Adaptive methods for the computation of PageRank

Analysis by Taras Mazurkevych and Tetiana Batsenko

What is PageRank?

PageRank is an algorithm that computes the importance of graph nodes.

PageRank of page x is a probability that at the current time web crawler is on page x.

Web here is represented as a directed graph, where links from page to page are graph connections.

Probability that at the current time web crawler is on page **x** is formed by following equation:

$$PR(x) = \frac{1-\lambda}{N} + \lambda \sum_{y \to x} \frac{PR(y)}{out(y)}$$

Web crawler could come to the page **x** in two different ways.

That's why this equation contains two parts which form probability that web crawler is on page **x** at the current time.

1st way: Web crawler could choose the page **x** randomly.- left part of the sum in the equation

2nd way: Web crawler could come to the page **x** from some page **y** that has a link to **x** **-**

- right part of the sum in the equation.

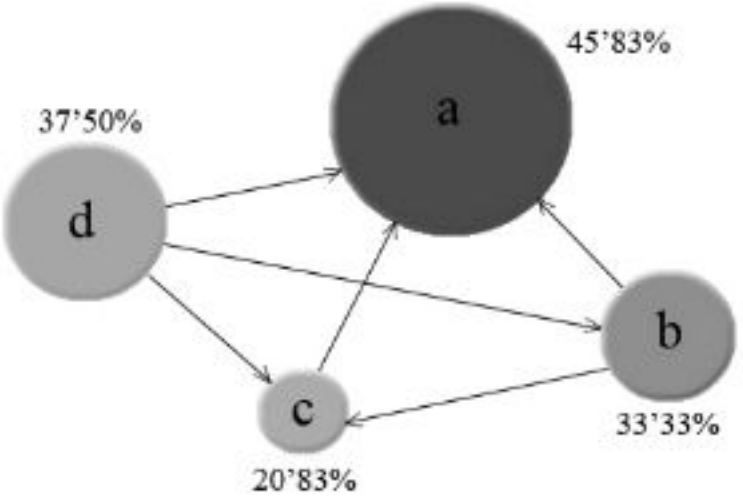
PR(x) == PageRank of the node **x,** note that

$\lambda \in [0, 1]$ - weight that represents the probability of web crawler is travelling by links, not randomly

N - amount of nodes in the graph, in our case - total amount of pages

out(y) - amount of outgoing edges (links) from page y.

(for example for node d on the picture below out(d) = 3)



Problem Statement

PageRank algorithm runtime is quick for most pages on the web.
However, it can be much slower for a relatively small group of pages.
But it seems that those pages are the most important - these are pages with high PageRank.

There is a need to shorten the runtime of PageRank algorithm for those pages.

Task: shorten the runtime of PageRank algorithm for pages that converge* slow.

General description of the proposed algorithm

To solve this problem authors of the article propose an algorithm which speeds up the computation of PageRank by nearly 30%.

They call this algorithm Adaptive PageRank. The optimization behind it is that the algorithm does not recompute PageRank of pages which have already converged and their impact on neighbour pages.

Where is Adaptive PageRank used?

Adaptive PageRank is a more efficient modification of PageRank.
First of all, it's the algorithm behind Google's search results.

The principles of PageRank are entirely general and apply to any graph or network in any domain.
Thus, PageRank **is widely used** in bibliometrics, social and information network analysis, and for link prediction and recommendation.

In neuroscience, the PageRank of a neuron in a neural network has been found to correlate with its relative firing rate.

Personalized PageRank is used by Twitter to offer users new accounts to follow.

A new use of PageRank is to rank academic doctoral programs based on their records of placing their graduates in faculty positions. In PageRank terms, academic departments link to each other by hiring their faculty from each other (and from themselves).

PageRank has been used to rank spaces or streets to predict how many people (pedestrians or vehicles) come to the individual spaces or streets.

Step by step description

Algorithm 3. Adaptive PageRank

```
1 function adaptivePR(A, x^(0), v) {
2 repeat
3     x_N^(k+1) = A_N x^(k);
4     x_C^(k+1) = x_C^(k);
5     [N, C] = detectConverged(x^(k), x^(k+1), ε);
6     periodically, δ = ||A x^(k) − x^k||_1;
7 until δ < ε;
8 return x^(k+1);
  }
```

Xo - is the initial PR vector - for every i - Xo[i] = 1 / n, where n is total amount of vertices
Vector x^k contains PageRanks for all websites on **k-**th iteration
We divide it into Xn and Xc.
Xc contains pages which have already *converged** and Xn the ones which haven't yet.

