Applications of PageRank and its variations to actors network

Final assignment for Algorithms for Massive Datasets course at Università Statale di Milano 2020/2021

Giulia Pais - Student ID number T15156
August 17, 2021

1 Introduction

Among link analysis algorithms, PageRank is certainly one of the most known and used in a wide variety of scenarios, mostly thanks to variations of the base algorithm that can be used to address different needs. The aim of this project is to implement from scratch the PageRank algorithm, both in the "classic" (original) version and in other variations, and apply it on a network that represents relationships between actors.

1.1 Network structure

The network we're interested in the context of this project, contains data about actors and movies. More precisely, the information can be represented as a labeled property graph with the following characteristics:

- Actors are represented as *nodes*. Nodes are associated with a unique numeric identifier, the name of the person and optionally other relevant information.
- Nodes are connected by an edge if the actors played a role together in at least one movie.
- Edges also possess properties, in particular each edge is labelled with the list of movies in which the actors played a role together.

It is also worth to make an important consideration from a mathematical stand point: the relation ***co-acted*** between two actors is *symmetric* and this translates in the fact that the corresponding graph is *undirected*. Since PageRank works on directed networks, we need to derive a directed graph from the our graph: we can easily do that by simply transforming each original edge in two edges with opposite directions. Unfortunately adding edges may impact negatively on performance, since the symmetry does not apply to the transition matrix, making this property impossible to exploit for optimization purposes.

1.2 An overview on this report

As previously mentioned, in this report we're going to present in detail the process and reasoning behind the algorithm implementation, the experiments performed and finally the results we obtained. Through the course of this dissertation we are going to refer frequently to the Jupyter notebook (`https://github.com/GiuliaPais/PageRank-IMDb/blob/master/PR_for_imdb.ipynb`) containing all the code of the project, which can be found in the dedicated GitHub repository (`https://github.com/GiuliaPais/PageRank-IMDb`).
Section 2 briefly illustrates technologies and tools used in the development of the project, section 3 contains an in-depth explanation on all implemented algorithms and possible optimizations adopted, section 4 presents the test data set used for algorithm validation while in section 5 states all the processing done on the data set and the final structure of actors network. Finally sections 6, 7, 8 and 9 contain respectively the explanations of all the applications of algorithms on data, the produced results, considerations on scalability and finally some closing considerations.

2 Technologies and tools

The project was developed in Python 3 (version 3.7) using a Jupyter notebook fully executable from Google Colaboratory. Code execution was tested in Google Colaboratory with a standard hosted runtime. *Apache Spark* was used for data import, processing, cleaning and algorithm implementation. Other packages were used for data visualization (*plotly*, *jupyter_dash*, *dash_cytoscape*, *dash_bootstrap_components*) and other tasks (*itertools*, *multiprocessing*, *pandas*, *numpy*).

2.1 Spark, DataFrames and GraphFrames

Apache Spark™ is a well-known unified analytics engine for large-scale data processing, compatible with the Hadoop framework but a lot faster. Spark relies on distributed computation in clusters and it is based on the concept of RDD (Resilient Distributed Dataset), a read-only multiset of data items distributed over a cluster of machines, that is maintained in a fault-tolerant way. With the advent of Spark 2.0, Spark SQL was introduced as a new component on top of Spark Core to provide support for structured and semi-structured data and introduced an abstraction over RDD called *DataFrame*. DataFrames are generally faster and optimized for better memory usage, thus it is recommended to use this data structure rather than plain RDD for better performance [8].
Another module of Apache Spark is *GraphX*, a distributed graph-processing framework that fully supports property graphs and features a good number of graph algorithms, which can be efficiently executed in a distributed environment. GraphX is not to be mistaken as a graph database (such as Neo4j), since it is based on RDDs which are immutable, thus making graphs built in this way immutable as well. Since it would be useful to compare the results

of the computation of our algorithms, an immutable property graph with natively implemented PageRank is more than sufficient for our purposes. Unfortunately though, GraphX does not offer a Python API, so we opted for *GraphFrames* [2] instead.
GraphFrames is not a proper component of Spark, it is instead available as a Spark package. The basic principle is the very same of GraphX, but GraphFrames is based on DataFrames rather than RDDs. The package features, among several graph algorithms, PageRank as well, making it a suitable alternative to GraphX.

3 Algorithms

In this section we are going to present more in detail the selected algorithms, how they were implemented and optimization adopted.

