UNIT 11 LINK ANALYSIS

Structure **Page No.**

11.0 INTRODUCTION

In the previous Units of this Block, you have gone through the concept of measuring the distances and different algorithms of handling data streams. In this section, we will discuss about the link analysis, which is used for computing the PageRank. The PageRank algorithms use graphs to represent the web and computes the Rank based on probability of moving to different links. Since, the size of web is enormous, the size of computation requires operations using very large size matrices. Therefore, the PageRank algorithm uses MapReduce programming paradigm over the distributed file system. This unit discusses several of these algorithms.

11.1 OBJECTIVES

After going through this Unit, you will be able to:

- Define link analysis
- Use graphs to perform link analysis
- Explain computation of PageRank
- Discuss different techniques for computation of PageRank
- Use MapReduce to compute PageRank

11.2 INTRODUCTION TO LINK ANALYSIS

Link analysis is a data analysis technique used in network theory to analyze the links of the web graph. Graphs consists of a set of nodes and a set of edges or connections or intersections between nodes. The graphs are used everywhere, for example, in social media networks such as Facebook, Twitter, etc.

Purpose of Link Analysis

The purpose of link analysis is to create connections in a set of data that can actually be represented as networks or the networks of information. For example, in Internet, computers or routers communicate with each other which can be represented as a dynamic network of nodes which represent the computer and routers. The edges of the network are physical links between these machines. Another example is the web which can be represented as a graph.

Representation of the World Wide Web (WWW) as a graph

WWW can be represented as a directed graph. So, in the graph, nodes will correspond to web pages. Therefore, every web page will be a node in this graph and direct links between these web pages correspond to hyperlinks. These hyperlink relationships can be used to create a network.

11.3 PAGE RANKING

PageRank is an algorithm that was designed for the initial implementation of the Google search engine. The PageRank is defined as an approach for computing the importance of the web pages in a big web graph.

PageRank – Basic Concept: A page or a node in a graph is as important as the number of links it has. Not all in-links are equal kind of links. The links coming from important pages are worth more. The importance of a page depends on the importance of other pages that points to it. Thus, the importance of a given pointed page depends on the importance of the others, and its importance then gets passed on further through the graph.

Concept of PageRank scores of a graph

Figure 1 shows a graph with a set of directed edges, where the size of a node is proportional to its PageRank scores. The PageRank scores are normalized so that they sum to 100. The page rank scores are summarized as follows:

- The node B has a very high PageRank score because it has lot of other pages pointing to it.
- The node C also has a relatively high PageRank score, even though it receives only a single incoming link. The reason behind this is that the very important node B is pointing to it.
- The very small purple colored link nodes have relatively small PageRank score, as it is pointed by a web page that has a low PageRank.
- The PageRank score of node D is higher than the PageRank of node A because D points to A.
- The node E has kind of intermediate PageRank score and E points to B and so on.

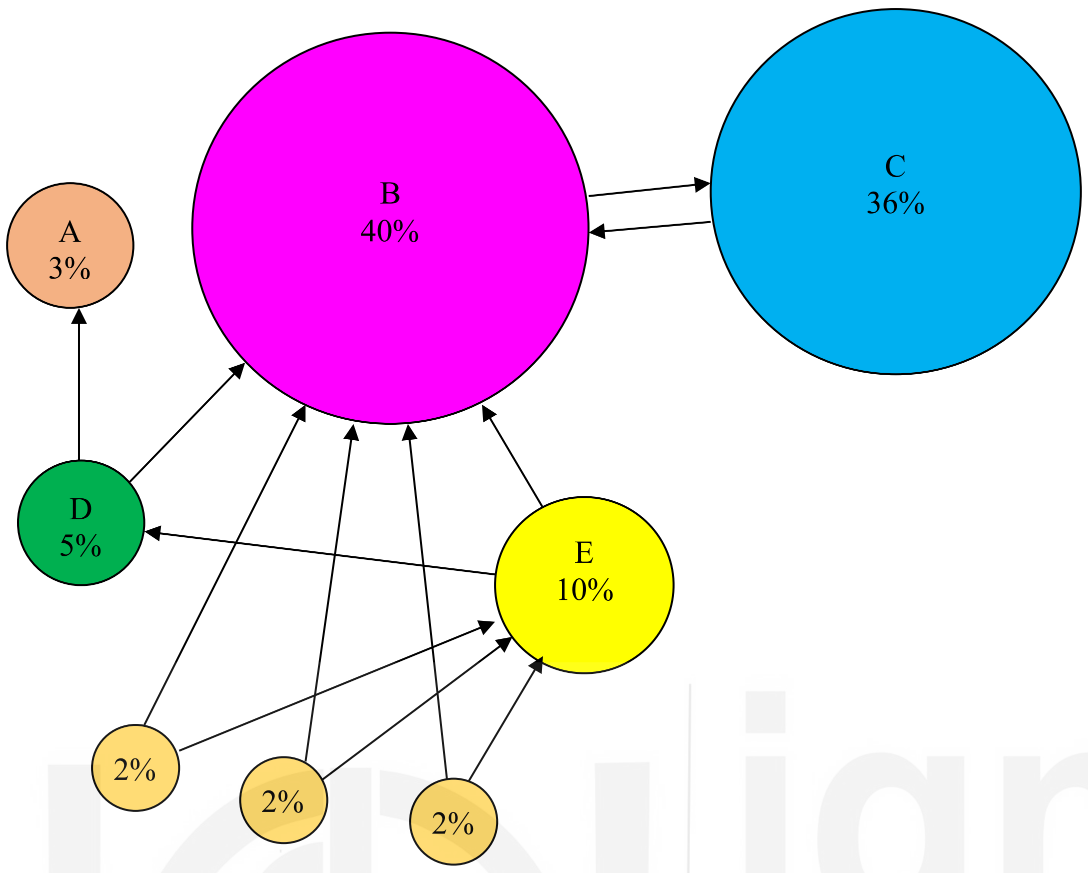


Figure 1: PageRank Scores [1]

The PageRank, as shown in Figure 1, are kind of intuitive, and they correspond to our intuitive notion of how important a node in our graph is. However, how the PageRank can be computed? Next sections answer this question.

11.4 DIFFERENT MECHANISMS OF FINDING PAGERANK

In the last section, we discuss about the basic concepts that can be used to compute the PageRank, without giving details on how actually you can compute the PageRank. This section provides details on the PageRank computation process.

11.4.1 Different Mechanisms of Finding PageRank

In this section, we discuss two important techniques that are employed to compute the PageRank, viz. Simple recursive formulation and the flow model.

The *Simple Recursive Formulation* technique will be used to find PageRank Scores in which:

- Each link is considered as a vote and the importance of a given vote is proportional to the importance of the source web page that is casting this vote or that is creating a link to the destination.
- Suppose that there is page j with an importance r_j.
- The page j has n outgoing links.
- This importance r_j of page j basically gets split on all its outgoing links evenly.

- Each link gets r_j divided by n votes i.e., r_j/n votes.

This is how the importance gets divided from *j* to the pages it points to. And in a similar way, you can define the importance of a node as the sum of the votes that it receives on its in-links.

