TP2 (Q1), and Atelier MRR

Vithor Bertalan, Matricule 2135362

In [2]:

```
import pandas as pd
import scipy.io
import igraph as ig
import numpy as np
from statistics import mean
import time

## Loads matrix and CSVs
m = scipy.io.mmread('tp-matrix.dgt')
articles = pd.read_csv("12-articles.csv", sep=",")
matrix_names = pd.read_csv("tp-matrix-names.csv", sep=" ")

## Transpose the matrix for the PageRank calculation
mat = m.transpose()

## Calculates adjacencies - takes a VERY LONG time
#g = ig.Graph.Adjacency(mat)
## If this is the first execution, saves the graph to a local file (uncomment line below,
and comment last line of this cell)
#g.write_graphml("file.gra")
## If the file is already written locally, reads it from this file, to speed executions
g = ig.Graph.Read_GraphML("file.gra", index=0)
```

Q1, Part 1 - Calculate 10 recommendations of each of the articles in the 12-articles subset

In [3]:

```
## Calculates PageRank for the graph, using iGraph's standard method
pr = g.pagerank()

## Gets the indexes of the 12-articles set in the overall matrix
def gets_indexes():
    pos = []
    for i in articles["id"]:
        count = 0
        for j in matrix_names["x"]:
            if (i==j):
                pos.append(count)
            count+=1
    return pos

## Classifies the recommendations of a given index, based on their PageRanks
def gets_recommendations(i):
    rows, cols = m.tocsr()[i].nonzero()
    ranks = np.empty([0, 2])
    for j in cols:
        ranks = np.vstack([ranks,[j,pr[j]]])
    return ranks[ranks[:, 1].argsort()[::-1]]

## Gets all the recommendations of the 12 articles set
def gets_articles_recommendations():
    pos = gets_indexes()
    count = 1
    for i in pos:
        recs = gets_recommendations(i)
        ## Adds one variable i, because index 0 is equivalent to line 1 in the dataset
        print("For paper number {}, number {} in the 12-articles set, the top-10 recomme
ndations by PageRank are:".format(i+1,count))
```

```
        vector = recs[:,0][:10].astype(int)
        ## Adds one to the recommendations, because index 0 is equivalent to line 1 in th
e dataset
        print([x+1 for x in vector])
        count+=1

gets_articles_recommendations()
```

```
For paper number 15089, number 1 in the 12-articles set, the top-10 recommendations by Pa
geRank are:
[22252, 3269, 3277, 16362, 18172, 8132, 9137, 13231, 6568, 6106]
For paper number 35353, number 2 in the 12-articles set, the top-10 recommendations by Pa
geRank are:
[338, 12764, 282, 5907, 12769, 25724, 2004, 12467, 333, 3187]
For paper number 50496, number 3 in the 12-articles set, the top-10 recommendations by Pa
geRank are:
[21335, 30836, 13710]
For paper number 50497, number 4 in the 12-articles set, the top-10 recommendations by Pa
geRank are:
[97, 107, 281]
For paper number 11636, number 5 in the 12-articles set, the top-10 recommendations by Pa
geRank are:
[1642, 6095, 1368, 4941, 33, 2859, 2083, 1323, 2271, 6204]
For paper number 12593, number 6 in the 12-articles set, the top-10 recommendations by Pa
geRank are:
[17, 373, 2040, 9665, 3043, 4259, 8312, 3221, 372, 5197]
For paper number 36565, number 7 in the 12-articles set, the top-10 recommendations by Pa
geRank are:
[8355, 696, 811, 939, 483, 334, 703, 8044, 3411, 7413]
For paper number 12215, number 8 in the 12-articles set, the top-10 recommendations by Pa
geRank are:
[11776, 4975, 12217, 6716, 12218, 12216, 9523, 7113, 6714, 3583]
For paper number 18645, number 9 in the 12-articles set, the top-10 recommendations by Pa
geRank are:
[6731, 18724, 8563, 69, 3348, 876, 8927, 7307, 18885, 10844]
For paper number 1594, number 10 in the 12-articles set, the top-10 recommendations by Pa
geRank are:
[8425, 18760, 9310, 8021, 13384, 19336, 13054, 11993, 9319, 9315]
For paper number 35304, number 11 in the 12-articles set, the top-10 recommendations by P
ageRank are:
[19649, 3729, 19812, 6138, 6119, 15177, 23327, 37796, 8179, 6126]
For paper number 18539, number 12 in the 12-articles set, the top-10 recommendations by P
ageRank are:
[1627, 8420, 18540, 2143, 18011, 18554, 18548, 9396, 18541, 5613]
```

