Heidelberg University
Winter semester 2024/25

Group Artificial Intelligence for Programming (AIP)
Prof. Dr. Artur Andrzejak, Kim Tuyen Le

Problem Set 9 for lecture Mining Massive Datasets

Due January 20, 2025, 23:59 CET

Exercise 1 (2 points)

In this exercise, we will use the graph in Figure 1.

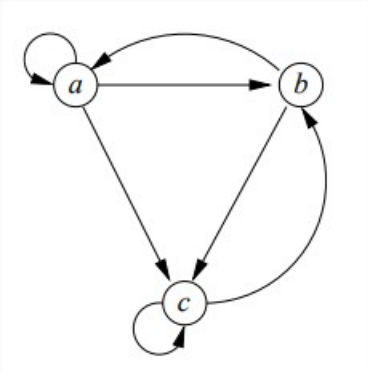

Figure 1: Example graph for manual computations

a) Compute by hand the PageRank of each page represented in Figure 1 (using the "Flow/Matrix" Formulation, i.e. without random teleports). Use the Power Iteration method with $\varepsilon = \frac{1}{12}$. Submit as your solution the matrix M and report the vector r at each step of the iteration.

b) What is the stationary distribution of the random walk in part **a)**?

Hint: calculate the eigenvector of matrix M with eigenvalue 1.

c) Repeat part **a)** with the random teleports method, assuming $\beta = 0.8$.

d) What is the stationary distribution of the random walk in part **c)**?

Exercise 2 (3 points)

Suppose the Web consists of a clique (set of nodes with all possible arcs from one to another) of n nodes and a single additional node that is the successor of each of the n nodes in the clique (so $N = n + 1$). Figure 2 shows this graph for the case $n = 4$.

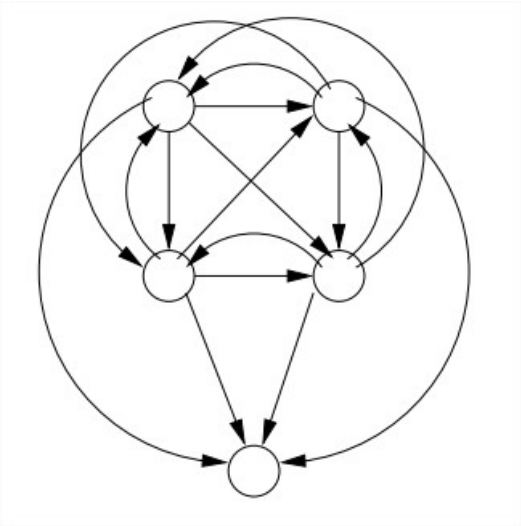

Figure 2: Example graph with a clique of size $n = 4$

a) Explain briefly why all nodes in the clique have the same PageRank value.

b) State the general form of the matrix A like in the Google formulation (see slide "The Google Matrix" of the lecture), but with the assumption that we explicitly follow random teleport links with probability 1.0 from dead-ends (this assumption changes slightly the 2nd matrix contributing to A).

c) Assuming that we explicitly follow random teleport links with probability 1.0 from dead-ends, determine the PageRank of each page as a function of n and β, i.e. as a formula which applies for any $n > 1$ and any $\beta \in (0, 1)$.

Hint: set up equations for the PageRank x of any node in the clique, and for the PageRank y of the additional node outside it. Then solve this equation.

d) Explain briefly why random teleports not only solve the spider trap problem but also the dead ends problem. In addition to slides, you might read Section 5.1.4 of the book *Mining of Massive Data Sets*[1] to understand random teleports.

Exercise 3 (4 points)

In this exercise you should implement a simple routine which computes the PageRank of a graph using the Google formulation (see slide "The Google Matrix" of the lecture) and the Power Iteration method. To simplify, your code does not need to be scalable, and you are encouraged to use Python and NumPy (or Pandas) for data structures and operations. Submit your code and results of the program runs.

a) Implement a routine which receives as input a (dense) matrix M, a scalar β, and a scalar ϵ, and outputs the PageRank vector of M. To simplify further, you can pre-compute matrix A and store it as a dense matrix, i.e. you don't need to consider the matrix $(1 - \beta)[\frac{1}{N}]_{N \times N}$ as "virtual".

b) Write a routine which generates a matrix $M^{(n)}$ for a clique graph with $n = N$ vertices, for a given $n = N > 1$ (i.e. $M^{(n)}$ is a matrix describing the fully connected part of the graph like in Figure 2). Then create two matrices $M^{(4)}$ and $M^{(6)}$ (i.e. for $n = 4$ and $n = 6$), and execute the routine from a) using $\beta = 0.8$ and $\epsilon = 1/12$. Report the computed PageRank vectors.