Figure 2 shows a simple graph that shows the votes related to a page labeled as *j*. You can calculate the score of node *j* as a sum of votes received from r_i and r_k, using the following formula:

$$r_j = r_i/2 + r_k/3$$

This is so because page *i* has 3 out links and page *k* has 4 out links. The score of page *j* further gets propagated outside of *j* along with the three outgoing links. So, each of these links get the importance of node *j* divided by 3.

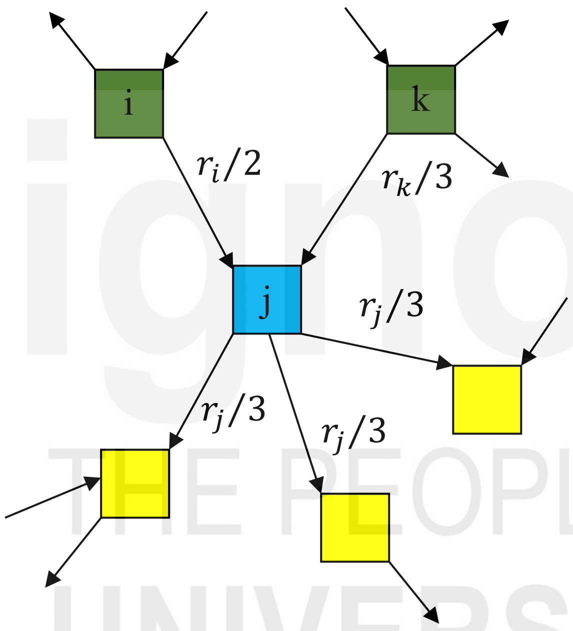


Figure 2: A Graph to calculate score of pages [2]

This is basically the mechanism of how every node collects the importance of the pages that points to it and then propagate it through the neighbours.

The Flow Model: This model is basically the vote flow through the network. So that is why this is called the flow formulation or a flow model of PageRank. To understand the concept, let us use a web graph of WWW that contains only three web pages named *a, b,* and *c* (Please refer to Figure 3).

- *a* has a self-link and then it points to *b*
- *b* points to *a* and *c*
- *c* points backwards to *b* only.

The vote of an important page has more weight than the normal page. So a page is more important if it is pointed to by other important pages.

Formula to compute PageRank:

$$r_j = \sum_{i \to j} \frac{r_i}{d_i}$$

Where d_i is the out-degree of node i. You may please recall that out degree of a node is number of links that go out of it. Thus, importance score of page j is simply the sum of all the other pages i that point to it. The importance of a page i is computed by dividing its importance by the out degree.

This means that for every node in the network, we obtain a separate equation based on the number of links. For example, the importance of node a in the network is simply the importance of a divided by 2 plus the importance of b divided by 2. Because a has 2 outgoing links and then similarly node b has 2 outgoing links. Thus, the three equations, one each for individual node, for the Figure 3, would be:

$$r_a = r_a/2 + r_b/2$$

$$r_b = r_a/2 + r_c$$

$$r_c = r_b/2$$

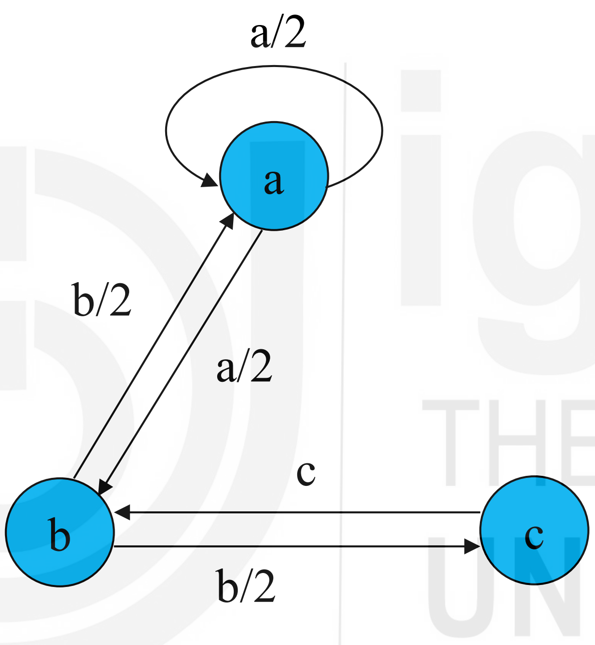


Figure 3: A very simple web graph [2]

The problem is that these three equations do not have a unique solution. So, there is a need to add an additional constraint to make the solution unique. Let us add a constraint - *The total of all the PageRank scores should be one*, which is given by the following equation:

$$r_a + r_b + r_c = 1$$

You can solve these equations and find the solution to the PageRank scores, which is:

$$r_a = 2/5;\ r_b = 2/5;\ r_c = 1/5$$

Problem: This approach will work well for small graphs, but it would not work for a graph, which has a billion web pages, as this would result in development of a billion of equations or system of billions of equations that we would be required to solve for computing the PageRank. So, you need a different formulation.

11.4.2 Web Structure and Associated Issues

In a web structure, the original idea was to manually categorize all the pages of the WWW into a set of groups. However, this model is not scalable, as web has been growing far too quickly.

Another way to organize the web and to find things on the web is to search the web. There is a rich literature, particularly in the field of information retrieval, that covers the problem of how do you find a document in a large set of documents. You can think of every web page as a document, the whole web is one giant corpus of documents, and your goal would be to find relevant document based on a given query from this huge set.

The traditional information retrieval field was interested in finding these documents in relatively small collection of trusted documents. For example, finding a *newspaper collection*. However, now the web is very different. The web is huge and full of untrusted documents, random things, spam, unrelated things, and so on.

Issues: The associated issues are:

- Which web pages on the web should you trust?
- Which web pages are legitimate and which are fake or irrelevant?

So, the solution is PageRank in which trustworthy webpages will be linked to each other.

But the other problem that happens on the web is that sometimes queries can be rather ambiguous. For example, the users may ask; What is the best answer to a query word "*newspaper*"? However, if a user wants to identify all the good newspapers on the web, then PageRank algorithm need to again look at the structure of the web graph in order to identify the set of pages or a set of good newspapers that are linking to each other. And again, get the result out of the structure of the web graph.

The way to address these challenges is to basically realize that the web as a graph has very rich structure. So, there is a need to rank nodes of this big graph. Basically, we would like to compute a score or an important notch of every node in this web graph. The idea is that some nodes will collect lots of links. So they will have high importance and some other nodes will have a small number of links or links from untrusted sources, so they will have low importance.

There are several approaches to compute the importance of the nodes in a graph. Broadly these approaches are called as *link analysis* because we are analyzing the links of the web graph to compute an important score of a node in a graph.

Check Your Progress 1:

Question 1: What is the basic principle of flow model in PageRank?

Question 2: How can you represent webpages and their links?

Question 3: What are the issues associated with the web structure?

11.5 USE OF PAGERANK IN SEARCH ENGINES

So far, you know that the importance of page *j* in a web graph is the sum of the importance of page *i* that point to it divided by the out-degree of the i^{th} node. This can be represented using the following equation, as given earlier:

$$r_j = \sum_{i\to j} \frac{r_i}{d_i}$$

Where r_i is the score of the node *i,* and d_i is its out-degree.

