Computer Scientists Retrieval

WIR Project

Students:

- Luca Tomei
- Daniele Iacomini
- Andrea Aurizi

Directed by:

Andrea Vitaletti

Luca Becchetti

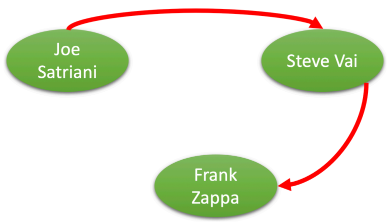
Project Repository

A.Y. 2019-2020

Summary

The paper we chose presents a method to find the most influential rock guitarist by applying Google's PageRank algorithm to information extracted from Wikipedia articles. The influence of a guitarist is computed by considering the number of guitarists citing him/her as influence.

Basically, the experiment consists of building a directed graph where nodes are rock guitarists. There is an outgoing edge from guitarist A to another guitarist B, if guitarist A is influenced by guitarist B.



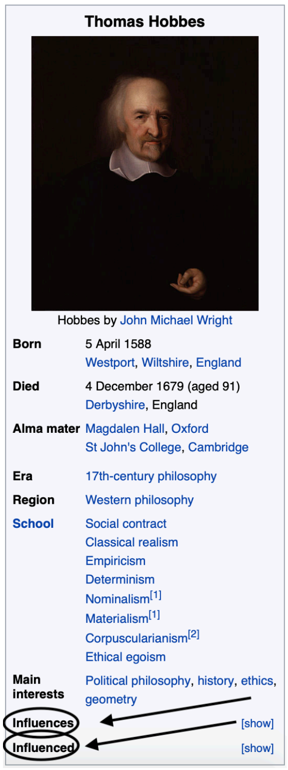
Joe Satriani is influenced by Steve Vai and the latter by Frank Zappa

We decided to replicate the same experiment with the same methodology but choosing computer scientists as the field of study.

Contents

1 Introduction

In this project we focused on the analysis of data concerning a different category from that of guitarists: computer scientists. Unlike a guitarist or a philosopher, a computer engineer does not have much relevant data on wikipedia and there is also no information regarding his school of thought or the influences he had during his life. In fact, the first difficulty encountered during the preliminary phase of the project was precisely to try to create a one-to-one correspondence between two computer scientists, which is not always possible.



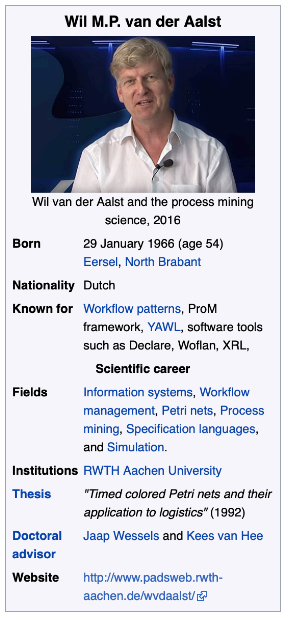


Figure 1: Poet vs Computer Scientist

The previous figures show the data fields on the WikiPedia "infobox biography vcard" tables. The difference in the number of fields between a philosopher and a computer scientist is clear and the latter also has no "Influences" and "Influenced" type fields that we can draw on to create connections between one computer scientist and another.
Therefore, to overcome this problem, we first decided to use *SPARQL* to explore and extract the information contained in RDF graphs from a knowledge base distributed on *DBpedia*.
Subsequently it was decided to carry out a further verification of the results by going directly to Wikipedia since it is undoubtedly one of the greatest resources of the Web for all knowledge.