3.1 Classic PageRank

PageRank is a way of measuring the importance of website pages. It was originally developed by Larry Page and Sergey Brin at Stanford University in 1996 [1]. It was the first and best known algorithm used by Google to order search queries results, but as of now it is not used for this purpose anymore. The PageRank algorithm outputs a probability distribution used to represent the likelihood that a person randomly clicking on links will arrive at any particular page.
Supposing we have a set of n web pages, we can produce the transition matrix of the network by filling the matrix, for each page i in a column-wise manner, with the value 0 if two pages are not linked or with $\frac{1}{outDegree(i)}$ otherwise. The transition matrix M, if the network is strongly connected and does not have dead ends, is column-wise stochastic and values in each column correspond to probability values. Following the random surfer model presented in the original paper, PageRank can be obtained by calculating the main eigenvector by applying the power method until convergence. Supposing the random surfers have equal probability of starting from any of the nodes, we can initialize the PageRank vector PR_t to $[\frac{1}{N}]_N$ where N is the total number of pages.
PageRank, therefore, can be obtained by iterating until convergence the product between the transition matrix M and the vector PR_t

$$PR_{t+1} = MPR_t$$

To address the issue of spider traps and dead ends, a further correction should be applied to the formula above:

$$PR_{t+1} = \beta MPR_t + (1-\beta)\frac{1}{N}\hat{1}$$

where β is a scalar between 0 and 1 and corresponds to the probability that the random surfer chooses to follow an outgoing link. Consequently, $1-\beta$ is the probability of *teleportation*, that is, the probability that a random surfer chooses to navigate to a random page following a normal probability distribution.

3.1.1 Implementation

The full code for the function is available in the notebook in section 2.1. The algorithm was implemented using DataFrames as base data structures. The function is logically divided in 4 phases:

1. From the input graph (a GraphFrame object) compute the transition matrix (already multiplied by beta)
2. Obtain the initial vector PR_t and separate the connected nodes from the isolated nodes
3. Compute the PageRank for the isolated components and the sum of square differences between the initial vector and current vector (to calculate distance)
4. Apply the power method for connected components and check for exit conditions

The computation of PageRank with this function always terminates since it is possible to set two stop conditions, the maximum number of iterations `n_iter` and the tolerance threshold `tolerance`: at each iteration the function will check if at least one of the 2 conditions is verified and if not it proceeds to the next iteration. Euclidian distance was used to check for convergence at each step. It should be noted that the problem of dangling nodes, in particular isolated nodes, in this implementation has been addressed solely with the *taxation* (teleportation) method, therefore the PageRank vector components might sum up to a value which is less than 1 if isolated nodes are present in the input data.

3.1.2 Optimizations

In the development process a few aspects were taken into consideration for optimizing performance, in particular:

- Avoiding operations that produce shuffling as much as possible and opt instead for windowing operations.
- Avoiding unnecessary operations
- Relying on caching frequently used data (for example the transition matrix)

Optimizing by reducing unnecessary operations
To speed up computation it is crucial to avoid unnecessary workload. One example of this is the computation of PageRank for isolated nodes: this step can be done only once before the iteration. Isolated nodes are those nodes that do not have any in-link or out-link to other nodes (in the test set we introduced node 9 as an example of this scenario): this means that technically speaking they don't appear in the triplet-form transition matrix since all their associated transition values are 0s. Thus, the PageRank value for these nodes solely depends on the second component of the equation, $(1-\beta)\frac{1}{N}\hat{1}$, and does not depend on the previous value of the vector. Along this line of reasoning, since we use Euclidian distance between the PageRank

vectors at time t and $t+1$ to check for convergence, we can calculate the sum of squared differences for isolated nodes only once. Recalling that Euclidian distance between two vectors is calculated as

$$d(v,u) = \sqrt{\sum_{i=1}^{n} |v_i - u_i|^2}$$

we can calculate

$$diff\_isol = \sum_{i=1}^{k} |v_i - u_i|^2$$

supposing we have k isolated nodes, just once before iterating, and

$$diff\_connected = \sum_{i=k+1}^{n} |v_i - u_i|^2$$

for the remaining connected components at each iteration. Finally, the actual distance can be obtained at each iteration with:

$$d(PR_t, PR_{t+1}) = \sqrt{diff\_isol + diff\_connected} = \\ = \sqrt{\sum_{i=1}^{k} |v_i - u_i|^2 + \sum_{i=k+1}^{n} |v_i - u_i|^2}$$

Also note that the distance for isolated components is relevant only for the first iteration, since PageRank remains unchanged for the other iterations, making the distance for isolated components equal to 0.

3.2 Topic-sensitive PageRank

As the name suggests, topic-sensitive PageRank is a variation of the algorithm that, for a set of pre-defined topics, biases the ranking towards pages (nodes) that are about said topics [4]. In particular, for each of the topics, a different PageRank vector is calculated by slightly modifying the original equation.
Suppose we have a a set of j topics, T, then for each $t_j \in T$:

$$PR_{t_j}(t+1)_i = \begin{cases} \beta M PR_{t_j}(t)_i + (1-\beta)\frac{1}{|S|}\hat{1} & \text{if i is in topic } t_j \\ \beta M PR_{t_j}(t)_i & \text{otherwise} \end{cases}$$

where β is the inverse teleportation probability, M is the transition matrix, $PR_{t_j}(t)$ is the PageRank vector at the previous iteration and $|S|$ is the cardinality of the *teleporting set*, which is the set of nodes about topic t_j.
Differently from the classical algorithm, the PageRank vector at step 0 has the same number of components as the number of nodes but this time the values for each component are assigned differently, namely:

$$\begin{cases} \frac{1}{|S|} & \text{if node i is in topic } t_j \\ 0 & \text{otherwise} \end{cases}$$

3.2.1 Implementation

In the context of this project, we can apply the topic-sensitive version of PageRank by considering as topics the different movie genres. The function, `topic_sensitive_page_rank` (full code available in the Jupyter notebook in section 2.2), has an additional argument with respect to the first function, `topics`, where the user can specify either a list of movie genres or `None` for running the classic version of the algorithm. For each of the genres contained in the list, a separate PageRank vector will be calculated independently: at the end of computation the function returns a DataFrame containing nodes and one column of PageRank values for each topic in input.

