Building a transfer graph for the evolution of pages in Wikipedia

Richard Zhi-Wei Chou (6998364)
r.z.chou@students.uu.nl
Utrecht University
Utrecht, The Netherlands

KEYWORDS

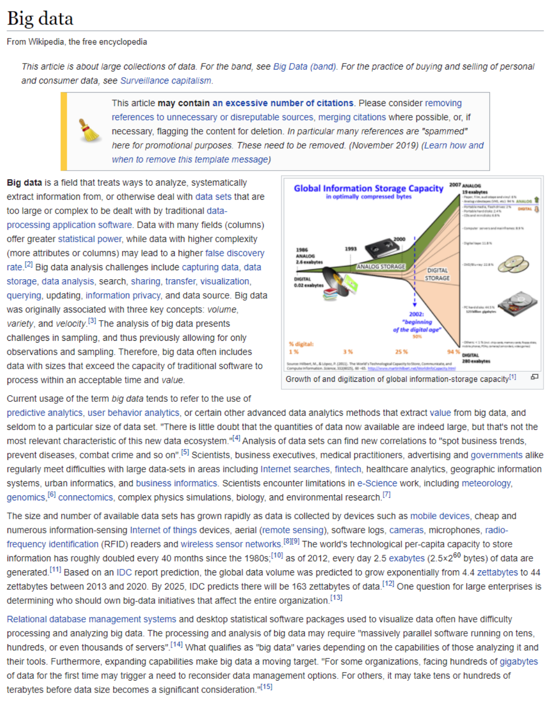
Data Intensive Systems - Wikipedia - Spark - Similarity graph - Neo4J - MapReduce

ABSTRACT

This report addresses the problem of modelling the relationship between sets of page histories using a similarity graph. The problem that will be addressed is: what kind of changes have been made throughout the different versions of the sets? And how is it different in comparison to the previous versions of the sets? For this assignment, the page histories with the related references are generated and transformed into a dataframe. Followed up by a Spark program that processes the dataset and calculates the Jaccard distance to showcase the similarity between the sets. The result will be that the data is transformed into a temporal similarity graph. This is to display the different nodes as page versions with the page similarity as the edges between the nodes.

1 INTRODUCTION

Since the launch of Wikipedia in 2001, the free encyclopedia is grown to one of the world's biggest and most valuable knowledge sources. As of 2020, the size of the amount of Wikipedia articles is counted up to 52 million on Wikipedia [4]. Every Wikipedia page contains a title, different sections, texts, and links or references to other pages. When all the text is removed, then a page would only contain the references or the links to the different pages. These links can be referred to as entities because they are referring to other Wikipedia subjects. In Figure 1, there is an example provided about the topic *Big Data*.


Big data

From Wikipedia, the free encyclopedia

This article is about large collections of data. For the band, see Big Data (band). For the practice of buying and selling of personal and consumer data, see Surveillance capitalism.

This article **may contain an excessive number of citations**. Please consider removing references to unnecessary or disreputable sources, merging citations where possible, or, if necessary, flagging the content for deletion. *In particular many references are "spammed" here for promotional purposes. These need to be removed. (November 2019) (Learn how and when to remove this template message)*

Big data is a field that treats ways to analyze, systematically extract information from, or otherwise deal with data sets that are too large or complex to be dealt with by traditional data-processing application software. Data with many fields (columns) offer greater statistical power, while data with higher complexity (more attributes or columns) may lead to a higher false discovery rate.[2] Big data analysis challenges include capturing data, data storage, data analysis, search, sharing, transfer, visualization, querying, updating, information privacy, and data source. Big data was originally associated with three key concepts: *volume*, *variety*, and *velocity*.[3] The analysis of big data presents challenges in sampling, and thus previously allowing for only observations and sampling. Therefore, big data often includes data with sizes that exceed the capacity of traditional software to process within an acceptable time and *value*.

Global Information Storage Capacity
in optimally compressed bytes
1986 ANALOG 2.6 exabytes
DIGITAL 0.02 exabytes
1993
2000
2007 ANALOG 19 exabytes
ANALOG STORAGE
DIGITAL STORAGE
2002: "beginning of the digital age" 50%
% digital: 1 % 3 % 25 % 94 %
DIGITAL 280 exabytes

Growth of and digitization of global information-storage capacity[1]