2 Query DBpedia with SPARQL

To query DBpedia it was decided to use *SPARQLWrapper*, wrapper with the aim of creating the URI of the query and possibly converting the result into a more manageable format.

```
"""Query DBpedia with SPARQL Code"""
from SPARQLWrapper import SPARQLWrapper, JSON

sparql = SPARQLWrapper("http://dbpedia.org/sparql")
sparql.setQuery("""
  PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
  SELECT *
  WHERE {
    ?p a
    <http://dbpedia.org/ontology/Computer_scientists> .
    ?p <http://dbpedia.org/ontology/influenced> ?influenced.
  }
   """)
sparql.setReturnFormat(JSON)
results = sparql.query().convert()
```

Listing 1: Query DBpedia

With the data stored in this way, the creation of the graph is simple: it is sufficient to process each row (each consists of two items: influenced and influence) and insert a corresponding arc in the graph.
After encapsulating the query output in the variable `result` we are ready to examine the data obtained and to create a graph to calculate the pagerank.

```
G = nx.DiGraph()

for result in results["results"]["bindings"]:
  try:
    cs = result["p"]["value"].split("/resource/")[1]
    influenced = result["influenced"]["value"].split("/resource/")[1]

    G.add_edge(influenced, cs)
  except: print("",end="")

""" Computing Pagerank """
pr = nx.pagerank(G, alpha=0.85)
sorted_pr = sorted(pr.items(), key=operator.itemgetter(1))
sorted_pr.reverse()

for i in sorted_pr: print(i)
```

Listing 2: Computing Pagerank

3 Manual scraping Wikipedia

DBpedia offers few results for Computer Scientists, which is why it was decided to take a more demanding path but that would give us more results: Web Scarping.
It was decided to divide this process into 4 logical phases:

1. Collect in a file all the links of computer scientists present in the English version of Wikipedia (*List Of Computer Scientists*).
2. For each link of a computer scientist obtained from the previous phase, it was checked whether the relative web page contained an information table that included the list of influences and influences of this computer scientist.
3. From the results obtained, the Pagerank was calculated with the aim of quantifying the importance of the relative computer scientist within the set of related documents.
4. It was decided to dwell on the classification of the various fields of study of a computer scientist.

3.1 First Phase: Collect data

The English version of Wikipedia contains a list of all the computer scientists (*link*) with an existing article, alphabetically sorted. In the first phase, we simply copy each link and its associated name in a .json file.
The total number of computer scientists retrieved is 509.

3.2 Second Phase: Check informations in Biographic Table

Before starting to extrapolate the information of each computer scientist, for a matter of code optimization and efficiency in terms of performance it was decided to download the entire Wikipedia pages of each of them, so as not to make $n\_cs = 509$ requests.
After collecting the web pages of each computer scientist, for each of them it was checked whether the corresponding page contained an information table (*Infobox* HTML) which included the list of influences and influencers of the said computer scientist.

The number of Computer Scientists who meet this requirement is 62.

```
def bio_table(self, page):
    name = page.rsplit('/')[-1]
    page = open(page, 'r')
    soup = BeautifulSoup(page, "html.parser")
    table = soup.find('table', class_='infobox biography vcard')
    try:  influencers = table.find_all('ul', class_='NavContent')[0]
    except: influencers = []
    try:  influenced = table.find_all('ul', class_='NavContent')[1]
    except: influenced = []
    final_influencers, final_influenced = ([] for i in range(2))
    if influencers != []:
        for a in influencers.find_all('a'):
          final_influencers.append(a.get('title'))
    if influenced != []:
        for a in influenced.find_all('a'):
          final_influenced.append(a.get('title'))
```

Listing 3: Checking Bio Table Function

3.3 Third Phase: Second Approach and Pagerank

Since the result obtained from the second phase was not at all satisfactory, it was decided to use a second approach: instead of checking only the biography table of a computer scientist, the entire page associated with him was analyzed.

```
def make_links(self, path):
  # Define output dict
  inlinks = SortedDict()
  outlinks = SortedDict()

  SetofNames = SortedSet()

  #reading all the folders from the path and creating a set of CS names
  for name in self.read_names(path):
    if name == "Guy_L._Steele,_Jr": name = "Guy_L._Steele,_Jr."

    SetofNames.add(name)

    #creating an empty inlinks of names as sortedSet
    inlinks[name] = SortedSet()

  #reading their inlinks and outlinks
  for name in SetofNames:
    SetOfInLinks = SortedSet()
    fp = open(path + "/"+ name,'r',encoding = "utf-8")
    soup = BeautifulSoup(fp.read(),"html.parser")
    linksFound = []
    linksFound = soup.findAll('a', href=re.compile("/wiki/"))

    HTML = ""
    for link in linksFound:
      HTML = HTML + str(link)
      HTML = HTML + " and "

    #get All the outlinks by calling get_links
    outlinks[name] = self.get_links(SetofNames,HTML)

    if name in outlinks[name]:  outlinks[name].remove(name)

    for outlink in outlinks[name]:
      SetOfInLinks.add(name)
      inlinks[outlink].update(SetOfInLinks)
  return (inlinks,outlinks)
```