This concept has also been used in Google formulation of the PageRank algorithm. Consider a graph, as given in Figure 4, consisting of 5 nodes. The rank of each of these nodes can be represented using the equations given below.

$r_a = r_d/2 + r_e/2$ (as node *a* has in-links from node d and node e both of which has two out-links.)

$r_b = r_a/3 + r_e/2$ (as node *b* has in-links from node a, which has three out-links and node e, which has two out-links.)

$r_c = r_a/3 + r_b/2$ (as node *c* has in-links from node a, which has three out-links and node b, which has two out-links.)

$r_d = r_a/3 + r_b/2 + r_c$ (as node *d* has in-links from node a, node b and node c node, these nodes have three, two and one out-links respectively.)

$r_e = r_d/2$ (as node *e* has in-links from node d, which has two out-links.)

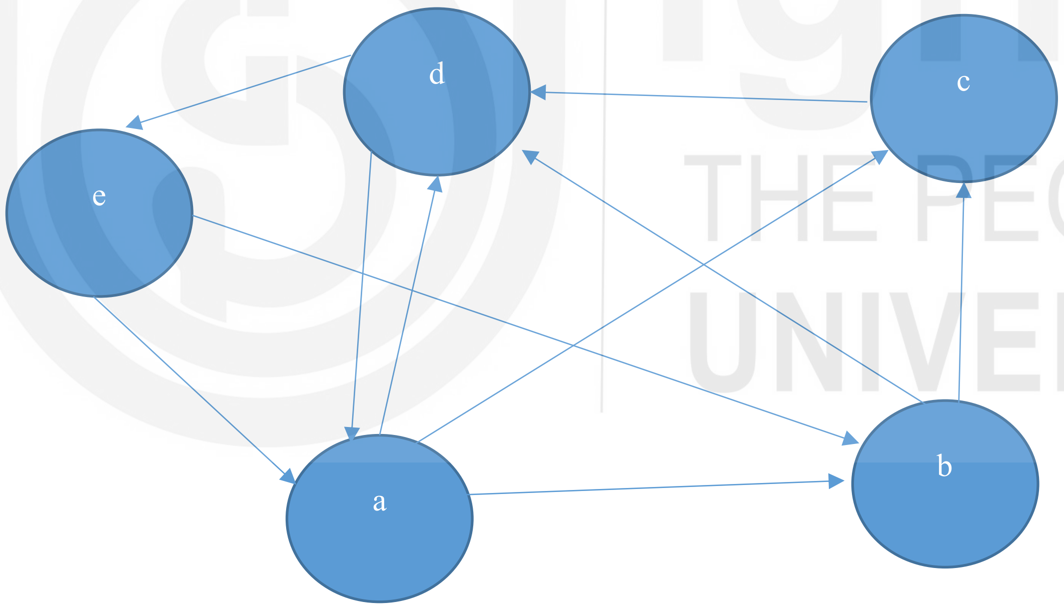


Figure 4: Directed graph with 5 nodes

The graph in Figure 4 can be written with the following matrix, where *i^th^* row is representing the *i^th^* node. The matrix represents the equations given above.

$$M=\begin{pmatrix} 0 & 0 & 0 & 1/2 & 1/2 \\ 1/3 & 0 & 0 & 0 & 1/2 \\ 1/3 & 1/2 & 0 & 0 & 0 \\ 1/3 & 1/2 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1/2 & 0 \end{pmatrix}$$

In order to compute the PageRank, a recursive formulation is used, which is represented using the following Matrix equation:

$$r^{t+1} = M\, r^{t} \qquad (1)$$

Where r^t is the PageRank at the t^{th} iteration, and Matrix M is the link matrix, as shown above.

The following example explains this concept.

Example: Consider the graph of Figure 3 and compute the PageRank using Matric M, assuming the starting value of PageRank as:

$$\begin{matrix} r_a \\ r_b \\ r_c \end{matrix} = \begin{matrix} 1/3 \\ 1/3 \\ 1/3 \end{matrix}$$

The matrix $M = \begin{matrix} 1/2 & 1/2 & 0 \\ 1/2 & 0 & 1 \\ 0 & 1/2 & 0 \end{matrix}$

Application of equation (1) will be:

$$r^1 = M\, r^0$$

$$r^1 = \begin{matrix} 1/2 & 1/2 & 0 \\ 1/2 & 0 & 1 \\ 0 & 1/2 & 0 \end{matrix} \quad \begin{matrix} 1/3 \\ 1/3 \\ 1/3 \end{matrix}$$

$$r^1 = \begin{matrix} 1/3 \\ 1/2 \\ 1/6 \end{matrix}$$

On applying the equation again, the value would be:

$$r^2 = \begin{matrix} 1/2 & 1/2 & 0 \\ 1/2 & 0 & 1 \\ 0 & 1/2 & 0 \end{matrix} \quad \begin{matrix} 1/3 \\ 1/2 \\ 1/6 \end{matrix}$$

$$r^2 = \begin{matrix} 5/12 \\ 1/3 \\ 1/4 \end{matrix}$$

On repeated application, you will get:

$$r^3 = \begin{matrix} 3/8 \\ 11/24 \\ 1/6 \end{matrix}$$

$$r^4 = \begin{matrix} 20/48 \\ 17/48 \\ 11/48 \end{matrix}$$

$$r^5 = \begin{matrix} 37/96 \\ 42/96 \\ 17/96 \end{matrix}$$

$$r^6 = \begin{matrix} 79/192 \\ 71/192 \\ 42/192 \end{matrix}$$

You may observe that PageRank of a = 0.41; PageRank of b = 0.37 and PageRank of c = 0.22, which are converging to 0.4, 0.4 and 0.2 respectively.

Next, you will get answers to the following two questions:

Question 1: Does the equation (1) converge?

Question 2: Does the equation (1) converge to what we want?

Let us answer these questions one by one.

<u>So, here is the first question: Does the equation (1) converge?</u>

We multiply the matrix *M* with *r* to compute *r* iteratively. From an initial approximation of the RageRank that can be initialized randomly, the algorithm will update it until convergence.

Imagine a very simple graph as shown in Figure 5 where:

- There are two nodes and node *a* points to node *b* and node *b* points back to node *a.*
- If we initialize vector *r*, or *r* at time 0, to have two values: value 1 on the node *a* and value 0 on node *b*.
- The value of Matrix *M* for this graph would be: $M = \begin{matrix} 0 & 1 \\ 1 & 0 \end{matrix}$
- Now when we are multiplying *n* times *r*, then following is the outcome:

$$r^1 = \begin{matrix} 0 & 1 \\ 1 & 0 \end{matrix} \quad \begin{matrix} 1 \\ 0 \end{matrix} = \begin{matrix} 0 \\ 1 \end{matrix}$$

$$r^2 = \begin{matrix} 0 & 1 \\ 1 & 0 \end{matrix} \quad \begin{matrix} 0 \\ 1 \end{matrix} = \begin{matrix} 1 \\ 0 \end{matrix}$$

Likewise

$$r^3 = \begin{matrix} 0 \\ 1 \end{matrix} \, and \, r^4 = \begin{matrix} 1 \\ 0 \end{matrix}$$

- So, in the next time step, the values will flip. And now when we multiply again, the value will flip again.

