On building a transfer graph for the evolution of pages in Wikipedia

César Di Meglio, 2042789
Utrecht University
c.t.dimeglio@students.uu.nl

ACM Reference Format:
César Di Meglio, 2042789. 2021. On building a transfer graph for the evolution of pages in Wikipedia. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

The velocity aspect of Big data leads to evolution and change over time in the information considered. Keeping track of those changes contributes into a better understanding of how data behaves with respect context throughout time. In this project, we measure the similarity of pairs of Wikipedia pages. If we consider one Wikipedia page, measuring its similarity to other pages can be an indicator of a trend. For instance, if we consider the page "Big data", modifications are made to it over time. These modifications will increase or decrease its similarity to other pages. At one time step, "Big data" could be more similar to "Apache Spark" than "SQL". This would mean that "Apache Spark" plays a larger role in "Big data" than "SQL" at the given timestep. The main challenge is to match the timesteps of our data. That is to say, we need to compare pages that have existed together at the same point in time. Aggregating these comparisons may serve as a representative estimate of how two pages relate to one another given a time frame (section 2.2.3).
How can we measure the temporal similarity of pages in time ? The latter will be problematized in section 2, solutions will be presented in section 4.

2 PROBLEM STATEMENT

2.1 Wikipedia pages as entities

We denote W as a set of Wikipedia pages, referred to as "entities". Each entity $w \in W$ is updated over time by Wikipedia users, we call each of those updates *versions*. For each entity, we only consider the *references* to other Wikipedia pages with respect to its versions. For instance, the entity "Utrecht" will have hyperlinks pointing to other pages like "The Netherlands" or "Amsterdam" given one of its versions.
Some pages are more or less regularly updated, we denote $\mathcal{N}(w) \in \mathbb{N}^*$ as the total number of versions since the creation of entity w on Wikipedia. More formally, we have the following:

Let w_i be a version of w at a timestep i.

$$\forall w \in W,\ \forall i \in \{1, ..\mathcal{N}(w)\},\ w_i = \{r_1, r_2, ..\} \tag{1}$$

where r_j are the *references* (to other Wikipedia pages) present in w_i. The release *date* of version w_i is noted $\tau(w_i)$. There are no versions of w that where released at the exact same date (Eq.2), these chronologically ordered (Eq.3).

$$\forall (i, j) \in \{1, ..\mathcal{N}(w)\}^2 \mid \tau(w_i) \neq \tau(w_j) \tag{2}$$

$$\tau(w_1) < \tau(w_2) < ... < \tau(w_{\mathcal{N}(w)}) \tag{3}$$

where the relation "<" here means that the version on the left was released prior to the version on the right. The version $w_{\mathcal{N}(w)}$ is the latest one, that is to say, the one being displayed when searching w on Wikipedia. From one version to another, *references* can either be added, removed or kept the same. Let w_i and w_j be two versions of w such that $\tau(w_i) < \tau(w_j)$, $(i, j) \in \{1, ..\mathcal{N}(w)\}$
Either one of these three cases can happen:

$$\begin{cases} w_i = w_j & \text{no new references were added from } i \text{ to } j \\ w_i \subseteq w_j & \text{new references were added from } i \text{ to } j \\ w_i \nsubseteq w_j & \text{references were deleted from } i \text{ to } j \end{cases} \tag{4}$$

Note that in the third case of (4): $w_i \nsubseteq w_j$, new references could also have been added into version w_j.

2.2 Measuring the similarity of entities

2.2.1 Definition: Measuring similarity. :
We denote $\mu : W \times W \rightarrow [0, 1]$ a function that computes the similarity of two Wikipedia distinct [1] pages w and w'. Let $\epsilon \in]0, 1[$ be the *similarity threshold*. We define the following property:

$$(w, w') \in W^2,\ \mu(w, w') \geq \epsilon \iff w \text{ and } w' \text{are similar} \tag{5}$$

2.2.2 Similarity metric for two versions. :
The similarity metric chosen to compare two versions of two Wikipedia pages is the *Jaccard similarity*:

$$\mu(w_i, w'_j) = \frac{\sharp(w_i \cap w'_j)}{\sharp(w_i \cup w'_j)} \tag{6}$$

It consists of the ratio of the cardinalities of the intersection and the union of w_i and w'_j. The latter measures the similarity in between the sets w_j and w'_j which contain references to other Wikipedia pages. This is a commonly used metric for measuring set similarity. It is also *commutative* i.e: $\mu(w_i, w'_i) = \mu(w'_i, w_i)$. If we had chosen a metric without this property, then we would have needed to make twice as much comparisons. Moreover, it fulfills the properties of the function evoked to in section 2.2.1.

