Project 2: Fast Convergence PageRank in Hadoop

Geet Manish Varma – GMV33
Joseph Kevin Bernard – JKB243
Sudharsan Coimbatore Premkumar – SC2466

Contents of solution.zip:

Source Folder – Unblocked and Blocked solution in java; Parser in python

Output files:

1. PageRank_Blocked_1 – Page rank values of highest number node in each block with initial value of page rank taken as $(1-d)/N$
2. PageRank_Blocked_2 – Page rank values of highest number node in each block with initial value of page rank taken as $1/N$
3. residual_random_blocked – residual values of all the passes till convergence for the run with random block to node mapping.
4. Project2_presentation – Presentation for demo.

Solution design:

1. **Input parsing** – Done in python. This is included in the source folder.

- Input Provided:

$$U\ V\ random$$

Where, $\forall\ U, V\ \in G,\ there\ exists\ U \to V$

Use the random and a range generated from the netid to obtain a subset of G for page rank calculation.

- Filter Parameters :
Netid used – SCP2466
rejectMin- 0.657558
rejectMax- 0.667558
Number of edges selected- 7524423

- Process the input provided to get file with tuples in the following format

$$U\ PR(U)\ deg(U)\ < list\ of\ V's >$$

Where, $\forall\ V\ in\ list,\ theres\ exists\ U \to V\ in\ G$

And $PR(U) = \frac{1-d}{N},\ where\ d - damping\ factor\ and\ N\ - number\ of\ nodes$

2. **Unblocked implementation**

a. Mapper

For every line in the input – emit

1. The line itself with the source node as key:

$$< U; PR(U)\ \deg(U)\ \{List\ of\ V's\} >$$

Where, there exists an $U -> V$ in G

2. A tuple for V from the list of the form

$$< V; \frac{PR(U)}{\deg(U)} >$$

Data structure used - Strings

b. Reducer

For all tuples of the form: $< V; \frac{PR(U)}{\deg(U)} >$ - find the summation of $\frac{PR(U)}{\deg(U)}$

Find the residual and update the node's page rank in the tuple:

$$< U; PR(U)\deg(U)\ \{List\ of\ V's\} >$$

Store the residual in a global counter.

Data structure used - Strings

c. Global Convergence

The reducer at each pass end stores the residual for the node in counter. This counter value is divided by the total number of nodes to get the global residual value which is compared against the threshold – 0.1%.

3. Blocked implementation

a. Mapper

For every line in the input – emit

1. The line itself with the block of the source node as the key

$$< blockID(U); U\ PR(U)\deg(U)\ \{list\ of\ V's\} >$$

Where, there exists a U->V in G

2. A tuple for every edge in/leaving the block of the form

$$< blockID(V); U\ V >$$

3. A tuple for every edge entering the block of the form

$$< blockID(V); U\ V\ \frac{PR(U)}{\deg(U)} >$$

Data structure used – Strings

b. Reducer

Inputs :

$$<; V, PR(V), deg(V), [U1, U2, \dots]\ >$$

$< U, V >$ - BE edges

$< U, V, R >$ - BC edges

Emit:

$$< V;\ PR(V), deg(V), [U1, U2, \dots.]\ >$$

Data structures used – Hashmaps for the following purposes

Input key value map

- $\{V : V,\ PR(V), \deg(V),\ [U1,\ U2, \ldots]\}$

Page rank at beginning of pass

- $\{V\!: PR(V)\}$

BE edges Map

- $\{V : [U1,\ U2, \ldots]\}$

BC edges Map

- $\{V : [R1,\ R2, \ldots]\}$

c. Pseudocode

```
• Store initial PR, $IPR[v]\ \forall\ v \in B$
• Till block convergence - $itrResidual < threshold$
    – IterateBlockOnce
    – $itrResidual\ = \sum residual_i[v]\ \forall\ v \in B$, where
          $residual_i[v] = \frac{|PR_{i-1}[v] - NPR_i[v]|}{NPR_i[v]}$
    – Compute $blockResidual\ = \sum residual_b[v]\ \forall\ v \in B$, where
          $residual_b[v] = \frac{|IPR\ [v] - NPR_i[v]|}{NPR_i[v]}$
    – Set the ‘residual’ hadoop counter to $blockResidual$*10000
```

d. Special cases – Handling sink nodes in a block

In the first pass of the reducer run we check for all the nodes that are present in the BE edges map and not in the Input key value map, we create a dummy tuple of the format

$$\{V : V,\ PR(V), 0,\ -1\}$$

This tuple will hold the updated page rank for that node for all the consecutive passes.