So, what we see here is that we will never converge because the score of 1 gets passed between *a* and *b* and, a score of 0 gets passed between *b* and *a*. So it seems that PageRank computation will never converge. This problem is called the **spider trap problem**.

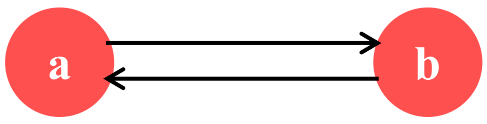


Figure 5: Spider Trap Problem [2]

<u>So, here is the second question: Does the equation (1) converge to what we want?</u>

Let us consider a very simple graph as shown in Figure 6 where:

- There are 2 nodes: node *a* and node *b* and one edge.
- Here, *a* starts with score 1 and *b* starts with score 0.
- The matrix M in this case would be: $M = \begin{matrix} 0 & 0 \\ 1 & 0 \end{matrix}$

$$r^1 = \begin{matrix} 0 & 0 \\ 1 & 0 \end{matrix} \quad \begin{matrix} 1 \\ 0 \end{matrix} = \begin{matrix} 0 \\ 1 \end{matrix}$$

$$r^2 = \begin{matrix} 0 & 0 \\ 1 & 0 \end{matrix} \quad \begin{matrix} 0 \\ 1 \end{matrix} = \begin{matrix} 0 \\ 0 \end{matrix}$$

- The first multiplication with matrix *M*, the scores get flipped i.e., $\begin{tikzcd} 1 \arrow[dr] & 0 \\ 0 \arrow[ur] & 1 \end{tikzcd}$
- But in the second step of multiplication is basically the score of 1 gets lost. Thus, *a* and *b* are not able to pass the score to anyone else. So the score gets lost.
- And we converge to this vector of zeros, which is a problem.

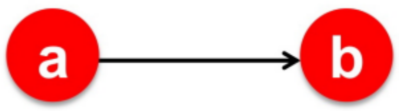


Figure 6: Dead end Problem [2]

The spider trap and dead end problem in PageRank:

In dead end problem, the dead ends are those web pages that have no outgoing links as shown in Figure 7. Such pages cause importance to "leak out". The idea behind is that, whenever a web page receives its PageRank score, then there is no way for a web page to pass this PageRank score to anyone else because it has no out-links. Thus, PageRank score will "leak out" from the system. In the end, the PageRank scores of all the web pages will be zero. So this is called the Dead End problem.

ignou THE PEOPLE'S UNIVERSITY

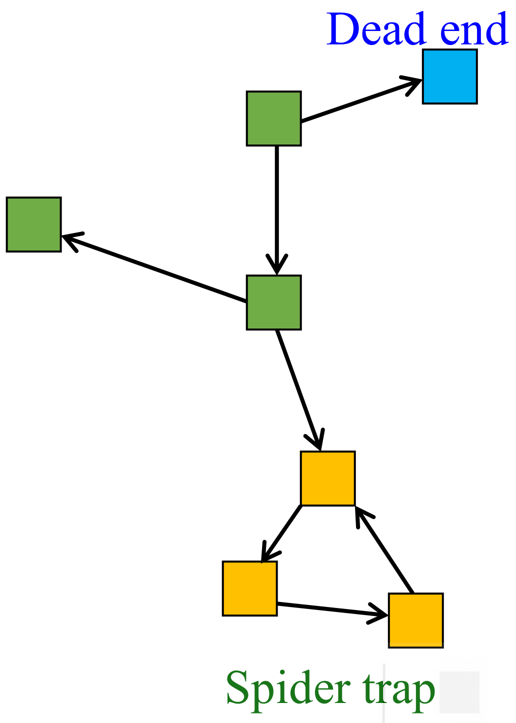


Figure 7: Dead end and Spider trap Problem [2]

In the spider trap problem, out-links from webpages can form a small group as shown in Figure 7. Basically, the random walker will get trapped in a single part of the web graph and then the random walker will get indefinitely stuck in that part. At the end, those pages in that part of the graph will get very high weight, and every other page will get very low weight. So, this is called the problem of spider traps.

Solution to spider trap problem: Random Teleports

A random walk is known as a random process which describes a path that includes a succession of random steps. Figure 8(a) shows a graph, with node *c* being spider trap. So whenever a random walker basically stuck in an infinite loop because there is no other way.

The way Google has solved the problem to the spider traps, is to say that at each step, the random walker has two choices. With some probability β occurs, the random walker will follow the outgoing link.

So, the way we can think of this now is that we have a random walker that whenever a random walker arrives to a new page, flips a coin and if the coin says yes, the random walker will pick another link at random and walk that link and, if the coin says no then the walker will randomly teleport basically jump to some other random page on the web.

So this means that the random walker will be able to jump out or teleport out from a spider track within only a few time steps.

After a few time steps, the coin will say yes, let us teleport and the random walker will be able to jump out of the trap.

Figure 8(a) shows a graph, with node *m* being spider trap where random walker will teleport out of spider trap within a few time steps.

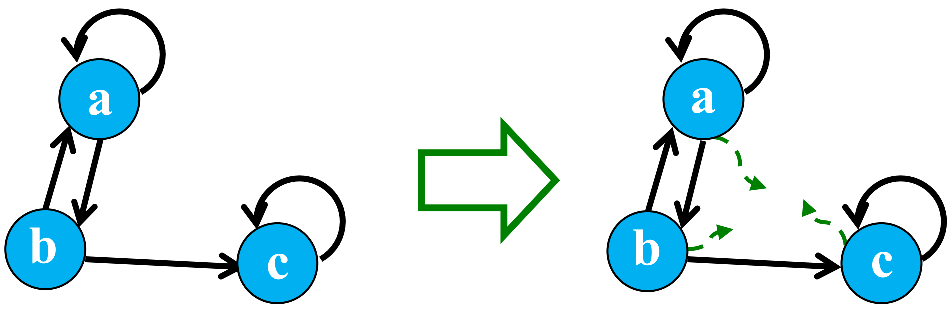


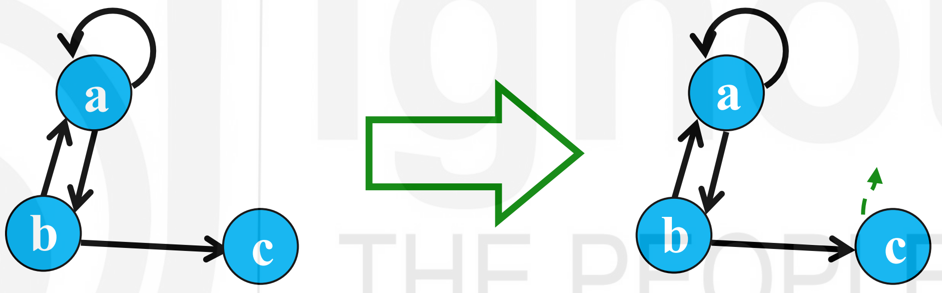
Figure 8(a): Random teleports approach [2]

Solution to dead end problem: Always Teleport

Figure 8(b) shows that if a node has no outgoing links and when we reach that node, we will teleport with probability 1. So, this basically means that whenever you reach node *m*, you will always jump out of it, i.e., you will teleport out to a random web page.