[1] We neglect cases were $w = w'$ as property (5) always holds

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

2.2.3 Similarity metric for two entities. :
In the scope of this project, we want to compare two Wikipedia pages given their respective histories. For the latter, the versions of the entities are computed using the metric defined in 2.2.2. We say that a version of an entity is *displayed*, if it is the one that appears when the entity is searched on Wikipedia at a given point in time. The challenge is that we must compare versions of entities that were *displayed* at the same time.

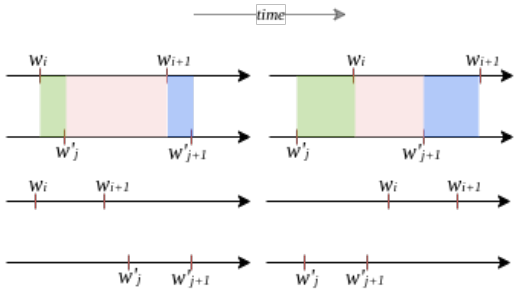

Figure 1: Overlapping configurations for w_i and w'_j

Let w and w' be two entities we want to compare given their respective histories (see Fig. 1). A version w_i is *displayed* from $\tau(w_i)$ included to $\tau(w_{i+1})$ excluded, $i \in \{1, ..., \mathcal{N}(w)-1\}$. In Fig.1, the two examples at the top (with the colors) exhibit configurations where versions can be compared because they are *displayed* together at a point in time. Hence, for these examples, the following comparisons can be made :

$$\begin{cases} \text{Top Left} : \mu(w_i, w'_j), \mu(w'_j, w_{i+1}), \mu(w_{i+1}, w'_{j+1}) \\ \text{Top Right} : \mu(w'_j, w_i), \mu(w_i, w'_{j+1}), \mu(w'_{j+1}, w_{i+1}) \end{cases} \tag{7}$$

In the case where w_{i+2} and w'_{j+2} are released before w'_{j+1} and w_{i+1} in the Top Left and Top right configurations respectively, the suited comparisons are made accordingly.

The two configurations (without colors) at the bottom illustrate the scenarios of time intervals where no comparison is possible since the versions are not *displayed* at the same time. Let $\aleph$ (Bottom Left) and $\aleph'$ (Bottom Right) be the sets of pairs that describe these configurations:

$$\begin{cases} \aleph = \{(w_i, w'_j) \mid (\tau(w_i) < \tau(w'_{j+1})) \wedge (\tau(w_{i+1}) < \tau(w'_j))\} \\ \aleph' = \{(w_i, w'_j) \mid (\tau(w_i) > \tau(w'_{j+1})) \wedge (\tau(w_{i+1}) > \tau(w'_j))\} \\ \text{with } (i, j) \in \{1, ..., \mathcal{N}(w)-1\} \times \{1, ..., \mathcal{N}(w')-1\} \end{cases} \tag{8}$$

If we omit these cases for all timesteps, then we can retrieve all of the pairs (w_i, w'_j) of w and w' that were once *displayed* at a same point in time. This set is noted $\Theta(w, w')$. Hence, we want to compare such pairs of versions of w and w':

$$\Theta(w, w') = \{(w_i, w'_j) \mid (w_i, w'_j) \notin \aleph \cup \aleph'\} \tag{9}$$

with $(i, j) \in \{1, ..., \mathcal{N}(w)\} \times \{1, ..., \mathcal{N}(w')\}$
Lastly, we need to aggregate the similarities of all of the "legal" comparisons to be made between the versions of w and w'. For this, we take the mean of the similarities of each pairs of versions, noted $M(w, w')$.

$$M(w, w') = \frac{1}{\sharp\Theta(w, w')} \sum_{(w_i, w'_j) \in \Theta(w, w')} \mu(w_i, w'_j) \tag{10}$$

Note that M still fulfills the properties mentioned in section 2.2.2, and especially the property 5 for two entities w and w'. Our metric (Eq.6) can vary for two versions of a page. We then chose to take the mean (Eq.10) to have a representative estimate of the similarity of pairs of entities throughout time.