Current usage of the term *big data* tends to refer to the use of predictive analytics, user behavior analytics, or certain other advanced data analytics methods that extract value from big data, and seldom to a particular size of data set. "There is little doubt that the quantities of data now available are indeed large, but that's not the most relevant characteristic of this new data ecosystem."[4] Analysis of data sets can find new correlations to "spot business trends, prevent diseases, combat crime and so on".[5] Scientists, business executives, medical practitioners, advertising and governments alike regularly meet difficulties with large data-sets in areas including Internet searches, fintech, healthcare analytics, geographic information systems, urban informatics, and business informatics. Scientists encounter limitations in e-Science work, including meteorology, genomics,[6] connectomics, complex physics simulations, biology, and environmental research.[7]

The size and number of available data sets has grown rapidly as data is collected by devices such as mobile devices, cheap and numerous information-sensing Internet of things devices, aerial (remote sensing), software logs, cameras, microphones, radio-frequency identification (RFID) readers and wireless sensor networks.[8][9] The world's technological per-capita capacity to store information has roughly doubled every 40 months since the 1980s;[10] as of 2012, every day 2.5 exabytes (2.5×2^{60} bytes) of data are generated.[11] Based on an IDC report prediction, the global data volume was predicted to grow exponentially from 4.4 zettabytes to 44 zettabytes between 2013 and 2020. By 2025, IDC predicts there will be 163 zettabytes of data.[12] One question for large enterprises is determining who should own big-data initiatives that affect the entire organization.[13]

Relational database management systems and desktop statistical software packages used to visualize data often have difficulty processing and analyzing big data. The processing and analysis of big data may require "massively parallel software running on tens, hundreds, or even thousands of servers".[14] What qualifies as "big data" varies depending on the capabilities of those analyzing it and their tools. Furthermore, expanding capabilities make big data a moving target. "For some organizations, facing hundreds of gigabytes of data for the first time may trigger a need to reconsider data management options. For others, it may take tens or hundreds of terabytes before data size becomes a significant consideration."[15]

Figure 1: Excerpt of the Wikipedia page Big Data, displaying the text and links to other pages

1.1 Problem Statement

But to measure the similarity between the different edits made over time is considered a problem in data mining and data retrieval applications. According to Bollobas et al. [1], in most of the data mining applications, it is necessary to search within a database for series that are similar to the provided query. But when edits are made by multiple people over time, then this will create different baselines, outliers and/or different scaling factors. This will lead to deficiencies when measuring the similarities for two or more sets of pages, which is undesirable for getting the right output.

The aim of this report is to develop a scalable algorithm to control the page edits over n time. This with the purpose to measure the similarity of a set of pages when the different pages are updated by several users. A set is defined as a Wikipedia topic containing all the page versions made over time. The sets are measured by using the Jaccard distance, see Figure 2 for the equation. For the two sets, the equation is defined by the cardinality of the intersection divided by the cardinality of the union. The A stands for the set of pages A and the B stands for the set of pages B.

$$\mathcal{J}(A,B) = 1 - \frac{|A \cap B|}{|A \cup B|}$$

Figure 2: Jaccard Distance

When applying the Jaccard distance on the two sets, the entities will be paired based on the intersections of entities on the pages. In this case, when comparing the similarity of the sets, the numerator is then the number of entities intersecting over time in the different page versions within the set, and the denominator is the total number of entities of the different versions over time. The output shows the percentage of how similar each entity are between these sets. A is set one and B is set two where $_n$ stands for the page edits of the applicable set that are done over time. This will result in the following formula in Figure 3 (where X represents the entities of set A_n, while Y represents the entities of set B_n):

$$\mathcal{J}(A_n, B_n) = 1 - \frac{|X \cap Y|}{|X \cup Y|}$$

Figure 3: Jaccard Distance for 2 Wikipedia pages

The Jaccard distance is chosen, because of the accessible appliance for MapReduce and Spark. MapReduce is a technique to handle large sets of data. Which in this case are the different page versions within the sets. Spark is a fast cluster computing system for processing both batch and real-time data processing. These systems and the applicable solutions are explained later on in the report (Section 3: Solution). For this project, it is chosen to apply the Spark solution to the generated dataset. But in the end, the solution should be accessible and reproducible for other programmers who are interested in calculating the Jaccard distance between different sets of Wikipedia pages.

The report is structured as follows. Section 2 frames how the dataset is generated. Section 3 describes the solution, where the MapReduce and Spark approaches are discussed. And finally, in Section 4, the result of the Spark solution is modelled in a similarity graph and described how the approach was done.

2 DATASETS

2.1 Generating Dataset

The dataset is generated by using the *Wikipedia-histories* library from Nathan Drezner. This library allows the user to scrape data from the applicable subject with all the previous page versions and the content made in the past.

