COEN 242
“PAGE-RANK”

Team members

Drashty Majmudar

Mitali Sahoo

Vyoma Shah

Poonam Kanani

Guided By: Prof. Zhiqiang Tao

Scope of this presentation…

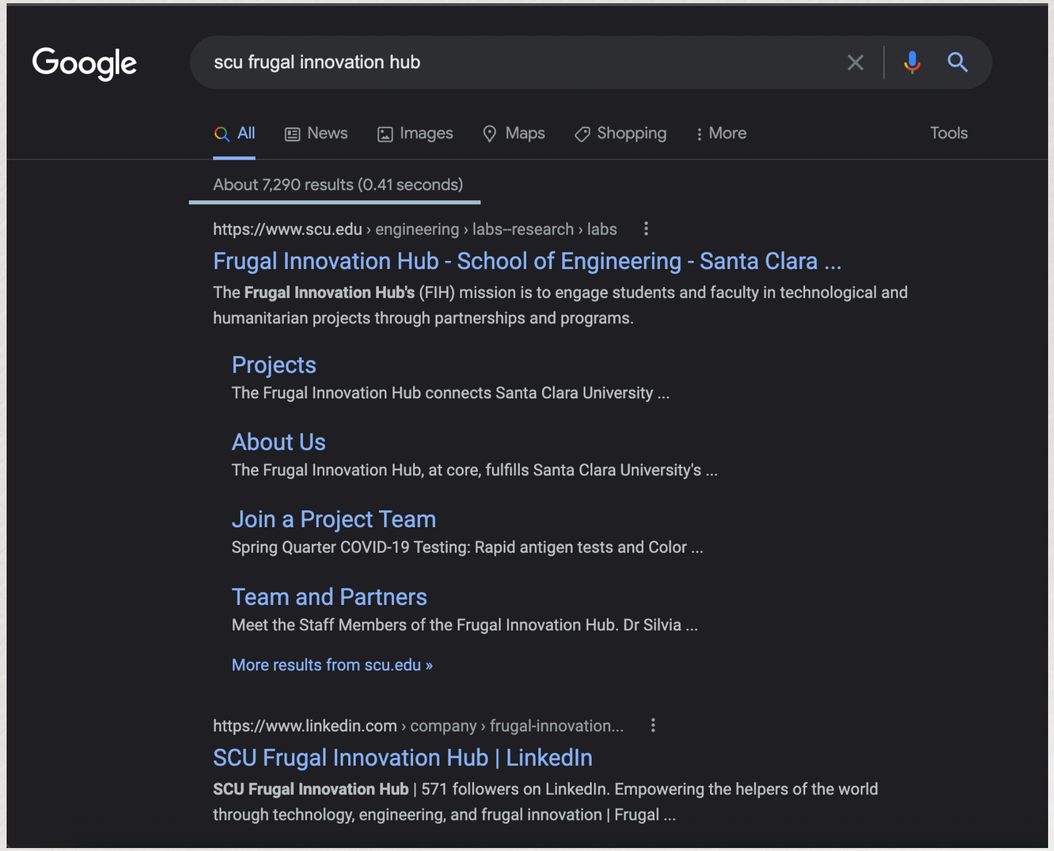
- Motivation
- Introduction of PageRank
- Problem Dataset
- Implementation of PageRank algorithms
    1. Naive Implementation
    2. Power Iteration Method
- Analysis of these PageRank algorithms
- Comparison between with and without partition
- Experiments
- Current progress and what to do next…

Motivation

- This is an era of internet and technology. We use various search engines to search for data. Oftentimes, we use Google.
- Getting curious, we pondered how this actually works in back-end and how it is implemented at base-level.
- We found that Google uses PageRank (PR) algorithm to rank the web pages which is searched. Getting inspired from this, we thought of implementing simmering alike using Berkeley-Stanford web dataset.

What is Page Rank ?

- PageRank (PR) is an algorithm is used to rank the web pages used by Google search engine thus resulting into appropriate output.
- The websites having more similar and appropriate will be appeared on top following by other websites containing that information.
- It is an iterative algorithm and it works by counting number and quality of links which is related to the search which helps to determine the importance of website.
- Here attached is the google search of “scu frugal innovation hub” which provides around 7k results within a fraction of seconds. Isn’t this amazing!!!!!!!