3 [REAL] DATASET

In total, 30 entities were chosen arbitrarily. They all refer to either domains/sub-domains of Science (e.g Theoretical Physics, Topology..) or to famous scientists (e.g Marie Curie). For each **`entity`**, the following procedure was performed:
- The links (to the previous versions) are fetched from the HTML code of the page:
https://en.wikipedia.org/w/index.php?title=**`entity`**&action=history
using the API "Beautiful Soup".
The number of previous versions to be shown at once on this page can be set by tweaking the link above. We chose the maximum amount (approximately 5100). From one page to another, the number of previous existing versions can vary greatly with regard to how "famous" it is. For instance, Japanese DJ "Nujabes" only has a handful number of versions compared to the actor "Keanu Reeves". In this respect, we split the notoriety of the entities into three categories:

- Low entities: pages with a history of less than 1000 versions
- Medium entities: pages with a history of 1000 to 5000 versions
- High entities: pages with a history of more than 5000 versions

The challenge is that for pages that are updated on a very regular basis (which are usually "High entities"), all of the versions available from the link above will span from now to at most three years (approximately) in the past. Whereas the versions from "Low entities" and "Medium entities" will span from now up to several years back in time (up to the very first version of the page).
For Low entities, we chose one version per year. For Medium entities, we chose two versions per year. For High entities, we chose 5 version per year.

- For each version that we select, the names of the references to other Wikipedia pages are collected (Eq.1). As a result, for a single version, we obtain a vector of such a form:

$$[date, r_1, r_2, ...] \tag{11}$$

The versions ("vectors") are then stored in separate files with respect to the entity they belong to.

4 SOLUTION

4.1 Map-Reduce

The Map Reduce solution Alg.1 makes use of two procedures (functions) : `Map` and `Reduce`. The `Map` function, takes the dataset and maps "allowed" version pairs (Eq.9), to a unique `Id` with respect to distinct pairs of entities. Next, the `Reduce` function is in charge of computing the mean Jaccard similarity (Eq. 10) of two entities by regrouping the value pairs by their appropriate keys (`Ids`). It outputs pairs of entities whose mean Jaccard similarity is at least ϵ (Eq.5) with their similarity (Eq.10).

Algorithm 1 Jaccard Map Reduce

1: **procedure** MAP(D)
2: ▷ Only **distinct** pairs of entities are considered because of the commutativity of our metric
3: **for** $(A, B) \in \{S \in \mathcal{P}(D) | \sharp S = 2\}$ **do**
4: `Id` ← Assign unique a unique key to every (A,B) pair
5: **for** $v \in list(\{v \in B\})$] **do** ▷ Iterate over versions of A
6: **for** $v' \in list(\{v \in B\})$ **do** ▷ Iterate over versions of B
7: **if** $(v, v') \in \Theta(A, B)$ **then**
8: `Emit`(`Id`, pair(v, v'))
9: **end if**
10: **end for**
11: **end for**
12: **end for**
13: **end procedure**
14: **procedure** REDUCE(`Id`, pairs: $[p_1, p_2..,]$,ϵ)
15: Sum ← 0
16: **for** `Id` in $list$(`Id`) **do** ▷ $list$(`Id`): list of the keys `Map` created
17: **for** $p \in P_{\texttt{Id}}$ **do** ▷ $P_{\texttt{Id}}$: set of pairs of with the same `Id`
18: Jaccard = $\mu(p)$ ▷ (Eq.6)
19: Sum ← Sum + Jaccard
20: **end for**
21: M ← $\frac{1}{\sharp P_{\texttt{Id}}}$·Sum
22: **if** M≥ ϵ **then**
23: `Emit`(`Id`, M)
24: **end if**
25: **end for** ▷ (Eq.10)
26: **end procedure**

4.2 Spark

The Spark solution Alg.2 accepts the data D and a threshold ϵ. After empirical analysis, we chose $\epsilon = 0.015$ as the threshold (Prop.5). This rather low value was intentionally chosen to be able to show as much relevant relationships as possible. Generally, entities that can intuitively be considered as similar like "Database" and "Big data" have a relatively low Jaccard similarity (under 0.4 from empirical testing).