For this report, it is chosen for six topics of the subcategory within the Business Intelligence subject of Wikipedia: Business intelligence, Competitive intelligence, Data mining, Market intelligence, Open-source intelligence and Product intelligence. Not all the subcategories could be generated due to the computation time of generating the pages and their previous versions. That is why it is decided to select six topics out of the eleven topics classified as a subcategory for the topic Business intelligence.

Business intelligence (BI) is defined by Negash and Gray [6] as "a data-driven decision support system that combines data storage, data gathering, and knowledge management with analysis to provide input to the decision process." As reported by Calof and Wright [2], Competitive intelligence (CI) is defined as the "system of environmental scanning which integrates the knowledge of everyone in the company."

Widom [8] describes a Data warehouse as that it "encompasses architectures, algorithms, and tools for bringing together selected data from multiple databases or other information sources into a single repository, which is suitable for direct querying or analysis." Marketing intelligence (MI) is defined by Jamil [3] as a "process designed to constantly produce knowledge for business sectors from dispersed data and information for strategic market positioning, as an organisational continuum that aims to answer typical decision problems faced by firms when competing in actual business environments."

Column name	Data type	Description
revid	Integer	Page revision ID
time	Datetime	Date and time of edits/creation
text	Varchar	Content of the page

Table 1: Data types of the dataframes

According to Steele [7], Open-source Intelligence (OI) is defined as "unclassified information that has been deliberately discovered, discriminated, distilled, and disseminated to a select audience to address a specific question." McFarlane et al. [5] describe Product Intelligence (PI) in an industrial context as "the linking of a physical order or product to information and rules governing the way it is intended to be made, stored or transported."

The information is scraped from these pages and filtered into three columns: revid (stands for revision ID), time, and text. In Table 1, a short description is provided and the applicable data types are described.

The dataframes were saved into several variables. Because Product intelligence was only edited 17 times, the last 17 edits were filtered from the topics BI, CI, Data warehouse, OI, and MI. When the data were filtered to these page versions and subsetted to these columns, the dataframes were merged into one master dataframe (stored as *master_df.csv*). This master dataframe contains 104 rows distributed over the three columns mentioned in Table 1.

3 SOLUTION

In this section, the two solutions (Spark dataframes and MapReduce) will be discussed in both ways, and for this report, it is chosen to make a temporal similarity graph out of a Spark dataframe.

3.1 MapReduce

For computing the set similarity when using MapReduce on the two sets of pages, it is required to count the number of links on each individual page A_n and then followed up by how similar it is with the links of the set of individual pages, B_n (links are represented in elements X and Y). The links from element Y that do not overlap with X should be subtracted. In the end, the number of entities will be divided by the amount elements which will result in the coefficient of how similar the sets are.

This is a step-by-step how the algorithm will run within a MapReduce manner:

Map: XY

The map method takes a key/value pair (X, Y) and returns these values unchanged in the algorithm. It will result in the following:

X, Y

Reduce: Y

The follow-up is the Reduce function, which takes the key and the set of all the values, and then computes the cardinality of values of set Z. This returns the input key with each input value and Z.

This value will be used to compute the denominator of the Jaccard similarity.

X, (Y, Z)

Map: Switch (XY -> YX)

The next step is to take the key/value pairs, and then switch the strings between the key and the value. This will return the modified value and key of the pages. When using the Y element as the key, then this will result in combining the stages to collect all the X elements that correspond to each element of Y.

Y, (X, Z)

Reduce: Split

This reduce task continues with taking the key and the set of values, following up by splitting the set of values. This will output an empty key with each of the reduced lists of values (mapped as the value Q). The purpose is to load the balance of the collection of pairs in the last map method.

Q, {(X, Z)}

Map: Collect the pairs

This map takes in an empty key Q and the previously mapped list of values. Furthermore, it creates a series of X_i, X_j pairs with the first element in the list of values. This is also done in each of the following elements. The list of size n will produce n - 1 pairs. Except for the first one, one pair for each element in the list.

$(X_i, X_j), C_ij\ (C_ij = C_i + C_j)$

Reduce: Normalise the occurrences

Finally, this reduce will calculate the number of occurrences of X_i and X_j by counting the number of elements in the set of input values. This will be followed by dividing the numbers of occurrences by the total number of Y elements that occur with one or both of X_i and X_j.

$(X_i, X_j), |\{C_ij\}| / C_ij - |\{C_ij\}|$

Hereby the pseudocode of the solution (referring to algorithm 1).

After applying this algorithm on multiple pages and their previous pages (A_n, B_n), based on the output, a temporal similarity graph can be created based on the Jaccard distance (1 - Jaccard similarity) output of how similar each page within the set are with each other. The pseudocode for this solution is displayed in the "*Jaccard distance - MapReduce*" (referring to Algorithm 1).