Extra Credits:

a. Random Block partition:

Used the same block partition solution as mentioned above but changed the node to block mapping using the following equation:

$$\boldsymbol{long\ blockIDofNode(long\ nodeID)}\ \{\ nodeID\ \%\ Random;\ \}$$

$$\boldsymbol{Where\ Random\ was\ set\ to\ 100}$$

Results

The values varied depending on whether we used 1/N or (1-d)/N as the initial value of PageRank for all the nodes in the graph

1. ***Unblocked implementation: - for 7 passes***

(With 1/N as the initial value of PageRank for the nodes)

```
residual-->2.4205778542647787
pass-->1

residual-->0.3330686887157432
pass-->2

residual-->0.20286855284875568
pass-->3

residual-->0.10280902895236484
pass-->4

residual-->0.07144613406612972
pass-->5

residual-->0.039825211279732764
pass-->6

residual-->0.03316215819820819
pass-->7
```

(with (1-d)/N as the initial value of PageRank for the nodes)

```
residual-->0.23156288452473553
pass-->1

residual-->0.138824005843302
pass-->2

residual-->0.08439533294708776
pass-->3

residual-->0.0639228825691849
pass-->4

residual-->0.04230006202306091
pass-->5

residual-->0.03475685296448341
pass-->6

residual-->0.02470231134409081
pass-->7
```

2. ***Blocked implementation: - convergence seen in 8 passes***

(With 1/N as the initial value of PageRank for the nodes)

```
residual-->2.863274402214359
number of passes-->1
Average block iterations-->10.25
```

```
residual-->0.03593593626365222
number of passes-->2
Average block iterations-->5.0588235294117645

residual-->0.024630518321111416
number of passes-->3
Average block iterations-->4.602941176470588

residual-->0.01107007012527834
number of passes-->4
Average block iterations-->3.25

residual-->0.0050176378017607794
number of passes-->5
Average block iterations-->2.3088235294117645

residual-->0.002249379638372411
number of passes-->6
Average block iterations-->1.75

residual-->0.001024780637892947
number of passes-->7
Average block iterations-->1.3823529411764706

residual-->5.529214943979198E-4
number of passes-->8
Average block iterations-->1.1176470588235294
```

(with (1-d)/N as the initial value of PageRank for the nodes)

```
residual-->0.42515520809739943
number of passes-->1
Average block iterations-->12.911764705882353

residual-->0.021742197928585925
number of passes-->2
Average block iterations-->4.720588235294118

residual-->0.010695416091845829
number of passes-->3
Average block iterations-->3.6176470588235294

residual-->0.005481941016763687
number of passes-->4
Average block iterations-->2.7941176470588234

residual-->0.0028894599674668567
number of passes-->5
Average block iterations-->2.0735294117647066

residual-->0.001564003812201988
number of passes-->6
Average block iterations-->1.5441176470588236

residual-->0.0011051450019266915
number of passes-->7
Average block iterations-->1.39705882352941167

residual-->6.556125555578434E-4
```

```
number of passes-->8
Average block iterations-->1.1470588235294117
```

3. Extra credit: Blocked implementation with random:

(With 1/N as the initial value of PageRank for the nodes) – converged in 24 passes; Only 8 passes shown below

```
residual-->2.4205506919282826
number of passes-->1
Average block iterations-->3.088235294117647

residual-->0.33309388280464713
number of passes-->2
Average block iterations-->3.0

residual-->0.2025091778384254
number of passes-->3
Average block iterations-->2.9411764705882355

residual-->0.1021849382874775
number of passes-->4
Average block iterations-->2.9411764705882355

residual-->0.07088177259870497
number of passes-->5
Average block iterations-->2.9411764705882355

residual-->0.03949486430257837
number of passes-->6
Average block iterations-->2.9411764705882355

residual-->0.03285225724070221
number of passes-->7
Average block iterations-->2.9411764705882355

residual-->0.02015439964702307
number of passes-->8
Average block iterations-->2.9411764705882355
```