We start by defining an empty `spark.DataFrame`: *Result*. This dataframe has three columns (or "headers"): EntityA, EntityB and "Jaccard similarity". Here as well, only distinct pairs (EntityA, EntityB) of entities are considered. For every of those pairs we compute $\Theta(A, B)$, the versions which we are "allowed" to compare from EntityA and EntityB (Eq.9). The computation of such a set is done using Resilient Distributed Datasets (RDDs), hence the `RDD`$\langle . \rangle$ operator at line 6. This set is then converted into a `spark.DataFrame`: DF with headers : EntityA and EntityB. For each of the version pairs of those entities (i.e for each row, line 7), the Jaccard similarity (Eq.6) of EntityA and EntityB is computed and stored in a new column for DF : "Jaccard" (line 8). The mean (Eq.10) Jaccard similarity of those entities is then computed. If it is a least greater than the desired threshold $\epsilon = 0.015$ (Prop.5), then a row is concatenated to *Result* by matching its schema, names of entity A and B, as well as their Jaccard similarity.

Algorithm 2 Jaccard Spark

1: **procedure** JACCARD CALCULATOR(D,ϵ)
2: P ←{S ∈ $\mathcal{P}(D)$| ♯$S = 2$} ▷ Set of pairs from dataset D
3: *Result* ← `spark.DataFrame`($\emptyset$) ▷ Empty Dataframe
4: **for** $(A, B) \in P$ **do**
5: Θ ← `RDD`$\langle\Theta(A, B)\rangle$
6: DF ← `spark.DataFrame`(Θ)
7: **for** row in DF **do** ▷ headers: [EntityA,EntityB]
8: DF[Jaccard] ← $\mu(DF$[EntityA], DF[EntityB])
9: **end for**
10: M← $M(DF$[Jaccard]) ▷ Mean of Jaccard column
11: **if** M ≥ ϵ **then**
12: *Result* = *Result* || Row(A, B, M)
13: **end if**
14: **return** *Result*
15:

5 LOADING

5.1 Loading

The resulting data from Alg.2 was loaded into Neo4J Desktop. First, an index named "Node" was created on the names of the entities. The nodes were loaded into Neo4j with the `Nodes.csv` file were the index was applied for every entity. The relationships were then loaded with the `Links.csv` file. We then queried Neo4j to match the nodes that have a link and to show the value of their Jaccard similarity on the edges. This was realized using the "APOC" plugin for Neo4j databases. Please note that no attention needs to be payed at the direction of the edges since our metric is commutative. Moreover, Neo4j does not support undirected edges.

5.2 Nodes with the highest similarity

The following Cypher query was used to find the pair of entities with the highest similarity:

```
1. MATCH p=()-[r]->()
2. WITH p, r.score AS score
3. ORDER BY score DESC
4. LIMIT 1
5. RETURN nodes(p) AS nodes
```

We first select the relations and their value (lines 1 and 2), then order them in decreasing order (line 3). Next, we limit the number of relationships values to be shown to one (line 4). The corresponding nodes to this relationship are returned (line 5). We obtain the output shown in Fig.2

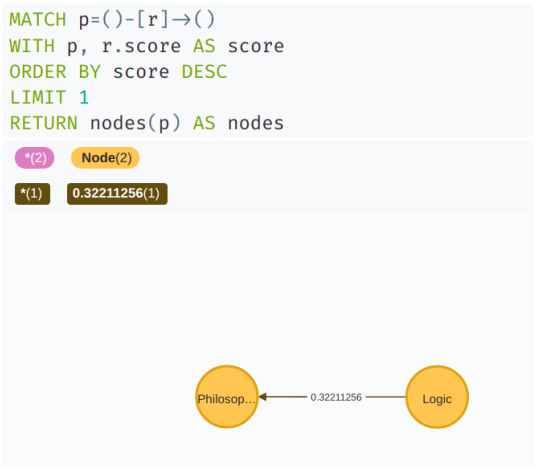

Figure 2: Pair of nodes with the highest similarity