Problem Dataset

- Dataset : Berkeley-Stanford Web Graph
- Dataset-resource : https://snap.stanford.edu/data/web-BerkStan.html
- This dataset consists of two columns:
  A. One column contains source node #id
  B. Second column represents destination #id

Dataset Analysis

Number of edges	20000
Number of Nodes	3821
Number of nodes with no outgoing edge	1862
Number of nodes with outgoing edges	1959
Average outgoing edges per node	11
Node having highest outgoing edges	254913
Number of outgoing edges of node 254913	96
Node that has largest incoming edges	401873
Number of incoming edges to node 438238	855
Number of nodes with incoming edges	3806
Number of nodes without incoming edges	15

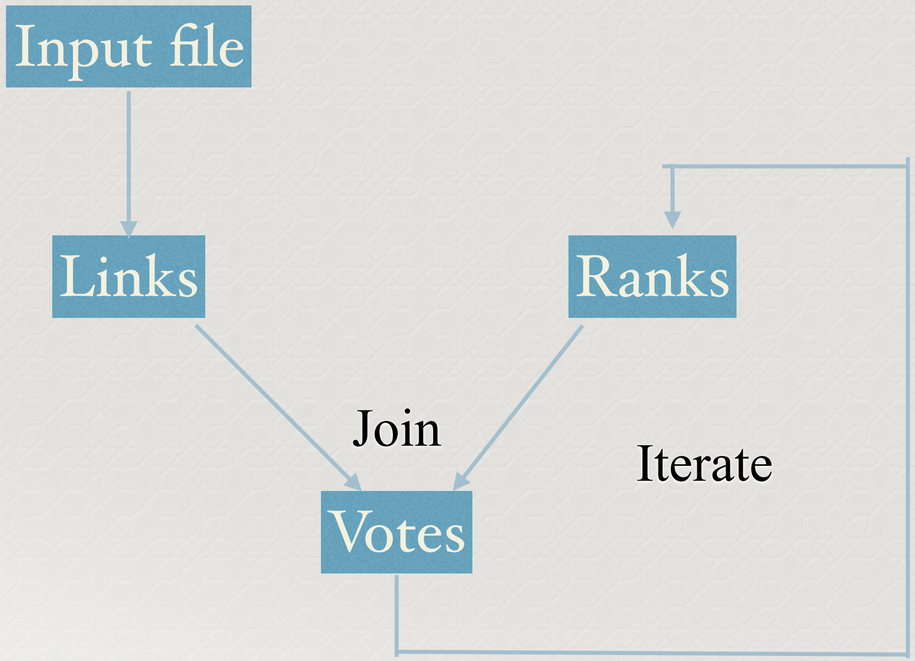
Naive Implementation

- Page rank naive method algorithm is as follows:
- Step-1: Read input file
- Step-2: initialize rank of each page as 1
- Step-3: For i=1 to number of iterations,

    1. calculate the contribution of each node as: $\sum_{i \to j} \frac{r_i}{d_i}$
    2. Set rank of each page as: rank=0.15 + 0.85 * contribution

Flowchart

- Below attached figure is the flowchart of naive implementation:



Naive Implementation - Code

- Our dataset looks like:

```
# FromNodeId        ToNodeId
1       2
1       5
1       7
1       8
```

- Extracting each row into “rows” RDD:

```
>>> rows=spark.read.text('ds1.txt').rdd.map(lambda x:x[0])
>>> rows.collect()
[u'src\tdest', u'1\t2', u'2\t3', u'2\t4', u'3\t4']
```

- “links” RDD is a collection of key-value pairs, where key is a node and values are the outgoing links for the key

```
>>> links=rows.map(lambda nodes:splitFunction(nodes)).distinct().groupByKey().cache()
>>> links.collect()
[(u'1', <pyspark.resultiterable.ResultIterable object at 0x7f6069979510>), (u'src', <
pyspark.resultiterable.ResultIterable object at 0x7f60699794d0>), (u'3', <pyspark.res
ultiterable.ResultIterable object at 0x7f60699809d0>), (u'2', <pyspark.resultiterable
.ResultIterable object at 0x7f6069980a10>)]
```

- Setting ranks of each node represented as key in “links” RDD to 1

```
>>> ranks=links.map(lambda x:(x[0],1.0))
>>> ranks.collect()
[(u'1', 1.0), (u'src', 1.0), (u'3', 1.0), (u'2', 1.0)]
```