Q1, Part 2 - Calculate the MRR for the references of each of the 12 articles.

In [4]:

```
## Creates ordered set of pageranks - the best pagerank will be first
def order_ranks():
    ranks = np.empty([0, 2])
    for i in range(len(pr)):
        ranks = np.vstack([ranks,[i,pr[i]]])
    return ranks[ranks[:, 1].argsort()[::-1]]

## Given the ordered set, finds the rank of a given index
def find_rank_index(i, ranks):
    count = 1
    for j in ranks[:,0]:
        if (i==j):
            return count
        count+=1

## Calculates mean reciprocal rank for each of the connections of a given index
def calculate_mrr(i):
    ## Orders the ranks
    ranks = order_ranks()
    rows, cols = m.tocsr()[i].nonzero()
    rrs = []
```

```
    ## For each of the references...
    for j in cols:
        ## Gets the rank of that index in the ordered list...
        index = find_rank_index(j,ranks)
        ## Calculates the reciprocal rank (1/rank)
        rrs.append(1/index)
    ## And returns the mean of those values
    return(mean(rrs))

## Calculates MRR for each of the 12 articles
def calculate_mrr_articles():
    pos = gets_indexes()
    count = 1
    for i in pos:
        mrr = calculate_mrr(i)
        ## Adds one to variable i, because index 0 is equivalent to line 1 in the dataset
        print("For paper number {}, number {} in the 12-articles set, the MRR is {}".for
mat(i+1,count,mrr))
        count+=1

start_time = time.time()
calculate_mrr_articles()
print("The MRR for the 12 papers was calculated in {} seconds".format((time.time() - star
t_time)))
```

```
For paper number 15089, number 1 in the 12-articles set, the MRR is 3.4244840276777825e-0
5
For paper number 35353, number 2 in the 12-articles set, the MRR is 7.448341720585364e-05
For paper number 50496, number 3 in the 12-articles set, the MRR is 2.7973042562303706e-0
5
For paper number 50497, number 4 in the 12-articles set, the MRR is 6.425012369398826e-05
For paper number 11636, number 5 in the 12-articles set, the MRR is 4.283362564269419e-05
For paper number 12593, number 6 in the 12-articles set, the MRR is 0.0002370609691134385
2
For paper number 36565, number 7 in the 12-articles set, the MRR is 0.0001120308400529870
5
For paper number 12215, number 8 in the 12-articles set, the MRR is 2.592960811804186e-05
For paper number 18645, number 9 in the 12-articles set, the MRR is 0.0001025079188714051
For paper number 1594, number 10 in the 12-articles set, the MRR is 3.4371001611510045e-0
5
For paper number 35304, number 11 in the 12-articles set, the MRR is 0.000140335746252915
97
For paper number 18539, number 12 in the 12-articles set, the MRR is 0.000121309549218497
27
The MRR for the 12 papers was calculated in 22.05690026283264 seconds
```

Atelier MRR - Calculate an alternative to MRR. In this case, we will use Mean Average Precision (MAP) and a custom metric.