Listing 4: Making Links

The previous function, after reading the html pages of each computer scientist as input, creates and returns two *SortedDict*:

- *inlinks*: maps from a name to a SortedSet of names that link to it.
- *outlinks*: maps from a name to a SortedSet of names that it links to.

For example:

- `inlinks['Ada_Lovelace'] = SortedSet(['Charles_Babbage', 'David_Gelernter'], key=None, load=1000)`
- `outlinks['Ada_Lovelace'] = SortedSet(['Alan_Turing', 'Charles_Babbage'], key=None, load=1000)`

To obtain all the *outlinks* we call `self.get_links(SetofNames,HTML)`, which return a SortedSet of computer scientist names that are linked from this html page. The return set is restricted to those people in the provided set of names. The returned list should contain no duplicates. This function take as input:

- A SortedSet of computer scientist names, one per filename.
- A string representing one html page.

```
def get_links(self, names, html):
  listofHrefs = []
  listofHrefTexts = []
  FinalSortedSet = SortedSet()
  splice_char = '/'

  for i in range(0,len(listofHrefs)):
    value = listofHrefs[i][6:]
    listofHrefTexts.append(value)

  listofHrefTexts = re.findall(r'href', html)

  for i in listofHrefTexts:
    value = i[6:]
    listofHrefs.append(value)
  listofHrefs = list(set(listofHrefs))

  for href in listofHrefs:
    for name in names:
      if(name == "Guy_L._Steele,_Jr"):
        names.remove(name)
        names.add("Guy_L._Steele,_Jr.")
      if(href == name): FinalSortedSet.add(name)

  return FinalSortedSet
```

Listing 5: Get Links

With the results obtained by the function *make_links* we can calculate the pagerank by calling the function `compute_pagerank`.

```
def compute_pagerank(self, urls, inlinks, outlinks, b=.85, iters=20):
  rw = defaultdict(lambda:0.0)
  pageRank = defaultdict(lambda:1.0)

  for outlink in outlinks:  rw[outlink]=len(outlinks[outlink])

  #initialize page ranks scores to 1
  for url in urls:  pageRank[url] = 1.0

  for i in range(iters):
    for url in urls:
      summ = 0.0
      for link in inlinks[url]: summ += 1.0 * pageRank[link]/rw[link]
      pageRank[url] = (1/len(urls))* (1.0-b)+b*summ
  return SortedDict(dict(pageRank))
```

Listing 6: Computing Pagerank

This function return a *SortedDict* mapping each url to its final PageRank value (float) by using this formula:

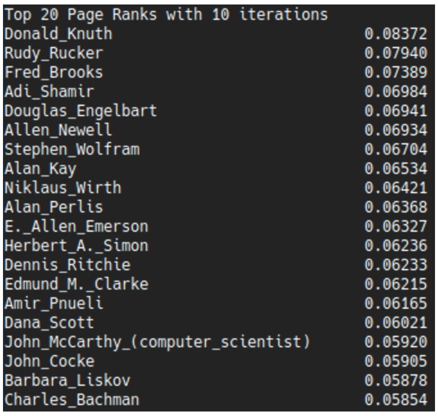
$$R(u) = \left(\frac{1}{N}\right)(1-b) + b \cdot \sum_{w \in B_u} \frac{R(w)}{|F_w|}$$

where:

- $R(u)$ = Pagerank value of the page u we want to calculate;
- $B(u)$ = A set of pages that contain at least one link to the u page. w represents each of these pages;
- $R(w)$ = PageRank values of each page w;
- F_w = Total number of links contained on the page offering the link;
- b = Damping factor. It is generally assumed that it will be set around 0.85.