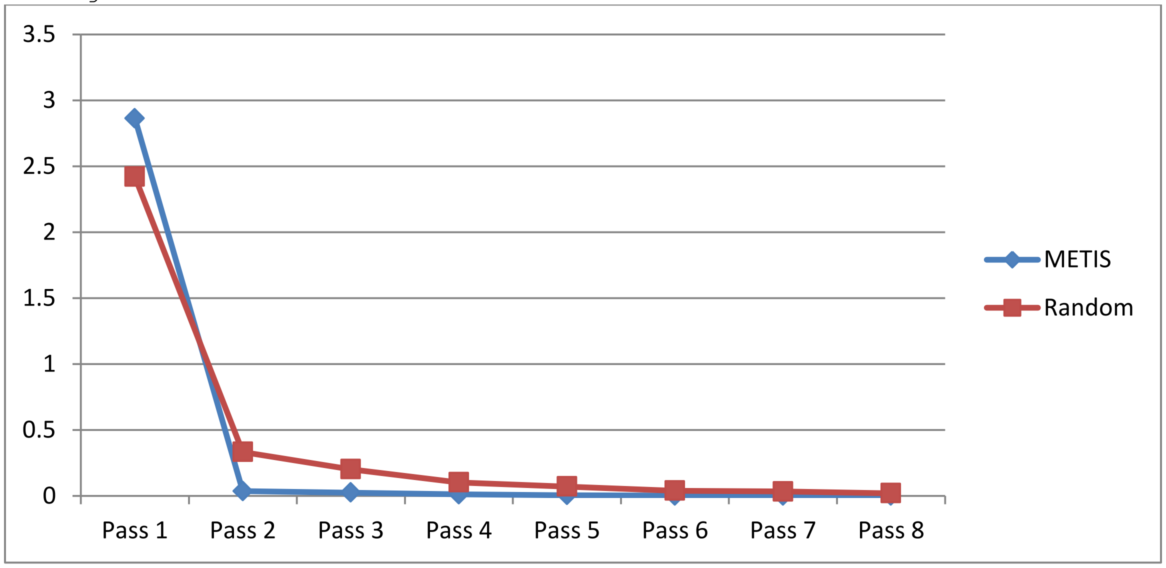

Figure 1: METIS vs Random for intial PR = 1/N

(With (1-d)/N as the initial value of PageRank for the nodes)

```
residual-->0.22678670989270436
number of passes-->1
Average block iterations-->2.9411764705882355

residual-->0.13918155344912223
number of passes-->2
Average block iterations-->2.9411764705882355

residual-->0.08468208510600995
number of passes-->3
Average block iterations-->2.9411764705882355

residual-->0.06393485837092966
number of passes-->4
Average block iterations-->2.9411764705882355

residual-->0.04230639493092405
number of passes-->5
Average block iterations-->2.9411764705882355

residual-->0.03473062396269198
number of passes-->6
Average block iterations-->2.9411764705882355

residual-->0.02470549466443816
number of passes-->7
Average block iterations-->2.9411764705882355

residual-->0.020987665530984828
number of passes-->8
Average block iterations-->2.9411764705882355
```

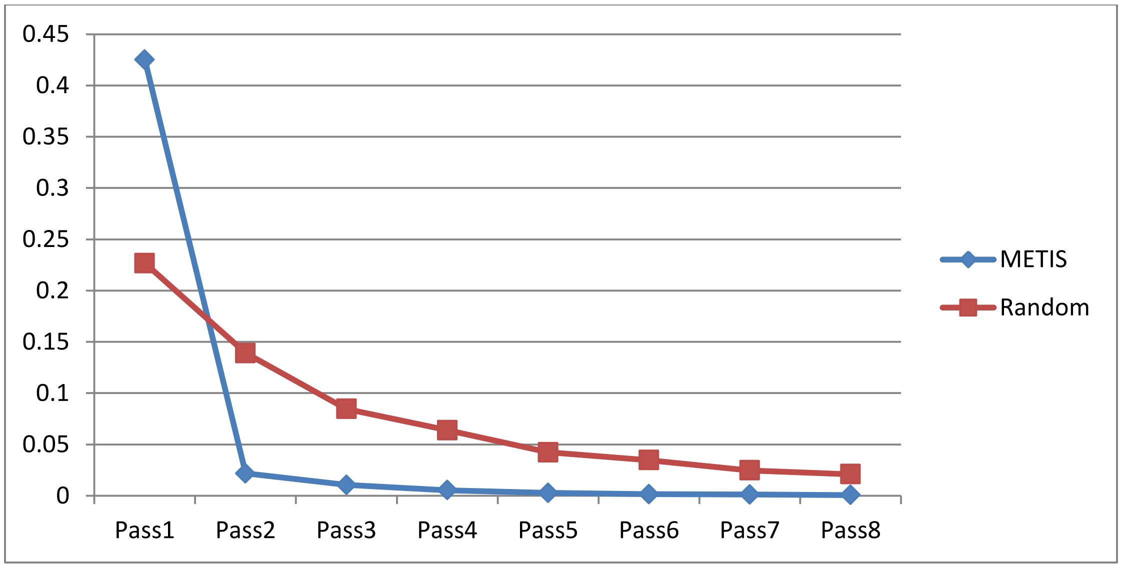

Figure 2: METIS vs Random for initial PR = (1-d)/N

Steps to run the solution

1. Parsing the input:
    a. Place parse.py with the edges.txt input in the same folder.
    b. Run parse.py to generate edges-output.txt which will be the input for the first pass of the mapper.
2. Running the MapReduce jobs
    a. Setting the arguments for the run –
        i. First argument: Package.main class (PageRankBlocked.PageRankBlocked)
        ii. Second argument: path to edges-output.txt
        iii. Third argument: path to the output folder
    b. The code runs with jre 1.0.6/1.0.7 and hadoop 1.0.4