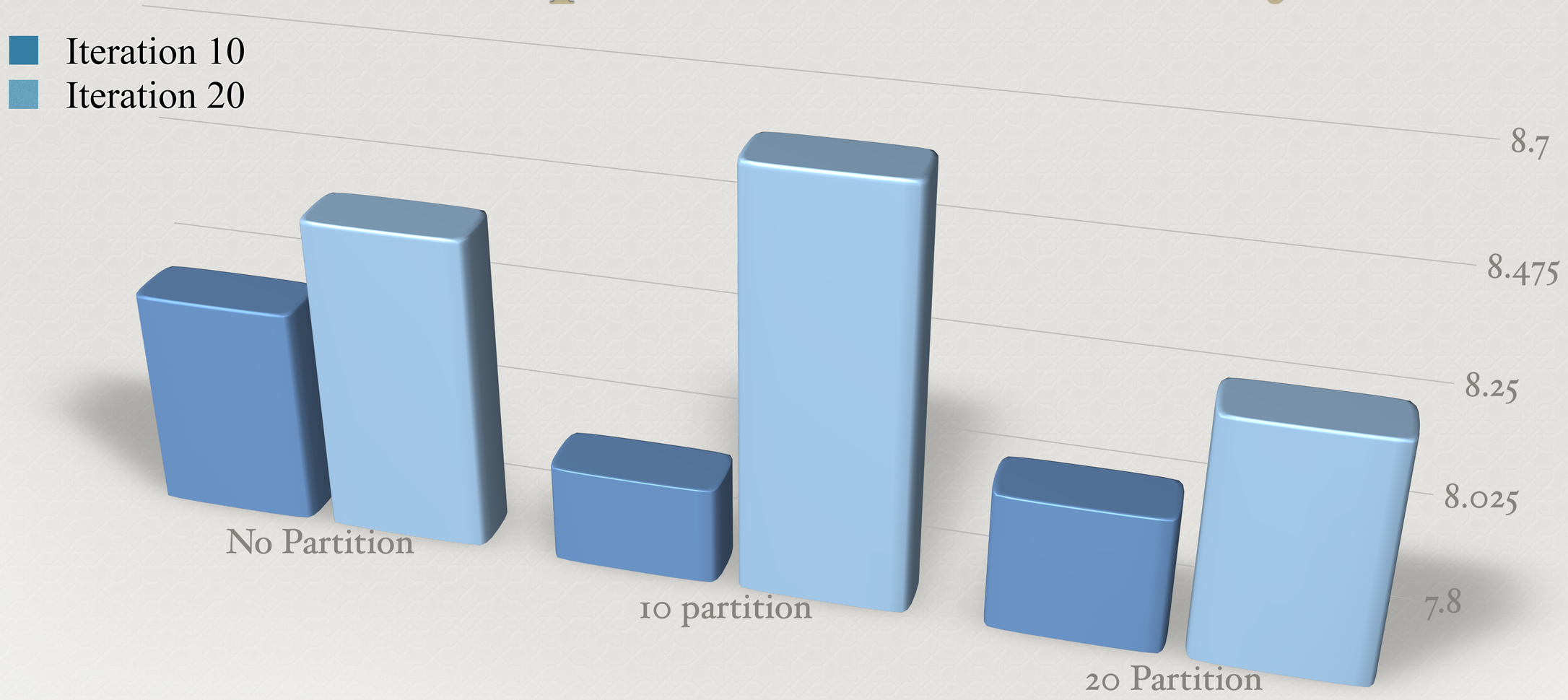
- Update the ranks for i=1 to i=number of iterations:
  - Calculate votes to rank of other nodes
  - Recalculate ranks based on neighbors’ votes

```
>>> for i in range(no_of_iterations):
...     votes=links.join(ranks).flatMap(lambda y:calculateVotes(y[1][0],y[1][1]))
...     ranks=votes.reduceByKey(add).mapValues(lambda vote:vote*0.85+0.15)
```

- Print rank of each page:

```
>>> for (node,rank) in ranks.collect():
...     print("%s has rank: %s" % (node,rank))
```

Naive Implementation Analysis



Power Iteration Method

- Data Preparation:
  - Load the input dataset file into RDD

```
>>> lines = spark.read.text('web-BesrStan.txt').rdd.map(lambda r: r[0])
```

  - Load all the nodes from the input file and initialize their neighbours.

```
>>> links = lines.map(lambda urls: parseNeighbors(urls)).distinct().groupByKey().partitionBy(partitions).cache()
```

  - Load count of nodes into N and Initialize neighbours for each node and assign 1/(# of Nodes) rank to each node.

```
>>> N = links.count()
```

```
>>> ranks = links.map(lambda node: (node[0],1.0/N)).partitionBy(partitions)
```

- For each node i, calculate the page rank (node) / no. of inbound link to node N for each out-link of ‘i’ and assign it to nodes with map method
- ReduceByKey function is used for each node, sum the votes and update the r_i value.

```
>>> for iteration in range(iterations) :
...         ranks = links.join(ranks).flatMap(lambda x : [(i, float(x[1][1])/len(x[1][0])) for i in x[1][0]])\
...           .reduceByKey(lambda x,y: x+y).partitionBy(partitions)
... sorted_ranks = ranks.top(5, key=lambda x: x[1])
```