3.3.1 My Pagerank Results

The following figure shows the terminal output after making the call to *compute_pagerank* with 20 iterations e 509 computer scientists.



```
Top 20 Page Ranks with 10 iterations
Donald_Knuth                              0.08372
Rudy_Rucker                               0.07940
Fred_Brooks                               0.07389
Adi_Shamir                                0.06984
Douglas_Engelbart                         0.06941
Allen_Newell                              0.06934
Stephen_Wolfram                           0.06704
Alan_Kay                                  0.06534
Niklaus_Wirth                             0.06421
Alan_Perlis                               0.06368
E._Allen_Emerson                          0.06327
Herbert_A._Simon                          0.06236
Dennis_Ritchie                            0.06233
Edmund_M._Clarke                          0.06215
Amir_Pnueli                               0.06165
Dana_Scott                                0.06021
John_McCarthy_(computer_scientist)        0.05920
John_Cocke                                0.05905
Barbara_Liskov                            0.05878
Charles_Bachman                           0.05854
```

Figure 2: *compute_pagerank* output

After applying the pagerank, it was decided to build a direct graph that highlighted the connections between each computer scientist. The figure shown below, built with the Python PIL library, shows that there is a thickening in the upper part which highlights the numerous connections that are among the top computer scientists obtained as an output from the *compute_pagerank* function.

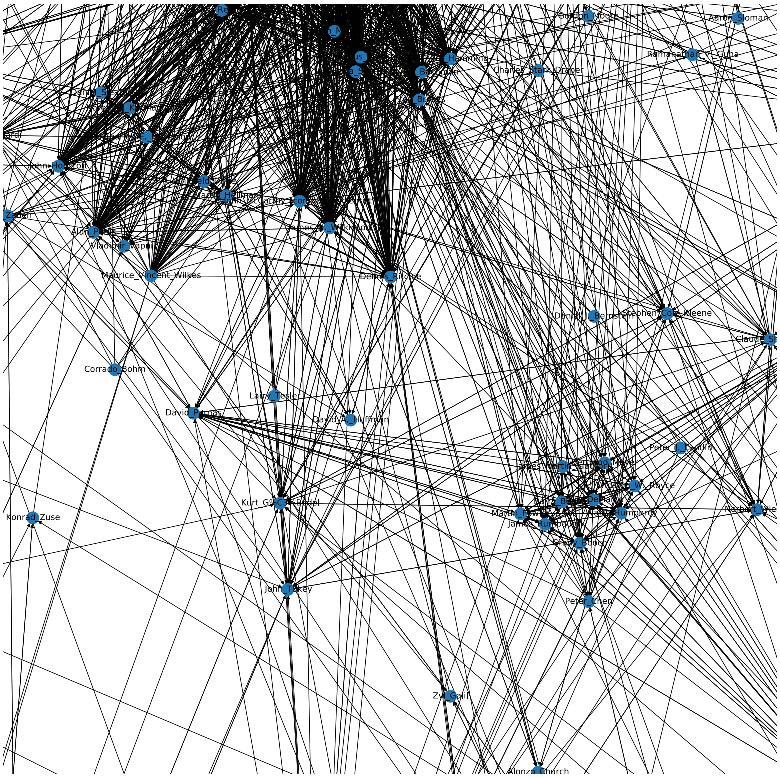


Figure 3: Pagerank Graph

4 Categorization

A further experiment was to try to draw up a ranking of the best branches of study carried out by these people.
In evaluating the fields that can be used, it has been verified that the majority of computer scientists present in the famous Wikipedia *infobox table* a field called *Field* which is right for us: it contains every category of study carried out from the person being examined.
Out of a total of 509 computer scientists, this field is present in about 480 people.
In the lower left is shown the code used to extrapolate this information and a function designed to create a json file that associates the name of a computer scientist with the categories to which it belongs.

```python
def get_fields(self, file_name):
  soup = BeautifulSoup(self.get_html_content(file_name), 'html.parser')
  infobox = soup.find('table', {'class':'infobox'})
  fields_list = []
  if infobox != None:
    for item in infobox.findAll('tr'):
      infobox_key = item.find('th')
      if infobox_key != None and infobox_key.get_text() == 'Fields':
        infobox_value = item.find('td')
        fields_list.append(infobox_value.get_text())
  this_name = self.get_cs_name_from_filename(file_name)
  return fields_list
```