3.2.2 Optimizations

All optimizations mentioned in section 3.1.2 stand true also for this function. A few additional optimizations were added and are briefly explained in the following paragraphs.

Leveraging parallel computation
Since the computation of vectors for each topic is independent, we can leverage on parallel computation to improve time performance [7]. Thanks to the package `multiprocessing`, we can spawn a number of threads equal to the number of given topics and distribute the workload through the functions `ThreadPool()` and `starmap()`. A schematic example of this reasoning is available in figure 1. It is also worth remembering that even if we have one thread for each topic, true parallelism solely depends on the number of available cores on a machine.
Optimization on individual workers
Each of the parallel workers will execute a function which is a mild modification of the classic PageRank algorithm: if in the classic implementation we could obtain the values in two different ways (for isolated and connected nodes), in the topic-sensitive version we have a total of four different ways, summarised in figure 2.

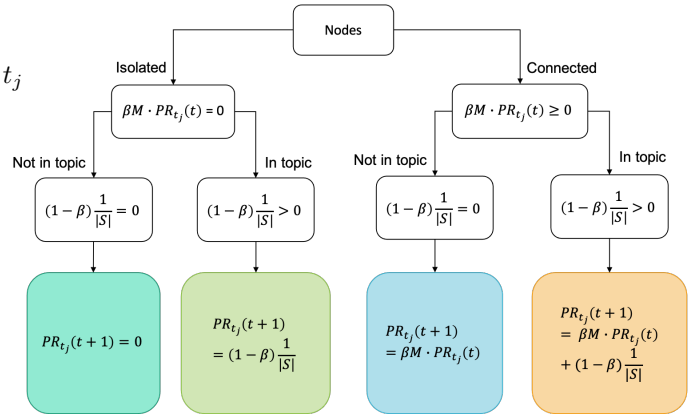

Figure 2: Different ways to compute values for PageRank according to the type of node

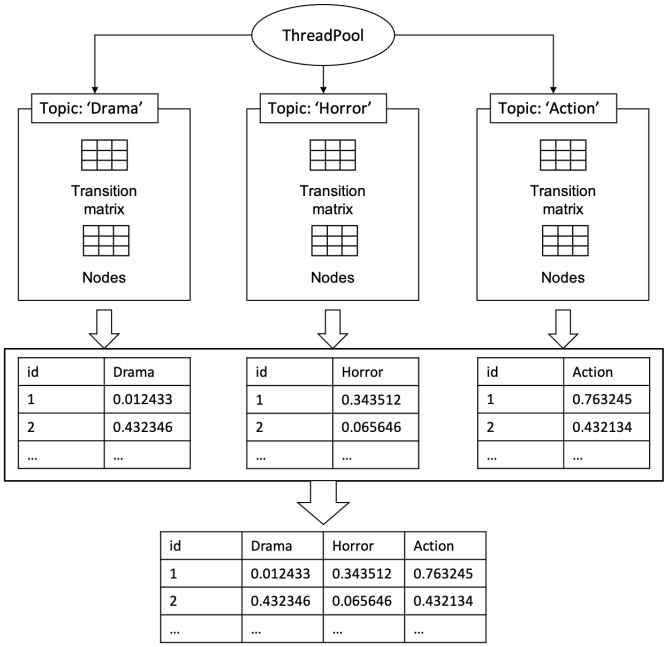

Figure 1: Distribution of workload across threads to compute PageRank for different topics

As for the other function, we can compute rank for isolated nodes just once prior iteration, while we will have to distinguish two possible cases in the iteration phase: if nodes are not in topic, rank will be calculated only from the component $\beta M PR_{t_j}(t)$. This can easily be achieved with a windowing function in combination with a boolean condition.

3.3 Edge-weighted PageRank

One of the most prominent limitations of the original PageRank algorithm is the fact that all edges and nodes are considered to have the same weight, whereas in reality it is useful to have a ranking of nodes in weighted graphs. Several different ways of accounting for weights in the calculation have been proposed [9] [3] [5], based on the type of bias to introduce: ranking can be biased towards specific nodes by assigning weights to them (also referred to as *Personalized PageRank*), towards nodes based on the weights of out-links (or in-links) or both.