3.2 Spark

For this project, it is chosen to make use of the Spark dataframes. For the Spark solution, the setup is slightly different. This solution has been approached by loading in the CSV file outputted from the generated dataset and convert it to a Spark dataframe. After converting it to a Spark dataframe, a function is created and used

Algorithm 1: Jaccard distance - MapReduce

```
map(String key, String value) // key: set_pages1_links //
  value: set_pages2_links
      for each set_pages1_link in key
      return: (set_pages1_links, set_pages2_links);

reduce(String key, Iterator values): // key: set_pages1_links
  // values: set_pages2_links
      for each set_pages1_links in key:
          count of links += Count(set_pages2_links);
      return: set_pages1_links, (set_pages2_links, count of
  links);

map_2(String key, String value) // key: set_pages1_links //
  value: set_pages2_links, count of links
      for each link in key
      return: set_pages2_links, (set_pages1_links, count of
  links);

reduce_2(String key, Iterator values): // key:
  set_pages2_links // values: set_pages1_links, count of links
      for each set_pages2_links in values:
      return: empty set, {(set_pages1_links, count of links)};

map_3(String key, String value) // key: empty set // value:
  set_pages1_links, count of links
      for each empty set in key
      return: (set_pages1_$links_i$, set_pages1_$links_j$), count
  of $links_{ij}$;

reduce_3(String key, Iterator values): // key:
  (set_pages1_$links_i$, set_pages1_$links_j$) // values: count of
  links$_{ij}$
      for each link in key:
          result += ParseInt(count of links$_{ij}$);
      return: set_pages1_$links_i$, set_pages1_$links_j$, results;
```

Algorithm 2: Jaccard distance - Spark

```
function pipeline (sparkdf);
Input   : sparkdf
Output : sparkdf with distCol (Jaccard distance column)
start:
  create model =
          stage 1: HashingTF_(sparkdf)
          stage 2: MinHash_LSH(sparkdf)

return: model

start:
        dfhashed = transform_model(sparkdf)
        dfmatches = calculate Jaccard distance

return: dfmatches (sparkdf with distCol);
```

Jaccard distance	Similarity level	Representing set
0.6 - 1	Similar	323
0.5	Somewhat similar	2448
0 - 0.4	Not similar	2585

Table 2: Pages similarity at different cutoffs

to calculate the Jaccard distance based on the text-similarity from the set of pages.

The pipeline function is using the *HashingTF* function from the PySpark library. The *HashingTF* maps a sequence of terms to the term frequencies using the hashing trick. Another function that has been applied for the pipeline function is by using the *MinHashLSH* function, which is also from the PySpark library. The *MinHashLSH* function inputs vectors, and then it calculates the Jaccard distance for the vectors. By combining the functions, it makes it possible to compute matches only once for each page, so that the cost of computation is growing linearly instead of exponentially.

The applied solution is constructed in the pseudocode "*Jaccard distance - Spark*" (referring to Algorithm 2). It is describing how the pipeline function works for calculating the Jaccard distance in the context of Spark.

Based on the newly created dataframe, it is possible to calculate how similar the pages are with each other. A 1 is representing a similarity between the sets of pages, and a 0 represents the dissimilarity of the sets of pages. When applying different cutoffs in the Spark solution, it will result that 323 pages of the different sets are similar (values between 0.6 - 1) to each other, 2448 pages of the different sets are somewhat similar (0.5), and that 2585 pages of the different sets are not similar at all (values between 0 - 0.4), see Table 2 and Figure 4.

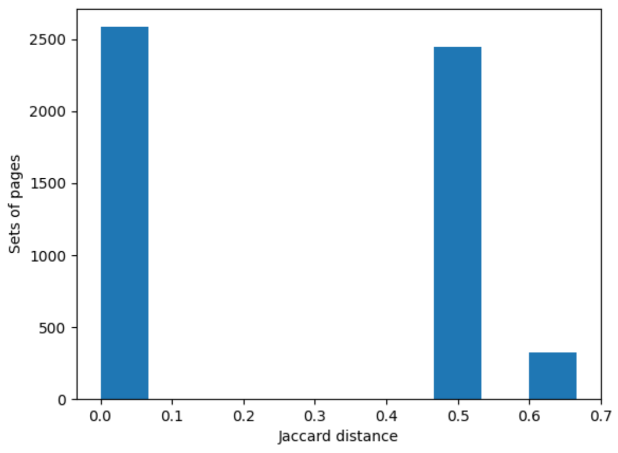


Figure 4: Jaccard distance column distribution

This concludes the Spark solution that has been applied for this assignment. For the source code, it can be found in the "*SRC*" folder.