So in stochastic (a stochastic matrix is a square matrix whose columns are probability vectors.) matrix *M*, column one will have values of 1/3 for all its entries. Basically whenever a random surfer comes to end, it teleports out and with probability 1/3 lands to any other node in the graph. So this is again the way to use the random jumps or random teleports to solve the problem of dead ends.

ignou THE PEOPLE'S UNIVERSITY



	a	b	c
a	$^1/_2$	$^1/_2$	0
b	$^1/_2$	0	0
c	0	$^1/_2$	0

	a	b	c
a	$^1/_2$	$^1/_2$	$^1/_3$
b	$^1/_2$	0	$^1/_3$
c	0	$^1/_2$	$^1/_3$

Figure 8(b): Always teleport approach [2]

So, the page rank equation discussed in section 11.5 is re-arranged into a different equation:

$$r_j = \sum_{i \to j} \beta M.r + \left[\frac{1-\beta}{N}\right]_N$$

Where M is a sparse Matrix (with no-dead ends). In every step, the random surfer can either follow a link randomly with probability *B* or jump to some random page with probability 1-β. The β is constant and its value generally lies in a range 0.8 to 0.9.

$\left[\frac{1-\beta}{N}\right]_N$ is a vector with all N entries $\left[\frac{1-\beta}{N}\right]$.

So in each iteration of page rank equation, we need to compute the product of matrix M with old rank vector:

$$r_{new} = \beta M . r_{old}$$

and then add a constant value $\left[\frac{1-\beta}{N}\right]$ to each entry in r_{new}.

Therefore, the Complete Algorithm of PageRank is discussed as follows:

Input: Graph G and parameter β

Directed Graph G with spider traps and dead ends

Output: PageRank vector r

Set: $r_j^{(0)} = \frac{1}{N}$, t=1

Do:

$$\forall j: r_j'^{(t)} = \sum_{i \to j} \beta \frac{r_i^{(t-1)}}{d_i}$$

$$r_j'^{(t)} = 0 \; if \; in - deg. of \; j \; is \; 0$$

Now re-insert the leaked PageRank:

$$\forall j: r_j^{(t)} = r_j'^{(t)} + \frac{1-S}{N} \; where: S = \sum j \, r_j'^{(t)}$$

t = t + 1

while $\sum j \left| r_j^{(t)} - r_j^{(t-1)} \right| > \in$

In the above algorithm, a directed graph G is given as an input along with its parameter *β*. The graph may have spider traps and dead ends. The algorithm will give a new Page rank vector *r*. If the graph does not have any dead-end then the amount of leaked PageRank is 1-*β*. On the other hand, if there are dead-ends, the amount of leaked PageRank may be larger. This initial equation assumes that matrix M has no dead ends. It can be either preprocessed to remove all dead ends or it has to explicitly follow random teleport links with probability 1.0 from dead-ends. If M has dead-ends then $\sum j \, r_j'^{(t)} < 1$ and you also have to renormalize r' so that it sums to 1. The computation of new page rank is done repeatedly until the algorithm converges. The convergence can be checked by measuring the difference between the old and new page rank value. The algorithm has to explicitly account for dead ends by computing S.

11.5.1 PageRank Computation Using MapReduce

MapReduce approach solves the problem by performing parallel processing using cluster and grid computing. This approach scaled up to very large link-graphs. Figure 8 shows traditional versus MapReduce approach.

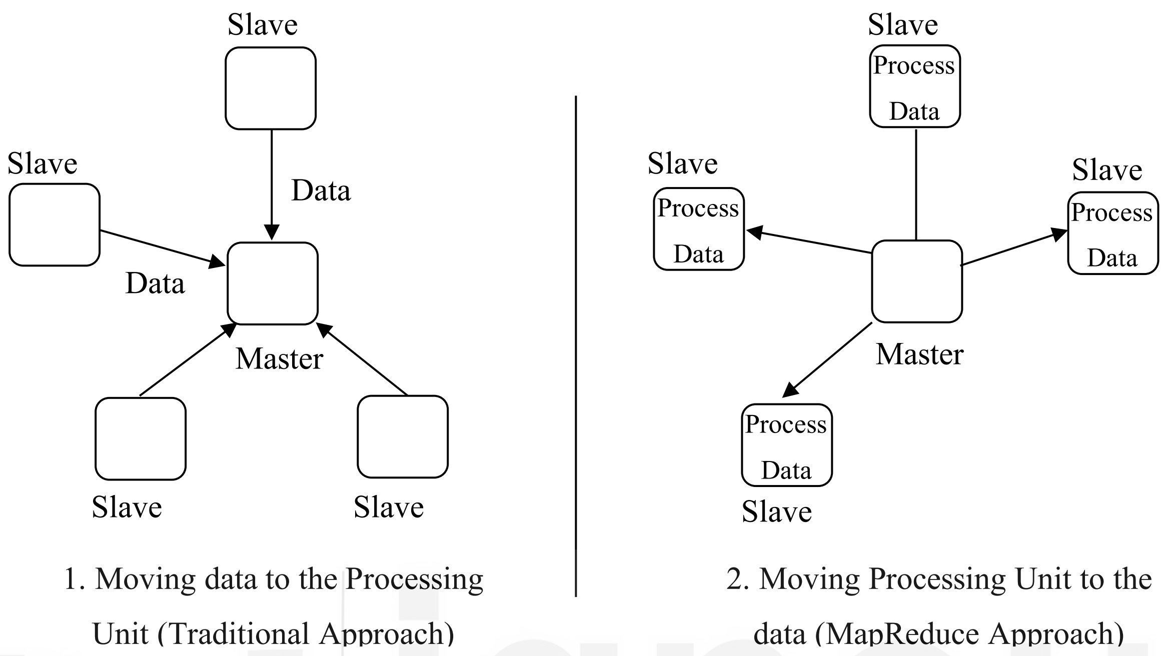

Slave
Slave
Data
Data
Master
Slave
Slave
1. Moving data to the Processing Unit (Traditional Approach)
Slave
Process Data
Slave
Process Data
Slave
Process Data
Master
Process Data
Slave
2. Moving Processing Unit to the data (MapReduce Approach)

Figure 9: Traditional vs. MapReduce approach [3]

ignou THE PEOPLE'S UNIVERSITY

There are 4 steps in MapReduce approach as shown in Figure 9 and are explained as follows:

Step 1-Input Split: The input data is raw and is further divided into small chunks called input splits. Each chunk will be an input of a single map and the data input will make a key-value pair(key1, Value1).

Step 2- Mapping: A node which is given with a map function takes the input and produces a set of (key2, value2) pairs, shown as ($K_{2,}$ V_2) in Figure 10. One of the nodes in the cluster is called as a "Master node" which is responsible for assigning the work to the worker nodes called as "Slave nodes". The master node will ensure that the slave nodes perform the work allocated to them. The master node saves the information (location and size) of all intermediate files produced by each map task.