3.3.1 Implementation

The version of weighted PageRank we chose to implement in this project biases PageRank values by taking into consideration edge weights only. It is based on the weighted version of PageRank proposed by Ding [3]. In our data, edges are labeled with a list of movie identifiers in which two actors collaborated on: each movie is associated with an individual weight and each edge weight is obtained by summing the weights of all movies in the label.

If i and j are nodes in the actors graph

$$w_{i \to j} = \sum_{k=1}^{M} m_k$$

where $w_{i \to j}$ is the weight of the edge between i and j, M is the number of movies contained in the edge label and m_k is the weight of the movie k.
Analogously to the *out degree* of a node, we can also define the *out weight* of a node, that is the sum of all outgoing edge weights. Given the node i and the set of its adjacent nodes A

$$out\_weight(i) = \sum_{h=1}^{|A|} w_{i \to a_h}$$

where $w_{i \to a_h}$ is the weight of the edge between i and the adjacent node a_h.
In order to modify the original algorithm, we need to change the transition matrix: values in the matrix, always filled column-wise, now will be

$$m_{ij} = \begin{cases} 0 & \text{if i and j are not connected} \\ \frac{w_{i \to j}}{out\_weight(j)} & \text{otherwise} \end{cases}$$

and we will denote this different transition matrix with $M^{'}$. PageRank can now be calculated as

$$PR_{t+1} = \beta M^{'} PR_t + (1 - \beta)\frac{1}{N}\hat{1}$$

Notice that everything except the transition matrix remains unchanged from the original formulation, since we retain the taxation method to account for dangling nodes and we don't change the distribution probability of new surfers, aka they have equal probability to start in each of the nodes in the graph. Once again, if there are isolated nodes in the data, the transition matrix is *sub-stochastic* and the sum of components of the PageRank vector may be lower than 1.
Another thing introduced to ensure the correctness of the algorithm is an argument, `na_politic`, which is used to deal with missing weights. In fact, it is very possible that the associated weight for a movie might not be contained in the data set for any reason, but we don't want the associated edges to have a weight of 0. The argument, therefore, accepts two possible values:

- `'min_value'`: the missing weights will be replaced with the minimal value of weight for all movies
- `'drop'`: movies without an associated weight will be dropped, possibly dropping the associated edge if the label contained only a null-weight movie

First option is recommended to avoid changing the structure of the network.

3.3.2 Optimizations

No further optimizations other than the ones already presented in section 3.1.2.

4 Test data for algorithm validation

Included in the project, there is a minimal data set for algorithm validation purposes only. The data is structured as follows:

- nodes: a data frame with 10 rows and 4 columns (Table 1)
- edges: a data frame with 30 rows and 3 columns (Table 2)
- weights: a data frame with 5 rows and 2 columns (Table 3)

The graphical representation of the test data is shown in figure 3.

id	nconst	primaryName	genres
1	n01	ACTOR1	["Drama", "Romance"]
2	n02	ACTOR2	["Drama", "Crime", "Thriller", "Horror"]
3	n03	ACTOR3	["Crime", "Thriller", "Horror"]
4	n04	ACTOR4	["Crime", "Thriller"]
5	n05	ACTOR5	["Drama", "Romance"]
6	n06	ACTOR6	["Crime", "Thriller", "Horror", "Comedy", "Animation", "Fantasy"]
7	n07	ACTOR7	["Crime", "Thriller"]
8	n08	ACTOR8	["Romance", "Drama", "Comedy", "Animation", "Fantasy"]
9	n09	ACTOR9	["Drama"]
10	n10	ACTOR10	["Comedy", "Animation", "Fantasy"]

Table 1: Test data, nodes data frame

movie_id	weight
m1	43
m2	35
m3	14
m4	6
m5	70

Table 3: Test data, movie weights data frame

src	dst	movie_ids
1	2	["m1"]
2	1	["m1"]
1	5	["m1", "m4"]
5	1	["m1", "m4"]
5	8	["m4"]
8	5	["m4"]
8	10	["m3"]
10	8	["m3"]
2	3	["m2", "m5"]
3	2	["m2", "m5"]
2	6	["m5"]
6	2	["m5"]
6	8	["m3"]
8	6	["m3"]
6	10	["m3"]
10	6	["m3"]
6	3	["m5"]
3	6	["m5"]
3	7	["m2"]
7	3	["m2"]
2	7	["m2"]
7	2	["m2"]
3	4	["m2"]
4	3	["m2"]
4	7	["m2"]
7	4	["m2"]
2	4	["m2"]
4	2	["m2"]
1	8	["m4"]
8	1	["m4"]

Table 2: Test data, edges data frame

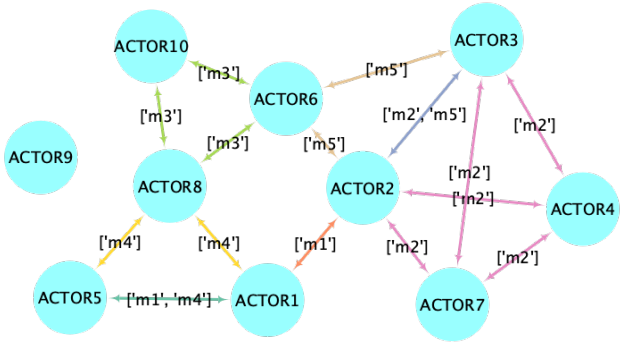

Figure 3: Structure of test data for algorithm validation

5 Data set details and pre-processing

The data set used in the context of this project is the IMDb data set, provided by Kaggle. The data is divided in five tables:

1. names.basics: contains information about people (actors, producers, writers etc.)