4 STORAGE

4.1 Temporal Similarity Graph

Based on the Spark dataframe solution, the following graph has been modelled within Neo4J, see Figure 5. The nodes represent the page IDs for each edit, while the edges have the values of the distance column stored in them.

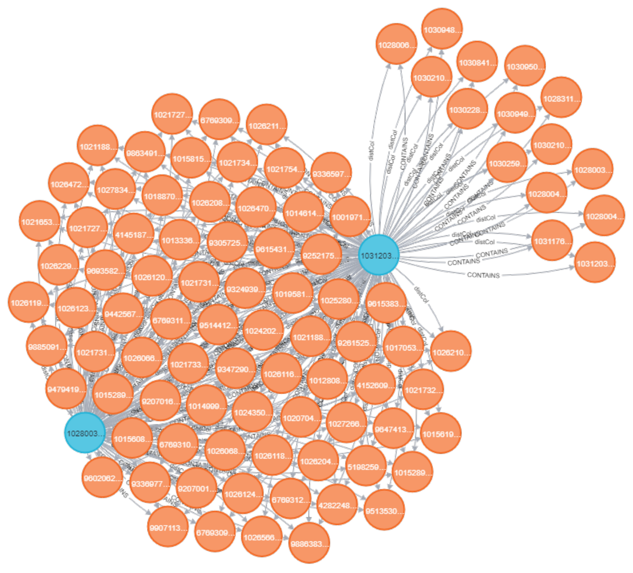
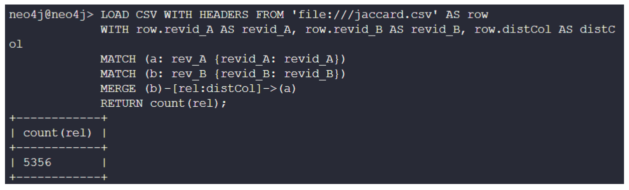


Figure 5: Neo4J graph

The graph is modelled via the Cypher query language. The query for the graph of Figure 5 can be seen in Figure 6. The graph is limited to 150 nodes. The reason behind this is to display the distinction between the similar nodes and the vast majority of nodes that are somewhat similar and are not similar at all.



```
neo4j@neo4j> LOAD CSV WITH HEADERS FROM 'file:///jaccard.csv' AS row
             WITH row.revid_A AS revid_A, row.revid_B AS revid_B, row.distCol AS distC
ol
             MATCH (a: rev_A {revid_A: revid_A})
             MATCH (b: rev_B {revid_B: revid_B})
             MERGE (b)-[rel:distCol]->(a)
             RETURN count(rel);
+------------+
| count(rel) |
+------------+
| 5356       |
+------------+
```

Figure 6: Cypher query for Neo4J graph

4.2 Loading

The loading was done via the CSV file that was outputted from the PySpark pipeline function. This data was loaded within the Neo4j environment and stored as a separate project. Then the data was modified via the query language and linked when applying the Cypher query mentioned in Figure 6. The relationship is stored as the edge "CONTAINS". This is how the data was loaded within the Neo4j environment.

REFERENCES

[1] Bela Bollobas, Gautam Das, Dimitrios Gunopulos, and Heikki Mannila. 1997. Time-series similarity problems and well-separated geometric sets. In *Proceedings of the thirteenth annual symposium on Computational geometry*. 454–456.

[2] Jonathan L Calof and Sheila Wright. 2008. Competitive intelligence. *European Journal of marketing* (2008).

[3] George Leal Jamil. 2013. Approaching Market Intelligence concept through a case analysis: Continuous knowledge for marketing strategic management and its complementarity to competitive intelligence. *Procedia Technology* 9 (2013), 463–472.

[4] Włodzimierz Lewoniewski, Krzysztof Węcel, and Witold Abramowicz. 2020. Modeling Popularity and Reliability of Sources in Multilingual Wikipedia. *Information* 11, 5 (2020), 263.

[5] Duncan McFarlane, Vaggelis Giannikas, Alex CY Wong, and Mark Harrison. 2013. Product intelligence in industrial control: Theory and practice. *Annual Reviews in Control* 37, 1 (2013), 69–88.

[6] Solomon Negash and Paul Gray. 2008. Business intelligence. In *Handbook on decision support systems 2*. Springer, 175–193.

[7] Robert David Steele. 2007. Open source intelligence. *Handbook of intelligence studies* 42, 5 (2007), 129–147.

[8] Jennifer Widom. 1995. Research problems in data warehousing. In *Proceedings of the fourth international conference on Information and knowledge management*. 25–30.