Nicola Dall'Asen 211662 Elena Zordan 211663

Big Data Technologies project

Erdős Graphwalker

Big data system that calculates the coauthorship graph and the analogous of the Erdős number for any given scholar

Erdős number have been a part of the folklore of mathematicians throughout the world for many years. It measures the distance between Paul Erdős and any other person, allowing to describe sort of "collaborative distance" among authors. It works by counting the number of hops needed to connect the author of a paper with the prolific late Paul Erdős: an author's Erdős number is equal to one whether it has directly co-authored a paper with him (Erdős was an Hungarian mathematician who wrote papers with a total of 509 coauthors, meaning 509 have Erdős number 1).
The system implemented allows to calculate the coauthorship graph for any given scholar, showing him/herself "Erdős number" with respect to another author. Publication data have been retrieved from Google Scholar.

The system is composed by 6 dockerized services, Neo4j as a graph databases as it naturally servers our purposes and matches our data; Redis as a queue service and caching level; two consumers which, extracting from the queues in Redis, obtain what they have to analyze and, by scraping Google Scholar, populate the database and recursively search for the authors; Flask API which provides an interface to requests the coauthorship graph with the distance information and, finally, an interface written in React to display graphically the results. More details are provided in the next section.

Services

Google Scholar does not provide any public API to obtain data about scholars, their publications and coauthors, so data must be scraped from Google Scholar directly. We decided to scrape data on two different levels: authors can provide a list of coauthors

directly on their page, and this can be easily and fast scraped but it is clearly incomplete, since not every coauthor will be listed and some people could decide to not put any person in it.
On the other hand, extracting information from every publication is a slow process leading to the most complete graph possible. This is why we scrape on two levels, to provide a fast response at first but slowly generating a complete graph in the background.

Graph generation starts from the first request, when the author will not be available in our system. The two consumers will start populating the database enriching the data available. A subsequent request of a non-present author will receive the highest priority possible in order to be analyzed as soon as possible and thus providing a fast answer to the user.
Going into details we analyze the various services.

Redis

It is used to provided two priorities queues, one for the short term consumer (direct coauthorship) and one for the long term one (publications-based analysis). We used Redis also a caching layer, storing the coauthorship graph in order to not having to recompute it every time the person is requested. The key has an expiration date, in case data change on the database, e.g. new coauthorships are added or new publications are written.
Finally, its fast key set and retrieval are used to provide a mechanism to understand if a certain author has to be analyzed or should be skipped because his/her data were already downloaded.

Short Term Consumer

Written in Python because of the support of libraries like Requests and BeautifulSoup, it consumes the data present in the queue on Redis and analyzes the coauthorship list available on the profile page on Google Scholar, and it enqueues every coauthor to be analyzed later on.

Long Term Consumer

Written in Python because it uses Selenium with headless Chromium, it simulates the human clicking activities on the web to click and obtain data on publications of an author. A solution with BeautifulSoup was not feasible. It consumes data from the other queue on Redis and, similarly to STC, enqueues back every coauthor to analyze.

API

Written in Flask, they provide two endpoints; the first allows to search for an author, which relies on a support library written by us that both searches on Google Scholar and on Neo4j to find people without Google ID but with similar name. These latter people were discovered by analyzing the publications but don't have a Google Scholar profile.
The second one, by providing the Google ID or the node ID, allows to return the coauthorship graph. Thanks to the use of Redis as a caching system the graph for every person needs to be calculated only the first time by performing a BFS, possible thanks to Cypher, the query language of Neo4j; subsequent requests of the same author will be server from Redis, and thus greatly speeding up the response time.

React

It provides the interface to interact with the system and it is composed by three pages. The first is a search page, similar to Google's one. Once the query is sent, results both from Google Scholar (thus having a Google ID) and from our Neo4j instance (only those without a Google ID are returned) are displayed.
Finally, after selecting the desired profile is selected, his/her graph and coauthor list with distances are displayed. If the selected profile is not available in our system it is enqueued with maximum priority, and the user is invited to come back after some minutes, in order to let STC and LTC to collect the data.

Deploy

In order to provide an easy-to-transfer and applicable solution we started from the very beginning to use dockerized version of the application, dividing the whole program into microservices to foster scalability. A CI environment was setup on GitLab to build and push to the GitLab registry our images, so to provide a ready-to-go way to use our app. The live version is hosted on Google Cloud Platform through the use of *docker-compose.* True scalability would be achieved by the use of services like Kubernetes and having already dockerized every microservice it could be achieved without much effort.