Step 3- Shuffling: The output of the mapping function is being clustered by keys and reallocated in a manner that all data with the same key are positioned on the same node. The output of this step will be (K_2, list(V_2)).

Step 4- Reducing: The nodes will now process respective group of output data by aggregating the shuffle phase values (output). The ultimate output will be from the shape of (list (K_3, V_3)).

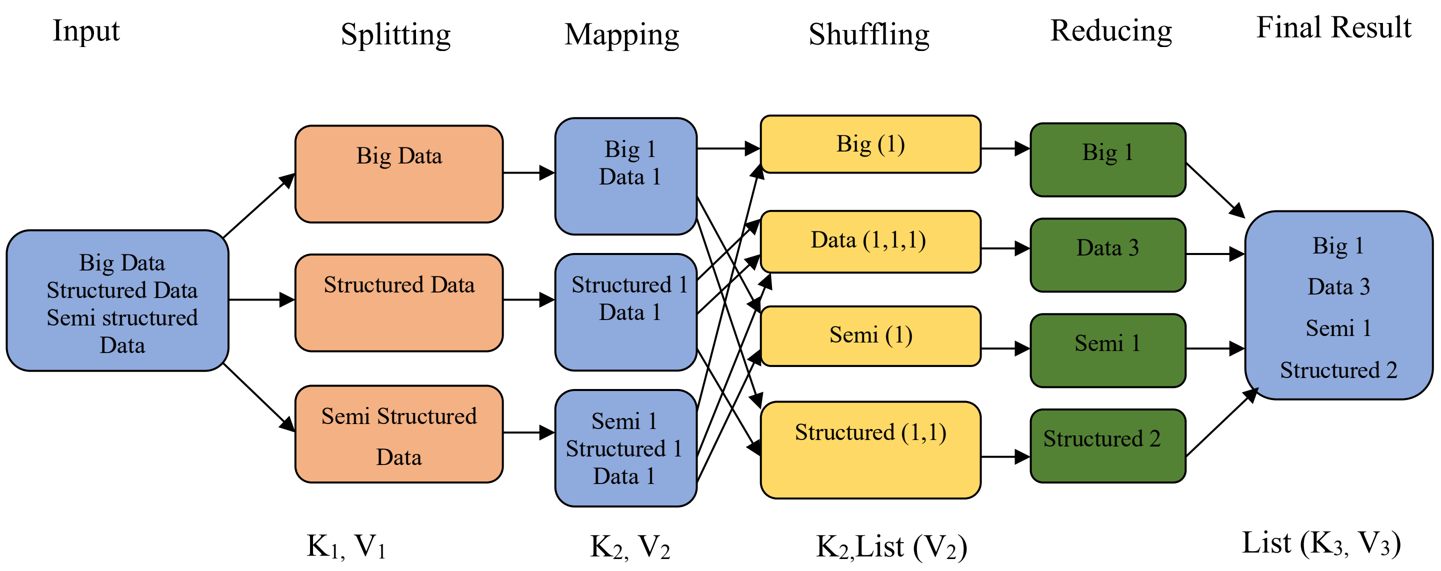
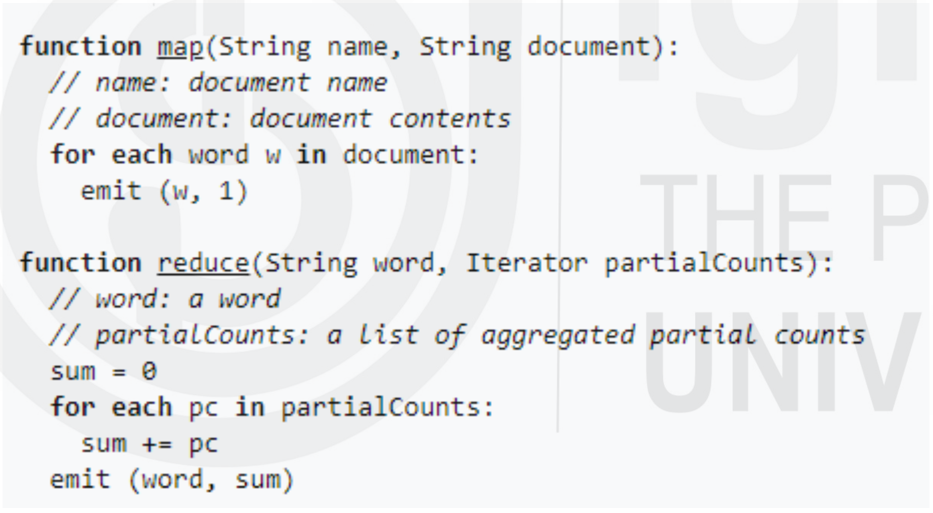


Figure 10: Example of MapReduce approach [3]

Figure 11 shows the pseudocode of the MapReduce approach with map and reduce functions.



```
function map(String name, String document):
  // name: document name
  // document: document contents
  for each word w in document:
    emit (w, 1)

function reduce(String word, Iterator partialCounts):
  // word: a word
  // partialCounts: a list of aggregated partial counts
  sum = 0
  for each pc in partialCounts:
    sum += pc
  emit (word, sum)
```

Figure 11: Pseudocode of MapReduce approach [3]

The MapReduce Program for PageRank computation is discussed as follows:

```
import findspark
findspark.init()
findspark.find()
import pyspark
findspark.find()
from pyspark import SparkContext, SparkConf
from pyspark.sql import SparkSession
conf =
pyspark.SparkConf().setAppName('appName').setMaster('local')
sc = pyspark.SparkContext(conf=conf)
spark = SparkSession(sc)
```

ignou THE PEOPLE'S UNIVERSITY

```
# Adjacency list
links = sc.textFile('links.txt')
links.collect()


# Key/value pairs
links = links.map(lambda x: (x.split(' ')[0], x.split('
')[1:]))
print(links.collect())

# Find node count
N = links.count()
print(N)

# Create and initialize the ranks
ranks = links.map(lambda node: (node[0],1.0/N))
print(ranks.collect())


ITERATIONS=20
for i in range(ITERATIONS):
    # Join graph info with rank info and propagate to all
neighbors rank scores (rank/number of neighbors)
    # And add up ranks from all in-coming edges
    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)
    print(ranks.sortByKey().collect())
```

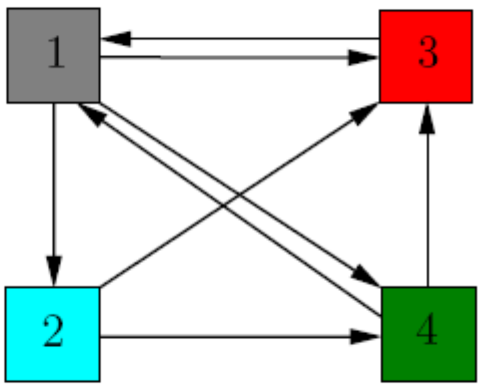
The program receives document as page input (webpages or xml data). The page is parsed using a regular expression method to extract the page title and its outgoing links. While computing the PageRank using MapReduce program, at first the web graph is split into some partitions and each partition is sorted as an adjacency matrix file. Each Map task will process one partition and computes the partial rank score for some pages. The Reduce task will then merge all the partial scores and produces the global Rank values for all the web page. Initially, page identifier and its outgoing link are extracted as key-value pair. Then node count is done and then we initialize the ranks for each page. After that in each iteration, join the graph information with rank information and propagate the same to all neighbors rank scores (rank/number of neighbors). Further, it adds up ranks from all in-coming edges to generation the final score.