In [5]:

```
## Calculates the average precision for the connections of a given index
## Source of idea: https://stats.stackexchange.com/questions/127041/mean-average-precisio
n-vs-mean-reciprocal-rank
## Second source: http://sdsawtelle.github.io/blog/output/mean-average-precision-MAP-for-
recommender-systems.html

def calculate_map(i):
    ## First, we order the papers paged on their PageRanks (the first has the higher Page
Rank)
    ranks = order_ranks()
    ## Then, we ge ordered list of recommendations (see method above)
    recs = gets_recommendations(i)
    aps = []
    precision_count = 1

    ## For each of the references...
    for j in recs[:,0]:
        ## Gets the rank of that index in the ordered list...
```

```
        index = find_rank_index(j,ranks)
        ## Calculates the precision (number of recommendations already given divided by i
ndex of current recommendation)
        aps.append(precision_count/index)
        precision_count+=1

    ## Returns the multiplication of (1 divided by the number of recommendations) * sum o
f average precisions
    return(1/len(recs) * sum(aps))

## Calculates MAP for each of the 12 articles
def calculate_map_articles():
    pos = gets_indexes()
    count = 1
    for i in pos:
        map_ = calculate_map(i)
        ## Adds one to variable i, because index 0 is equivalent to line 1 in the dataset
        print("For paper number {}, number {} in the 12-articles set, the MAP is {}".for
mat(i+1,count,map_))
        count+=1

start_time = time.time()
calculate_map_articles()
print("The MAP for the 12 papers was calculated in {} seconds".format((time.time() - star
t_time)))
```

```
For paper number 15089, number 1 in the 12-articles set, the MAP is 0.0001863237070876400
5
For paper number 35353, number 2 in the 12-articles set, the MAP is 0.0004123646549474845
6
For paper number 50496, number 3 in the 12-articles set, the MAP is 5.2378925833792124e-0
5
For paper number 50497, number 4 in the 12-articles set, the MAP is 9.641444363561471e-05
For paper number 11636, number 5 in the 12-articles set, the MAP is 0.0002541274689494590
6
For paper number 12593, number 6 in the 12-articles set, the MAP is 0.001303818573485607
For paper number 36565, number 7 in the 12-articles set, the MAP is 0.0003249088626849813
4
For paper number 12215, number 8 in the 12-articles set, the MAP is 0.0001859600374129505
For paper number 18645, number 9 in the 12-articles set, the MAP is 0.0007830707476872126
For paper number 1594, number 10 in the 12-articles set, the MAP is 0.0001891485895923592
For paper number 35304, number 11 in the 12-articles set, the MAP is 0.000498174519238168
2
For paper number 18539, number 12 in the 12-articles set, the MAP is 0.000917270322453069
8
The MAP for the 12 papers was calculated in 12.48408818244934 seconds
```

As we can see, MAP has similar results to MRR (e.g., article number 12593, number 6 in the articles set, is the best in both metrics), but the calculation time takes half the time of MRR - 12 seconds against 22 seconds.

In my point of view, if article 12593 is the top article, this means it has more influent citations than the other papers.

Let's confirm this information, by taking the first two reference in the ordered recommendation set of paper 12593, which are papers 17 and 373, as we can see in the beginning of the notebook.

In [8]:

```
#For paper number 12593, number 6 in the 12-articles set, the top-10 recommendations by P
ageRank are:
#[17, 373, 2040, 9665, 3043, 4259, 8312, 3221, 372, 5197]

## Creates ordered set of pageranks - the best pagerank will be first
def order_ranks():
    ranks = np.empty([0, 2])
    for i in range(len(pr)):
```

```
        ranks = np.vstack([ranks,[i,pr[i]]])
    return ranks[ranks[:, 1].argsort()[::-1]]

## Given the ordered set, finds the rank of a given index
def find_rank_index(i, ranks):
    count = 1
    for j in ranks[:,0]:
        if (i==j):
            return count
        count+=1

ranks = order_ranks()
print(find_rank_index(16,ranks))
print(find_rank_index(372,ranks))
```

```
300
1031
```

As we can see, paper 12593 is cited by, among others, the 300th and 1031th top papers by PageRank, making it very influential. Let's compare those results with the top 2 references of paper 50496, number 3 in the 12-articles set, the worst paper by MAP metrics:

In [32]:

```
#For paper number 50496, number 3 in the 12-articles set, the top-10 recommendations by P
ageRank are:
#[21335, 30836, 13710]

print(find_rank_index(21334,ranks))
print(find_rank_index(30835,ranks))
```

```
31666
31785
```

Besides being cited by only 3 papers, the top-2 references of 50496 are quite down below the ordered set of recommendations: it is cited by the 31666th and 31785th papers. Therefore, we can see the MAP is a very good and fast metric to order the most important papers of our query.