Results

Given the source of our data, it was not possible to bulk download the data and make our local copy of Google Scholar, the graph is built over-time *slowly,* requests for a given author can come at any time thus the response will always be incomplete and this is why we set an expiry date for cached data on Redis, in order to allow re-analyzing and re-computing.
Erdős number, being based on a BFS on the graph, is an extremely computing-heavy algorithm, with time complexity O(|V| + |E|), and this requires quite powerful machines, e.g. calculating the graph for *Daniele Miorandi* on GCP with 16 vCPU and 60GB of RAM took about 8 minutes, leading to a graph of 13650 nodes within distance 5.

Issues

From the very beginning of the process, the data retrieval, we faced some issues due to the lack of Google Scholar API's. At first, we implemented our web scraping through the BeautifulSoup Python library (*aragog.py*) which allowed us to extract useful information (like name, id, affiliation, coauthors) of the searched users profiles. However, this tended to an uncompleted graph and a distorted version of the reality. Extracting publications information has been the most proving part, as Google easily detects bots trying to access these data. We then tried to rely on external libraries like *scholarly.py* but, by requesting a large amount of data, we were detected and banned multiple times.
The solution was to write our own scraper through the use of Selenium, so simulating the human activity and not accessing directly the publications' pages. Part of the solution was to find a balance between our need for data and the speed at which Google allows us to scrape them, requests need to be slowed down and delayed in order to not bomb Google servers and get banned.
Finally, using Apache Spark with GraphX library would have provided an easy way to scale horizontally the computation but the difficulty to setup a Spark cluster, with the fact that a standard PC with 8GB of RAM would have hardly sustained the whole system to test it "offline" (Databricks Community Edition would have been insufficient with its 16GB of RAM too) led us to rely on standard BFS provided by Neo4j, which, in its Enterprise edition, provides a scalable solution.

Conclusion

Overall, the project scope was quite ambitious since it had to calculate the Erdős number and the coauthorship graph (arbitrary choice of "jumps", which we set equal to 5) of any author, even the ones without a Google Scholar profile.

While scraping Google Scholar some interesting aspects came up to our mind. Whereas the long time consumer takes into account all the possible relationships between authors, the short time consumer considers only those coauthors which were actually chosen by the author him/herself. In order to achieve comprehensive results, the Erdős number and the coauthorship graph are calculated from both consumers. However, the reason behind this choice goes beyond the purposes of completeness: the fact that an author can actually choose the coauthors list shown -or even *if* showing this list- might lead to a distorted image of the subject. Social Psychology and Behavioural Studies discovered that several effects can arise when people know they or their profile will be observed. Actually, when constructing the *self,* many aspects from both the outside (like the other people we choose to be friends with, the social-cultural environment, the roles we assume, etc) and the inside (self-image, self-esteem and ideal-self) affect the process[1]. Long story short, the personally chosen coauthors list might have been affected by the *confirmation bias* and overall this might lead to a *social desirability bias*. The former is defined as "the tendency to look for or interpret information to confirm our existing beliefs"[2], meaning that the author might have chosen to show those coauthors whom him/her share the most in common with (actually the sole act of the selection involves some changes). On the other hand, the latter states that people share the "tendency to report an answer in a way they deem to be more socially acceptable, if they believe are under observation, than would be their true answer"[3]. While at first glance this seems unlikely to happen in the academic context, in the last year it has be shown that all types of "social media are particularly affected by social desirability bias because people manage their presence online in order to generate a positive self-image"[4].

It would be interesting to delve more deeply into these effects from a psychological point of view in order to find out how and why authors choose and build their own network.

Finally, our project is nothing but a starting point for more complex and interactive way of showing networks among people. Further steps must be taken, ideally with respect to the possibility of filtering results according to the distance (given by the Erdős number) between authors.

---

[1] **D. MYERS J. TWENGE**, *Psicologia Sociale*, McGraw-Hill Education, Milano, 2013, p. 59

[2] *Ibidem,* p. 123

[3] **G. VELTRI**, *Digital Social Research*, Polity Press, Cambridge, 2020, p. 36

[4] *Ibidem, p.36*

References

- MYERS D.G. TWENGE J., *Psicologia Sociale*, translated by MARTA E., LANZ M., McGraw-Hill Education, Milano, 2013 (original ed. *Social Psychology*, McGraw-Hill Education, 2013)
- VELTRI G.A., *Digital Social Research*, Polity Press, Cambridge, 2020