Check Your Progress 2:

1 Explain the spider trap and dead-end problem in PageRank. What are the solutions for the spider trap and dead-end problem?

2. Why MapReduce paradigm is suitable for computation of PageRank?

3. Given the following graph, compute the matrix of PageRank Computation.



11.6 TOPIC SENSITIVE PAGERANK

Topic sensitive PageRank is also known as Personalized PageRank.

Earlier, the initial goal was to identify the important pages on the web. Now you do not necessarily want to find pages that have generically high PageRank score, but you would like to say: Which are the web pages that are popular within a given topic or within a given domain?

So, in order to identify this, your goal will be following:

- The web page should not be assessed only on the basis of the overall popularity, but also how closely they are to a particular set of topics, e.g., "sports" or "history".
- This is interesting because if you think of the web search the way PageRank was initially thought of was that somebody will come and ask a web search query, you will go and identify all the web pages that are relevant towards that web search query.
- But now you need to decide how to rank or present all these web pages to the user, you would basically take pages, simply sort them by their PageRank score and show the pages that have the highest PageRank score first to the user.
- Now of course, if you would have the personalized PageRank or topic specific PageRank for measuring the importance of a page, you could basically show to the user a given ranking depending on what the user wants, however, in particular, there can be many queries that are ambiguous.
- For example, a query Trojan could have very different relevant pages depending on what is the topic you are interested in, in a sense that Trojans could mean a sports team. It can also mean something different, if you are interested in history, or if it could mean something very different if you are interested in Internet security.
- In any case, the idea would be that you would like to compute different importance scores of different web pages based on their relation to a given topic.

The way to achieve this is by changing the random teleportation part. So, in the original PageRank formulation, the random walker can land at any page with equal probability. In the personalized PageRank, the random walker can teleport only to a topic or specific set of relevant pages.

So, whenever a random walker decides to jump, they do not jump to any page on the web, but they only jump to a small subset of pages and this subset of pages is called the teleport set. So, the idea here is, in some sense, that we are biasing the random walk i.e., when the walker teleports, they can only teleport into a small set of pages.

- Consider *S* as the teleport set.
- The set *S* contains only pages that are relevant to a given topic.
- This allows us to measure the relevance of all the other web pages on the web with regard to this given set *S*.
- So, for every set *S*, for every teleport set, you will now be able to compute a different PageRank vector *R* that is specific to those data points.

To achieve this, you need to change the teleportation part of the PageRank formulation:

$$A_{ij} = \begin{matrix} \beta\, M_{ij} + \dfrac{1-\beta}{|S|}\, if\ i\ \epsilon S \\ \beta M_{ij}\ otherwise \end{matrix}$$

Where *A* is stochastic matrix,

i is the entry,

S is the set,

$\frac{1-\beta}{|S|}$ is the random jump probability

- If the entry *i* is not in the teleport set *S* then basically nothing happens.
- But if our entry *i* is in the teleport set, then add the teleport edges.
- *A* is still stochastic.

The idea here is basically that you have lot of freedom in how to set the teleport set *S*. For example, when teleport set *S* is just a single node. This is called a random walk with restarts.

11.7 LINK SPAM

Link spamming is any deliberate activity to boost the score of the web page position in the search engine result page. So, link spam in some sense are web pages that are the result of spamming. Search engine optimization is the process of improving the quality and quantity of website traffic to a website or a web page from search engines.

Earlier, search engines operated the following way:

- Search engine would crawl the web to collect the web pages.
- It would then index the web pages by the words they contained and then whenever somebody searched, it will basically look at the given web page, see how much does it mention the words in the search query, and then it will rank the search results based on the number of times they mentioned a given search query term.
- So, the initial ranking was an attempt to order pages by matching a search query by importance, which means that basically the search engines would consider the number of times a query words appeared on the web page. In addition, it may also use the prominence of the word position, i.e., whether that word was in the title or in the header of a web page.

So now, this allows spammers to nicely spam and exploit this idea. Consider the following scenario:

- Imagine that a spammer has just started a business and he has made a web site. He wants that search engine should give preference to his web pages, as it is in his commercial interest. The spammer wants to exploit the search engines and make sure that people go and visit his web pages.

- So, spammer wants his web pages to appear as high on the web search ranking as possible, regardless of whether that page is really relevant to a given topic or not, as driving traffic to a website is good.

The spammer did this by using a technique called link spam. The following example illustrates the link spam:

- There is a T-shirt seller, she/he creates a web page and wants that her/his web page should appear, when any web searcher searches the word "theatre", as a person in theatre may be interested in her/his T-shirts.
- So this online seller can insert the word "theatre" 1000 times in the webpage. How can this be done? In the early days of the web, you would have a web page, which on the top of the page had the legitimate text, and at the bottom of the web page, there the seller inserts a huge list of words "theater". This list would use the same text color of these words as that of the color of the background.
- So, these words would not bother the user who comes to the web page. But the web search engine would see these words and would think that this particular webpage is all about theatre.
- When, of course, somebody will run a query for the word theatre, this web page would appear to be all about theatre and this page get it ranked high. This and similar techniques to this one are called term spam.

Spam farm were developed to concentrate PageRank on a single page. For example, a t-shirt seller creates a fake set of web pages that they all link to his own webpage, and all these web pages in the anchor text says that the target page is about movies. This is known as Spam farming.

Google came up with a solution to combat link spam. According to Google; Rather than believing what the page says about itself, let us believe what other people say about the page. This means to look at the words in the anchor text and its surrounding text. The anchor text in a web page contains words that appear underlined to the user to represent the link. The idea is that you are able to find web pages, even for the queries/words that the webpage itself does not even mention, but other web pages on the web may mention that word when referring to the target page.

The next section discusses a similar technique that overcomes the problem of link spam.

11.8 HUBS AND AUTHORITIES

The basic idea in hubs and authority model is that every web page is assigned two scores, not just one, as is the case of PageRank algorithm. One type of score is called a hub score and the other one will be called as an authority score.

An authority value is computed as the sum of the scaled hub values that point to a particular web page. A hub value is the sum of the scaled authority values of the pages it points to.

The basic features of this algorithm are:

- The hubs and authority based algorithm is named HITS algorithm, which is Hypertext Induced Topic Selection.
- It is used to measure the importance of pages or documents in a similar way to what we did with PageRank computation.
- For example, we need to find a set of good newspaper webpages. The idea is that we do not just want to find good newspapers, but in some sense, we want to find specialists (persons) who link in a coordinated way to good newspapers.

So, the idea is similar to PageRank such that we will count links as votes.