[1] *Mining of Massive Data Sets*, available online: http://www.mmds.org/

Exercise 4 (7 points)

In this exercise we will compute the dead–end pages in a subset of the Stanford web graph dataset provided by J. Leskovec *et al.*[2]. The dataset consists of nodes representing pages from Stanford University and directed edges corresponding to hyperlinks between them. In this exercise, only a part of this dataset[3] should be used. The graph (web-Stanford_small.txt) is represented as a tab-delimited text file with the following structure: `<FromNodeID>\t<ToNodeId>` (in addition, there are comment lines).

Note: Submit as solution your source code and results of program runs. You do not need to use Spark (however, you can use Spark if you wish), i.e. scalability of the solution is not required. As the programming language Python is preferred, but if you are more familiar with another common programming language like Java, this is also fine, if your tutor agrees.

a) For each node i in the graph, we need to store a list of nodes that link to i (*in-neighbors* of i) and a list of nodes that i links to (*out-neighbors* of i). Find a reasonable Python data structure to store i and its lists.

b) Compute and output a list of all dead-end pages in the graph. A page is considered as a dead-end page if it has no out-neighbors or all of its out-neighbors are dead-end pages. Assume that we use the recursive dropping method to deal with dead-end pages as described in Section 5.1.4 (*Avoiding Dead Ends*) of the book[5] (i.e. method #1, not the "taxation" method #2). The output list of dead-end pages should be sorted in their removal order. Use an efficient way to retrieve all dead-end pages described in the assignment by Hung Le[4], see FAQ.

Exercise 5 (Bonus, 2 points)

The graph in Figure 3 is a binary tree of n levels with a self-loop at the root node. Recall the dead-ends problem of PageRank. Suppose we recursively eliminate dead-end nodes from the graph in Figure 3 as described in Section 5.1.4 (*Avoiding Dead Ends*) of the book *Mining of Massive Data Sets*[5]. Use the "Flow" model for computing the following parts.

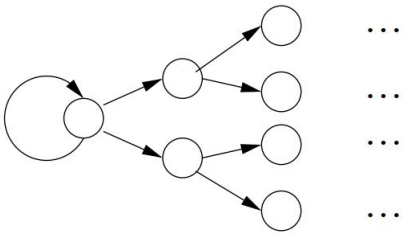

Figure 3: An example graph

a) After removing all the dead-ends nodes, what is the remaining graph? Compute the PageRank for each node in the remaining graph.

b) Estimate the PageRank for the dead-end nodes at i^{th} level. Use the first method in Section 5.1.4. What do you think about the sum of the PageRanks in this case?

[2] J. Leskovec et al., Stanford web graph: http://snap.stanford.edu/data/web-Stanford.html
[3] Available in heiBOX, data/dataset-problemset9-ex4.zip
[4] Hung Le, Programming Assignment for PageRank: https://hunglvosu.github.io/posts/2020/07/PA3/
[5] Available online: http://www.mmds.org/

Exercise 6 **(Bonus, 5 points)**

This exercise is related to processing data with Spark dataframes. Your implementation might be reused in future exercises.

Take a look at a dataset[6] with prices of so-called spot instances from Amazon Elastic Compute Cloud (EC2) service. This dataset contains prices collected for *seven* availability zones, grouped in 28 zipped files. Inside of each compressed file there is a tab-delimited text file with the structure

<Type>\t<Price>\t<Timestamp>\t<InstanceType>\t<ProductDescription>\t<AvailabilityZone>.

a) Implement a subroutine in Python/Spark which takes as an argument a file name of a compressed (*.gz) file with the structure as described above, reads its content into a new Spark dataframe, and returns a reference to this dataframe. To this aim define an appropriate schema with a correct name and (memory-efficient) data type for each column (e.g. use *TimestampType*() and not *StringType*() for column *Timestamp*[7]). The resulting dataframe should **not** contain the column *Type* (as each row has the value "SPOTINSTANCEPRICE"). Submit your code as the solution.

b) To test your implementation, read the file *prices-eu-central-1-2019-05-24.txt.gz* into a dataframe, and then calculate and output the average price per each combination *InstanceType*/*ProductDescription*.

[6] Available on heibox, see: data/dataset-EC2-series/

[7] see also pyspark.sql/data_types