Now, let's create a custom metric: we are going to multiply 1/index by the pagerank of the recommendation, for each recommendation, and sum the result. This is done to penalize papers with few recommendations, even if it has recommendations with high pageranks.

The formula for our new metric is: metric = Σ 1/index(n) * PageRank(n), with Σ going from 1 to n, with n being the full subset of articles that cite that paper.

In [7]:

```
## Calculates the custom metric for a given subset

## Orders ranks by PageRank
def order_ranks():
    ranks = np.empty([0, 2])
    for i in range(len(pr)):
        ranks = np.vstack([ranks,[i,pr[i]]])
    return ranks[ranks[:, 1].argsort()[::-1]]

## Given the ordered set, finds the rank of a given index
def find_rank_index(i, ranks):
    count = 1
    for j in ranks[:,0]:
        if (i==j):
            return count
        count+=1
```

```
## Calculates new metric for each of the connections of a given index
def calculate_new_metric(i):
    ## Orders the ranks
    ranks = order_ranks()
    rows, cols = m.tocsr()[i].nonzero()
    rrs = []

    ## For each of the references...
    for j in cols:
        ## Gets the rank of that index in the ordered list...
        index = find_rank_index(j,ranks)
        ## Calculates the new metric rank (1/rank)
        rrs.append(1/index * pr(j))
    ## And returns the sum of those values multiplied by
    return(sum(rrs))

## Calculates new metric for each of the 12 articles
def calculate_new_metric_articles():
    pos = gets_indexes()
    count = 1
    for i in pos:
        mrr = calculate_new_metric(i)
        ## Adds one to variable i, because index 0 is equivalent to line 1 in the dataset
        print("For paper number {}, number {} in the 12-articles set, the new metric is
{}".format(i+1,count,mrr))
        count+=1

start_time = time.time()
calculate_mrr_articles()
print("The metric for the 12 papers was calculated in {} seconds".format((time.time() - s
tart_time)))
```

```
For paper number 15089, number 1 in the 12-articles set, the MRR is 3.4244840276777825e-0
5
For paper number 35353, number 2 in the 12-articles set, the MRR is 7.448341720585364e-05
For paper number 50496, number 3 in the 12-articles set, the MRR is 2.7973042562303706e-0
5
For paper number 50497, number 4 in the 12-articles set, the MRR is 6.425012369398826e-05
For paper number 11636, number 5 in the 12-articles set, the MRR is 4.283362564269419e-05
For paper number 12593, number 6 in the 12-articles set, the MRR is 0.0002370609691134385
2
For paper number 36565, number 7 in the 12-articles set, the MRR is 0.0001120308400529870
5
For paper number 12215, number 8 in the 12-articles set, the MRR is 2.592960811804186e-05
For paper number 18645, number 9 in the 12-articles set, the MRR is 0.0001025079188714051
For paper number 1594, number 10 in the 12-articles set, the MRR is 3.4371001611510045e-0
5
For paper number 35304, number 11 in the 12-articles set, the MRR is 0.000140335746252915
97
For paper number 18539, number 12 in the 12-articles set, the MRR is 0.000121309549218497
27
The metric for the 12 papers was calculated in 22.640989065170288 seconds
```

Using this new metric, a paper with just one high-ranked PageRank citation would not be necessarily well placed, being surpassed by a paper with two not as well high-ranked PageRank citations. In this way, we penalize papers with not many recommendations.