text{node's trustrank} / \text{number of node's neighbors}$ 
    end for
  else
     $sum\_pc \leftarrow 0$ 
  end if
  if node is trust node then
    new trustrank of node  $\leftarrow (1 - \beta) * trust\_value + \beta * sum\_pr$ 
  else
    new trustrank of node  $\leftarrow \beta * sum\_pr$ 
  end if
end for
if Converge then
  break
end if
trustrank  $\leftarrow$  new\_trustrank
end for
```

---

```

TrustRank值:
Node 65: 0.03000283537852438
Node 13: 0.030000034166055095
Node 92: 0.030000000000000006
Node 7: 0.030000000000000006
Node 37: 0.030000000000000006
Node 82: 0.0036454806979262407
Node 60: 0.0031875000000000007
Node 68: 0.00255
Node 99: 0.001961538461538462
Node 73: 0.001961538461538462
Node 77: 0.0017030196726105925
Node 30: 0.0004923310329373932
Node 35: 0.00029418899565916863
Node 93: 0.0002709375
Node 62: 0.00026796421911261204
Node 36: 0.0002454219568850973
Node 63: 0.00022139353293739323
Node 52: 0.00022139353293739323

```

Figure 6: Part of Trustrank Outcome

## Spam Mass Calculation

By utilizing the trust page, pagerank of the spammed graph is calculated, combined with the trustrank, spammass can be calculated:

$$SpamMass = \frac{P - T}{P}$$

where  $P$  denotes pagerank, and  $T$  denotes trustrank.

Part of Spam Mass outcome is shown below: It is obvious that spam page has heavier spam mass, which means they are more likely to be spam pages.

```
Spam Mass值:
Node Spam_Page_9: 0.050035274317276855
Node Spam_Page_4: 0.04939032613777562
Node Spam_Page_18: 0.04824313366851217
Node Spam_Page_15: 0.04802618644382297
Node Spam_Page_10: 0.047220430809542316
Node Spam_Page_1: 0.04703249243436923
Node Spam_Page_7: 0.046757097013555377
Node Spam_Page_8: 0.045401511339764
Node Spam_Page_2: 0.04384253581982321
Node Spam_Page_17: 0.04381203497715825
Node Spam_Page_6: 0.04337789862038294
Node Spam_Page_13: 0.04330137148536399
Node Spam_Page_22: 0.042536346744347746
Node Spam_Page_12: 0.04210268898299283
Node Spam_Page_14: 0.04116548573034595
Node Spam_Page_21: 0.040326263951723444
Node Spam_Page_11: 0.04012997742716398
Node Spam_Page_25: 0.040064639560263855
Node Spam_Page_23: 0.03981916131830035
Node Spam_Page_26: 0.0381791245111447
Node Spam_Page_3: 0.037968103776940675
Node Spam_Page_20: 0.03763461097285774
```

Figure 7: Part of Spam Mass Outcome

## Appendix

More relevant detailed information you can find in: [Github Repository](#)