2. title.basics: contains information about works of art (movies, shorts, series, books, etc.)

3. title.akas: contains additional information about works of art, for example title translations

4. title.principals: contains information on relationships between people and works of art

5. title.ratings: contains limited information on works user ratings

Only a subset of this data will be retained after a pre-processing phase of data cleaning.

Additional considerations and premises on data Even if a first glance at the the data might suggest the related graph to be strongly connected, we should not take this observation as granted for at least two reasons:

- It is highly probable the data set does not contain all known actors and/or movies, therefore we may encounter disconnected/independent components that are effectively sub-graphs

- There may be movies in which the cast is composed by a single person, alternatively just a single actor out of several in the cast of a movie might be present in the data set. This directly translates in the possibility of having isolated nodes in the graph.

The possibility of having disconnected and isolated components is already accounted for in the PageRank original algorithm through the concept of *teleportation*, but, as previously mentioned, the presence of isolated nodes implies that the transition matrix would be sub-stochastic, therefore PageRank could converge to a value which is lower than 1. With a quick anti-join operation it is easy to detect in our data a considerable percentage of isolated nodes: we decided to exclude such nodes from computations to avoid PageRank converging to a value greatly smaller than 1 and also because in the visualization of results we chose to show the top 100 ranking nodes, making the presence of those isolated nodes effectively irrelevant.

5.1 Actors data

In order to extract relevant information about actors, a filtering operation is performed on the "names.basics" table, more precisely, only rows that contain "actor" or "actress" in the "primaryProfession" attribute are considered. Moreover, only the attributes "tconst" and "primaryName" are considered useful for our purposes, since the first one contains a unique identifier of the person and the second one contains the name for which the person is mostly known for (could also be a stage name). As additional steps, a further column containing a progressive and strictly positive numeric id is added for computational ease, and finally a joining operation with the processed relationship data set (presented in subsection 5.3) allows us to have all the movie genres associated with each actor. After processing, the final data frame has a total of 692,347 rows and 4 columns.

id	nconst	primaryName	genres
1	nm0000001	John Doe	[Drama, Crime]

Table 4: Final structure of the data frame containing actor data after processing (filled with sample data)

5.2 Movies data

For the scope of the project only data referring to *movies* or *tvMovies* is kept through a filtering operation on the "titleType" attribute. After processing, the data frame contains 656,672 rows and 9 columns.

5.3 Relationship data

The table "title.principals" contains information on relationships between actors and works of art, but, as stated in the introduction, our interest is in actor-actor relationships. To obtain such data we need to perform a series of transformation on the table in 2 steps:

1. Filtering only data relevant to actors with an inner-join operation with the table presented in section 5.1

2. Grouping by "tconst" (work identifier) perform a collect-set operation on actors ids: the obtained structure will contain for each work identifier a list of people involved

3. Filter only data relevant to movies with an inner join operation with the table presented in section 5.2

4. For each row of the transformed data "explode" the rows by computing permutations of two items on people list

src	dst	movie_id	movie_genres
1	2	t00432	[Drama, Crime]

Table 5: Intermediate structure of the data frame containing actor-actor relationships (filled with sample data)

This intermediate structure is represented in Table 5 and it is used in joining operation with the names data frame to store movie genres information in the latter. If we stopped here, we would have a *multigraph*, that is, a graph that can have multiple edges between the same pair of nodes. To collapse multiple edges into a single edge we do:

1. Drop the `movie_genres` column

2. Grouping by `src`, collect all movie ids in a list

The structure of the obtained data is represented in Table 6. After processing, the final data frame contains 10,167,884 rows and 3 columns.

src	dst	movie_ids
1	2	[t00432, t00565,...]