Experiments - Naive

```
dmajmudar@linux10616:~

 RUNTIME FOR 1 PARTITIONS AND 10 ITERATIONS = 7.89861798286
450783 has rank: 0.152598332464.
643 has rank: 0.285774534046.
450786 has rank: 0.388575846861.
173029 has rank: 0.161694531912.
173028 has rank: 0.161694531912.
592 has rank: 0.28193939602.
405714 has rank: 0.184095785593.
405719 has rank: 0.158872602404.
405718 has rank: 0.184095785593.
173025 has rank: 0.161694531912.
```

```
dmajmudar@linux10616:~

 RUNTIME FOR 1 PARTITIONS AND 20 ITERATIONS = 8.86169099808
450783 has rank: 0.152578551377.
643 has rank: 0.2834679075.
450786 has rank: 0.380195603942.
173029 has rank: 0.161395140863.
173028 has rank: 0.161395140863.
592 has rank: 0.279730617564.
405714 has rank: 0.183695395265.
405719 has rank: 0.158818092362.
405718 has rank: 0.183695395265.
173025 has rank: 0.161395140863.
```

Experiments - Power Iteration

msahoo@linux10601:~/tmp

```
<bound method PipelinedRDD.getNumPartitions of PythonRDD[88] at RDD at PythonRDD.scala:53>
Power Iteration with 1 partitions and 10 iterations 14.3713560104 secs
1 has rank: 1.83150114721e-05.
10 has rank: 5.07606955332e-06.
100 has rank: 0.000555314663736.
1000 has rank: 4.18909024106e-10.
1001 has rank: 4.18909024106e-10.
100130 has rank: 2.23864381546e-08.
1002 has rank: 4.18909024106e-10.
1003 has rank: 1.73791587053e-09.
1004 has rank: 1.73791587053e-09.
1005 has rank: 2.2122457586e-10.
1006 has rank: 2.2122457586e-10.
100646 has rank: 2.23864381546e-08.
1007 has rank: 2.2122457586e-10.
1008 has rank: 2.2122457586e-10.
1009 has rank: 2.2122457586e-10.
101 has rank: 0.000617122096932.
1010 has rank: 2.2122457586e-10.
1011 has rank: 2.2122457586e-10.
101163 has rank: 2.23864381546e-08.
1012 has rank: 2.2122457586e-10.
1013 has rank: 2.2122457586e-10.
1014 has rank: 2.2122457586e-10.
1015 has rank: 2.2122457586e-10.
1016 has rank: 2.2122457586e-10.
1017 has rank: 2.2122457586e-10.
1018 has rank: 2.2122457586e-10.
1019 has rank: 1.74866596314e-09.
102 has rank: 0.000554163163572.
1020 has rank: 1.98704419452e-10.
```

msahoo@linux10601:~/tmp

```
<bound method PipelinedRDD.getNumPartitions of PythonRDD[91] at RDD at PythonRDD.scala:53>
Power Iteration with 10 partitions and 10 iterations 15.2579669952 secs
1 has rank: 1.83150114721e-05.
10 has rank: 5.07606955332e-06.
100 has rank: 0.000555314663736.
1000 has rank: 4.18909024106e-10.
1001 has rank: 4.18909024106e-10.
100130 has rank: 2.23864381546e-08.
1002 has rank: 4.18909024106e-10.
1003 has rank: 1.73791587053e-09.
1004 has rank: 1.73791587053e-09.
1005 has rank: 2.2122457586e-10.
1006 has rank: 2.2122457586e-10.
100646 has rank: 2.23864381546e-08.
1007 has rank: 2.2122457586e-10.
1008 has rank: 2.2122457586e-10.
1009 has rank: 2.2122457586e-10.
101 has rank: 0.000617122096932.
1010 has rank: 2.2122457586e-10.
1011 has rank: 2.2122457586e-10.
101163 has rank: 2.23864381546e-08.
1012 has rank: 2.2122457586e-10.
1013 has rank: 2.2122457586e-10.
1014 has rank: 2.2122457586e-10.
1015 has rank: 2.2122457586e-10.
1016 has rank: 2.2122457586e-10.
1017 has rank: 2.2122457586e-10.
1018 has rank: 2.2122457586e-10.
1019 has rank: 1.74866596314e-09.
102 has rank: 0.000554163163572.
1020 has rank: 1.98704419452e-10.
1021 has rank: 2.03937902135e-10.
1022 has rank: 2.03937902135e-10.
1023 has rank: 2.03937902135e-10.
1024 has rank: 2.03937902135e-10.
```

What to do next…

Project report

Node rank analysis

Code refactoring

Any questions or suggestions?