Listing 7: Get Fields

```python
def compute_categorization(self, all_cs_files_list):
  to_ret = []
  none = 0
  for file_name in all_cs_files_list:
    cs_name = self.get_cs_name_from_filename(file_name)
    his_fields = self.clean_fields(self.get_fields(file_name))

    new_fields = []
    for item in his_fields: new_fields.append(item.capitalize())
    if new_fields != []:
      cs_name = urllib.parse.unquote(cs_name)
      to_ret.append({cs_name: new_fields})
    else:
      none += 1
  return to_ret
```

Listing 8: Categorization

After obtaining a json file containing the ordered set of categories associated with a computer scientist, a ranking was finally drawn up through the computation of the pagerank and hits algorithm.
The results obtained by the two algorithms are shown below:

hits_top_20_categories.txt

```
'Computer science', 0.326117140120153
'Mathematics', 0.0905002288461773
'Artificial intelligence', 0.06685354382887093
'Logic', 0.03427669758106854
'Electrical engineering', 0.02833238326892856
'Human-computer interaction', 0.021348931558433284
'Cognitive psychology', 0.01891198265547915
'Internet', 0.017865405333952915
'Cryptography', 0.01738984860471658
'Engineering', 0.017205005058716537
'Parallel computing', 0.016884266897918044
'Computer engineering', 0.01666589916367132
'Cognitive science', 0.012768419943534962
'Machine learning', 0.012000867181936405
'Theoretical biology', 0.011521251294523721
'Cryptanalysis', 0.011521251294523721
'Complex systems', 0.01109965631694415
'Political science', 0.0105244314790377
'Economics', 0.0105244314790377
'Biology', 0.010380178200082258
'Philosophy', 0.010380178200082258
```

pagerank_top_20_categories.txt

```
'Computer science', 0.04919569025380936
'Mathematics', 0.021521266029255075
'Artificial intelligence', 0.01889296875653205
'Human-computer interaction', 0.011111147419646208
'Theoretical computer science', 0.008998202553339458
'Logic', 0.008379779665639922
'Semantic web', 0.008122103462431782
'Machine learning', 0.00812210346243178
'Robotics', 0.007864427259223643
'Electrical engineering', 0.007761356779403845
'Entrepreneur', 0.006730651965107824
'Statistics', 0.006730651965107824
'Computer engineering', 0.006730651965107824
'Computational information systems', 0.006730651965107824
'Cognitive science', 0.006576046243182939
'Cryptography', 0.006215299558691543
'Parallel computing', 0.006215299558691543
'Computer graphics', 0.006215299558691543
'Operating systems', 0.006215299558691543
'Engineering', 0.005957623355483403
'Physics', 0.005957623355483403
```

References

[1] SPARQLWrapper: https://pypi.org/project/SPARQLWrapper/

[2] bs4 Beautiful Soup: https://www.crummy.com/software/BeautifulSoup/bs4/doc/

[3] urllib: https://docs.python.org/3/library/urllib.html

[4] requests: https://requests.readthedocs.io/en/master/

[5] Regular expression operations (re): https://docs.python.org/3/library/re.html

[6] NetworkX: https://networkx.github.io

[7] Matplotlib: https://matplotlib.org/users/pyplot_tutorial.html

[8] Sortedcontainers: http://www.grantjenks.com/docs/sortedcontainers/

[9] Collections: https://docs.python.org/3/library/collections.html

[10] Glob: https://docs.python.org/2/library/glob.html

[11] Tarfile: https://docs.python.org/3/library/tarfile.html

List of Figures

Listings