Table 6: Final structure of the data frame containing actor-actor relationships (filled with sample data)

5.4 Ratings data

For the computation of weighted PageRank we need to assign a unique weight value to each movie in order to bias the computation. In the provided data set, the table *title.ratings* contains for each work identifier its associated average user rating (a number between 1 and 10) and the total number of votes. What we aim to do is to obtain a single value that can suggest the "importance" or "impact" of a movie based on this two pieces of information by following these general ideas:

- A movie popularity can be determined by the number of votes: the more a movie is voted the more popular (known) it is
- A movie quality can be determined by the average rating given by users: the higher the rating, the higher the quality

Since the amount of information we have is limited, we propose this method for obtaining the *movie impact index*:

- Calculate z-scores for averages rating and number of votes. A z-score is an indicator of how much a single value is distant from the distribution average: for example, a big positive number of votes z-score for a movie m, means that movie was voted a lot more than other movies in the same population, indicating the movie is more popular than the average. The z-score for a value v is calculated as:

  $$z = \frac{v - \mu}{\sigma}$$

  where μ is the average of the population and σ is the standard deviation.
- Correct the z-scores values to avoid having negative or zero values. We can do that by summing the minimum z-score value with opposite sign and add 1.
- Obtain the final index as such:

  $$IMPACT(i) = \alpha z_{avgRating}(i) + \beta z_{numVotes}(i)$$

  where $z_{avgRating}(i)$ is the average rating z-score for movie i, $z_{numVotes}(i)$ is the number of votes z-score for movie i, α and β are two parameters that need to be set appropriately for which holds the relation $\alpha + \beta = 1$. In other words the two modifiers change the weight of each component (popularity/quality) in the index. In our case we used the values $\alpha = 0.40$ and $\beta = 0.60$, which means the impact of the movie would depend slightly more on its popularity rather than its quality.

In the end we obtain a data frame with the structure shown in Table 7.

movie_id	weight
t00432	6.3421

Table 7: Final structure of the data frame containing movie weights (filled with sample data)

A disclaimer is necessary: the proposed metric is an oversimplified one, with the sole scope to serve as a functional example in the project. In a realistic scenario, a lot more factors would contribute to the actual impact of a movie, metacritics score, lifetime gross, number of articles and posts mentioning the title just to name a few.

6 Experiments

In this section we explain how the algorithms were applied to different data.

6.1 Algorithm validation

Validation of algorithms is necessary to ensure both correctness of our approach, identification of potential issues or errors and possible performance bottlenecks. For this purpose we use the previously introduced test data set.

6.1.1 Classic PageRank

The PageRank algorithm implemented from scratch for this project, `page_rank()`, was validated against values produced by the algorithm implemented by GraphFrames (GraphX), `GraphFrame.pageRank()`. It worth mentioning that the GraphFrames algorithm checks for convergence using a different distance measure, namely the Chebyshev distance [6]. Experiment details:

- The function `page_rank()` was called on the test data set, with arguments `n_iter = 10` and `tolerance = 10e-6`
- The function `GraphFrame.pageRank()` was called on the test data set, with arguments `maxIter = 10`

6.1.2 Topic-sensitive PageRank

Unfortunately, GraphFrames does not implement a version of topic-sensitive PageRank, thus direct comparison of results is impossible in this case. Thus, to validate the algorithm, we will rely on basic expectations on our test data. We will run the function `topic_sensitive_page_rank` setting arguments as follows:

- `topics = ['Drama', 'Thriller']`
- `n_iter = 10`
- `tolerance = 10e-6`
- `beta = 0.85`

We will compare the results for each topic to the classic PageRank score: broadly speaking what we expect to observe is a higher value of rank for those nodes in the given topics with respect to the classic rank and a lower value for all other nodes. More precisely, for our test data we expect:

- For topic 'Drama'
  - Rank should be higher for nodes 1, 2, 5, 8, 9
- For topic 'Thriller'
  - Rank should be higher for nodes 2, 3, 4, 6, 7
  - Rank should be 0 for node 9 (isolated and not in topic)

6.1.3 Edge-weighted PageRank

As for topic-sensitive PageRank, we need to validate the algorithm against basic expectations rather than real values, since GraphFrames does not provide an implementation of edge-weighted PageRank. What we expect to see, in general, is a bias towards those nodes that have a high *in-weight*, which depends both on the number of in-links and on the weight of each in-link. Since, as stated in the introduction, our graph is technically undirected and the relation is symmetric, in-weight and out-weight for each node are equal, so for our test data we expect node ranks to be ordered as Table 8.

id	out_weight
2	288
3	245
6	168
4	105
7	105
1	98
5	55
8	40
10	28
9	0

Table 8: Nodes out-degrees, ordered from highest to lower

6.2 Applications on actors network

The code in the Jupyter notebook in section 5 is user-interactible and can be customized according to user preferences. As a test, we ran the algorithms with the following arguments:

- `n_iter = 15`
- `tolerance = 10e-4`
- `beta = 0.85`
- `na_politic = 'min_value'` (for weighted PR)
- `topics` = ['Action', 'Horror', 'Drama', 'Comedy'] (for topic-sensitive PR)

Configuration of the SparkSession for the IMDb data set was initialized with these options:

- "spark.sql.shuffle.partitions" = "100"
- "spark.sql.autoBroadcastJoinThreshold" = "-1"

7 Results

7.1 Algorithm validation

Results of the algorithm validation for classical PageRank show approximately the same trend (shown in figure 4). We can thus consider our algorithm reliable in term of results.

Figure 4: Comparison between PageRank values obtained with GraphFrames algorithm and custom implemented algorithm. Results obtained from the GraphFrames algorithm were corrected by dividing results by the number of nodes.