- For every page, you will have to compute its hub and its authority score.
- Each page will basically have a quality score of its own as an expert. We call this as a hub score
- Each page also has a quality score of it as a content provider, so we call it an authority score.
- So, the idea would be that the hub score is simply the sum of all the votes of the authorities that are pointed to. The authority score will be the sum of the experts votes that the page is receiving.
- We will then apply the principle of repeated improvement to compute the steady state score.

Hence, the way we will think about this is that generally pages on the web fall into two classes, hubs and authorities as shown in Figure 12.

Class of Authorities: These are the pages which contain useful information or content. So in our case of newspaper example, these are newspaper webpages.

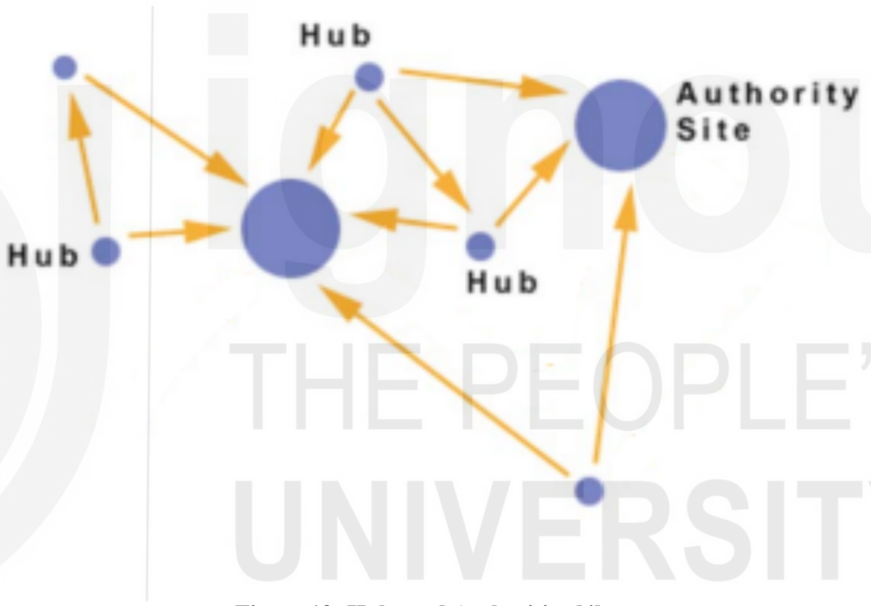


Figure 12: Hubs and Authorities [4]

Class of Hub: where pages link to good authorities or list of good things on the web. So in newspaper example, these are my favorite newspapers and that page would link to them.

The HITS algorithm, thus, minimizes the problem due to link spam and users would be able to get good results, as high-ranking pages.

Check Your Progress 3:

1. Explain the mechanism of link spam.
2. What is term spam?
3. What is a Hub score and Authority score in HITS algorithm?

11.9 SUMMARY

This unit introduces the concept of link analysis along with its purpose to create connections in a set of data. Further, PageRank algorithm is discussed with the calculation of page scores of the graph. Different mechanisms of finding PageRank such as Simple Recursive Formulation and the flow model with the associated problems are also presented in this Unit. The web structure with its associated issues are also included, which shows various ways to address these challenges. The use of PageRank in search engine page is also discussed with spider trap and dead end problem and their solutions. Further, this Unit shows rank computation using map-reduce approach and topic sensitive PageRank. The Unit also introduces the link spam and how Google came up with a solution to combat link spam. Lastly, hubs and authorities algorithm is explained, which can be used to compute PageRank.

11.10 SOLUTIONS/ANSWERS

ignou THE PEOPLE'S UNIVERSITY

Check Your Progress 1:

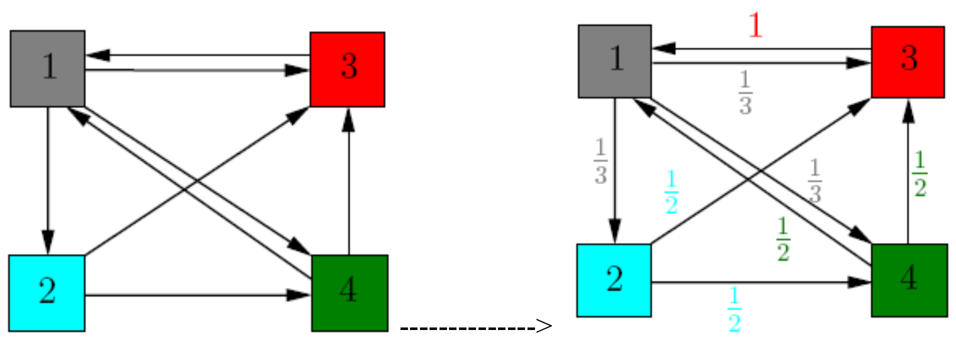
1. The flow model is the flow formulation of PageRank to show how the vote (or links)) flow through the network. The weight of a vote is computed by diving the vote of a node divided by the number of links it has. A higher vote of a node means that it has a higher probability that a user may reach at that web page.
2. Graphs are used to represent webpages. A Node represents a webpage and directed link between the nodes represents hyperlink from a web page to other web page.
3. The issues associated with the web structure are to find out the web pages on which users can trust and which of the web pages are illegitimate or fake.

Check Your Progress 2:

1. In dead end problem, the dead ends are those web pages that have no outgoing links. In the spider trap problem, out-links from webpages can form a small group where the random walker will get trapped. The solution to spider trap problem is "Random Teleports". The solution to dead end problem is "Always Teleport".

2. MapReduce paradigm is suitable for computation of PageRank because it solves the problems by performing parallel processing using cluster and grid computing. This approach is scaled up to very large link-graphs.

3.



Matrix= $\begin{bmatrix} 0 & 0 & 1 & \frac{1}{2} \\ \frac{1}{3} & 0 & 0 & 0 \\ \frac{1}{3} & \frac{1}{2} & 0 & \frac{1}{2} \\ \frac{1}{3} & \frac{1}{2} & 0 & 0 \end{bmatrix}$

Check Your Progress 3:

1. Link spamming is any deliberate activity to boost the score of the web page position in the search engine result page, by creating links from bogus webpages.
2. Term spam is a technique where somebody will run a query for the word *e.g.,* theatre, but some other web page (*e.g.,* T-shirt webpage) will appear with high rank.
3. The hub score is simply the sum of all the votes of the authorities that are pointed to. The authority score will be the sum of the experts votes that the page is receiving.

11.11 REFERENCES/FURTHER READINGS

[1] Liu D. "Big Data Technology." https://davideliu.com/2020/02/27/analysis-of-parallel-version-of-pagerank-algorithm/

[2] "Link Mining." https://www.inf.unibz.it/~mkacimi/PageRank.pdf

[3] Taylan K. "MapReduce Programming Model." https://taylankabbani96.medium.com/mapreduce-programming-model-a7534aca599b

[4] Leskovec J. "Link analysis: Page Rank and Similar Ideas." http://snap.stanford.edu/class/cs246-2012/slides/10-hits.pdf

ignou THE PEOPLE'S UNIVERSITY