For topic-sensitive PageRank, comparison between the classical PageRank values and topic-biased values meets the expectations presented in section 6.1.2 with the exception of node 2 for topic "Drama", as can be seen in figure 5. This discrepancy may be due to a cumulative lower contribution of surrounding nodes and doesn't show any intrinsic problems to the algorithm since the distribution of values follows a predictable trend: rank is higher for those nodes in topic (with a peak always on node 2) and lower for nodes not in topic (with a minimum on node 9).

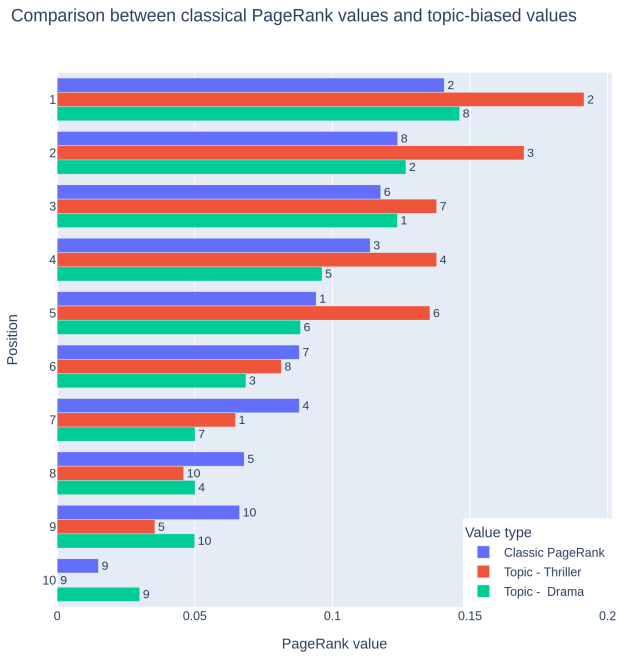

Figure 5: Comparison between classic PageRank values and topic-biased rank values.

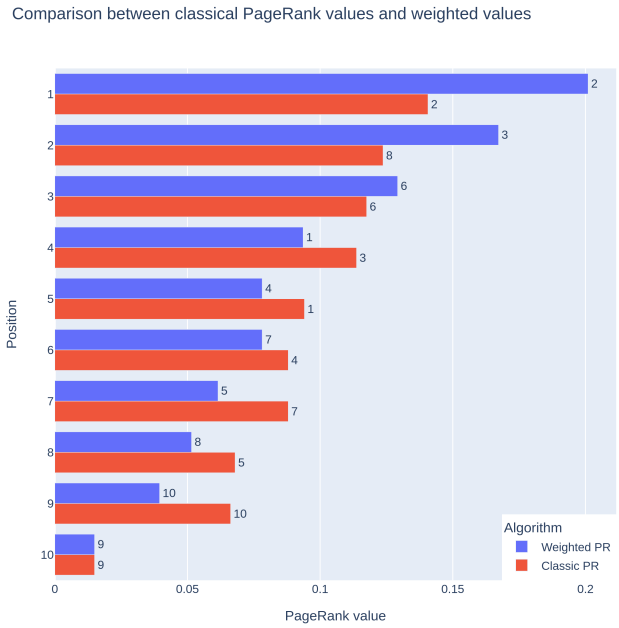

Figure 6: Comparison between classic PageRank values and weighted values. The results are ranked from highest to lower position. The label on the right of each bar is the node id the rank value corresponds to.

Results for the weighted PageRank validation are shown in Figure 6. The ranking shows weighted values can change the order of top ranked nodes and that it corresponds to the expectations presented in section 6.1.3 with the exception of node 1 which occupies a higher position in the ordering, likely due to the dominant contribution of node 2.

7.2 Applications on actors network

For each algorithm, the top 100 ranking nodes are extracted for visualization on the interactive dashboard in the Jupyter notebook in section 6. The dashboard contains a visualization of the sub-network (an example is shown in Figure 7) and a visual barplot of the rankings. It is possible to switch between different algorithms with the buttons located in the top portion.

8 A few considerations on scalability

All the proposed implementations of the different algorithms, due to the usage of the Spark framework, can manage scaling-up to bigger data sets horizontally, aka, by adding nodes to the cluster. Moreover, considerations on scaling up were taken in the development phase by avoiding the explicit use of broadcast joins which, however potentially faster, do have an hard-coded size limit for variables to be broadcasted (8 GB). A careful configuration of SparkSession is mandatory to avoid potential OutOfMemory errors and performance bottlenecks, in particular:

- shuffle partitions might need to be increased
- individual executor memory limit might need to be increased
- particular attention should be paid to data skweness and number of partitions since they can drastically decrease performance if not set properly
- disable broadcast joins entirely by setting "spark.sql.autoBroadcastJoinThreshold" to "-1"

9 Conclusion

In this project we showed how to implement different versions of the PageRank algorithm using Spark and how these can be applied not only to web pages but to generalized networks, in our case to movie actors network, to find nodes with central roles.

References

[1] S. Brin and L. Page. "The Anatomy of a Large-Scale Hypertextual Web Search Engine". In: Seventh International World-Wide Web Conference (WWW 1998). Brisbane, Australia, 1998. URL: `http://ilpubs.stanford.edu:8090/361/` (visited on 08/01/2021).

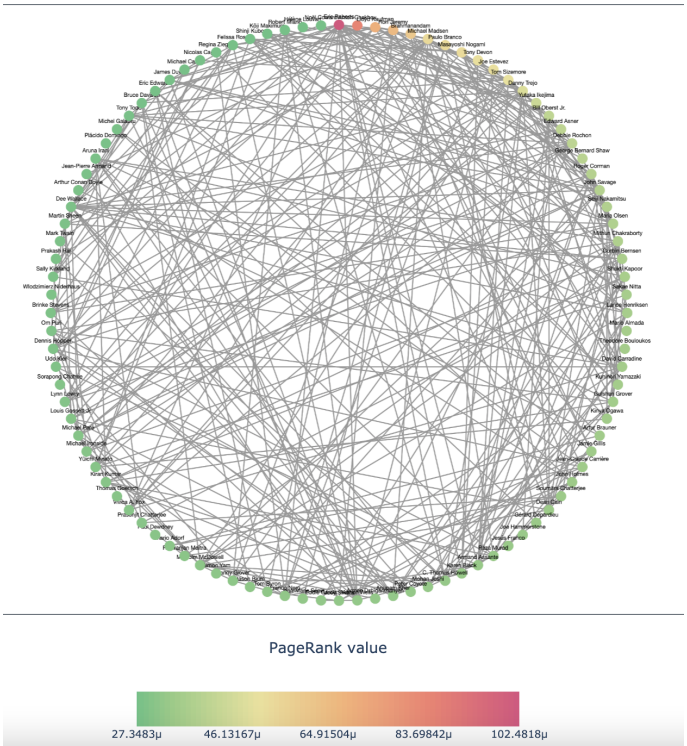

Figure 7: Screenshot of the Cytoscape canvas responsible for network visualization in the interactive dashboard.

[2] Ankur Dave et al. "GraphFrames: an integrated API for mixing graph and relational queries". In: *Proceedings of the Fourth International Workshop on Graph Data Management Experiences and Systems - GRADES '16*. the Fourth International Workshop. Redwood Shores, California: ACM Press, 2016, pp. 1–8. ISBN: 978-1-4503-4780-8. DOI: 10.1145/2960414.2960416. URL: http://dl.acm.org/citation.cfm?doid=2960414.2960416 (visited on 08/17/2021).

[3] Ying Ding. "Applying weighted PageRank to author citation networks". In: *arXiv:1102.1760 [cs]* (Feb. 8, 2011). arXiv: 1102.1760. URL: http://arxiv.org/abs/1102.1760 (visited on 08/17/2021).

[4] T.H. Haveliwala. "Topic-sensitive pagerank: A context-sensitive ranking algorithm for web search". In: *IEEE Transactions on Knowledge and Data Engineering* 15.4 (July 2003), pp. 784–796. ISSN: 1041-4347. DOI: 10.1109/TKDE.2003.1208999. URL: http://ieeexplore.ieee.org/document/1208999/ (visited on 08/17/2021).

[5] Xiaoming Liu et al. "Co-authorship networks in the digital library research community". In: *Inf. Process. Manag.* (2005). DOI: 10.1016/j.ipm.2005.03.012.

[6] *PageRank (Spark 2.1.3 JavaDoc)*. URL: https://spark.apache.org/docs/2.1.3/api/java/org/apache/spark/graphx/lib/PageRank.html (visited on 08/04/2021).

[7] somanath sankaran somanath. *Horizontal Parallelism with Pyspark*. Analytics Vidhya. Jan. 14, 2020. URL: https://medium.com/analytics-vidhya/horizontal-parallelism-with-pyspark-d05390aa1df5 (visited on 08/06/2021).

[8] *Spark Performance Tuning & Best Practices — SparkByExamples*. Spark by {Examples}. Aug. 1, 2020. URL: https://sparkbyexamples.com/spark/spark-performance-tuning/ (visited on 07/29/2021).

[9] Panpan Zhang, Tiandong Wang, and Jun Yan. "PageRank centrality and algorithms for weighted, directed networks with applications to World Input-Output Tables". In: *arXiv:2104.02764 [physics, stat]* (May 15, 2021). arXiv: 2104.02764. URL: http://arxiv.org/abs/2104.02764 (visited on 08/17/2021).

I declare that this material, which I now submit for assessment, is entirely my own work and has not been taken from the work of others, save and to the extent that such work has been cited and acknowledged within the text of my work. I understand that plagiarism, collusion, and copying are grave and serious offences in the university and accept the penalties that would be imposed should I engage in plagiarism, collusion or copying. This assignment, or any part of it, has not been previously submitted by me or any other person for assessment on this or any other course of study.