A - is a stochastic matrix P.transpose() influenced by random_walk coefficient, where P[i][j] = 1 / out(i) , **out(i)** is the outdegree of vertex **i**.
Thus the sum of **i**'s column of the matrix A is equal to 1 for every i, and A is the Markov matrix with corresponding eigenvalue $\lambda = 1$.
Then the PageRank vector is simply the eigenvector for eigenvalue $\lambda = 1$,
that solves Ax = x.

We repeat loop body, until the X is *converged**

1. Recompute non converged items from previous iteration
2. Copy converged items from previous iteration. That's the first part of optimisation.
3. detectConverged function checks all the pages, if they converged.
   Call this function and assign the result to Xn and Xc
4. To check if the whole X converged, recompute PageRank for all the pages in X and take the norm of that vector.
   If it's less than ε ** - X converged

(*) - We say that the PageRank x_i of page *i* has converged when the following condition is true: $\left|x_i^{k+1} - x_i^k\right| \; / \; \left|x_i^k\right| \; < \; 10^{-3}$
(**) - ε is initialized at the beginning (usually it's something less than 10^{-3} or even 10^{-5} depending on a given data and the accuracy we wanna get)

Time complexity

We claim that time complexity of AdaptivePR vary between O(n+m) and O($c * n^2$)
n *- number of nodes in the graph*
m *- number of edges in the graph*
c *- number of iterations that is needed for delta to be less than epsilon ($\delta < \varepsilon$)*
Let's explain why:
We have to touch at least every node (from the article we see that most pages will converge less than in 15 iterations), which gives us n in big O.
When we do matrix by vector multiplication (on the line 3 of pseudo code above) it has to look at each link (edge) at least 2 times - which gives us m in big O - the lower bound.

Now let's analyse the most costfull operation per iteration - a matrix-vector multiplication which can be done in O(n^2) time.
Then apply to that the amount of iterations and we get O($c * n^2$).

AdaptivePageRank doesn't recompute PageRank of pages which have already converged and their impact on neighbour pages by reducing matrix A.
A becomes a sparse matrix and it significantly reduces the cost of matrix-vector multiplication.

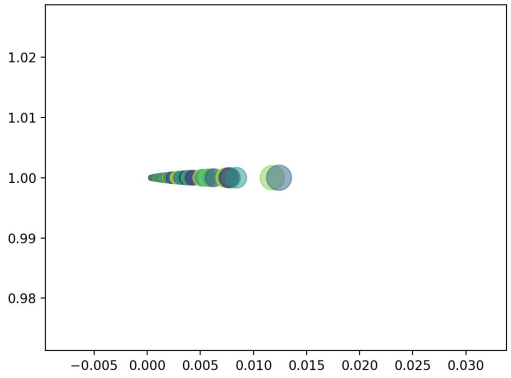
Thus, the final complexity vary between O(n+m) and O $(c * n^2)$

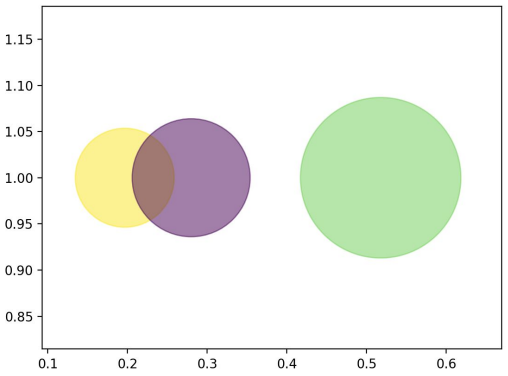
Experiment

https://lintool.github.io/Cloud9/docs/exercises/pagerank.html is a source of "small" dataset (93 nodes, 195 edges) and "large" one (1458 nodes, 3545 edges). We also generated a super-small dataset (3 nodes, 3 edges). On that website there are also PageRank values captured after running PageRank algorithm.

On Figure 1 there is a result of running Adaptive PageRank on "large" dataset. On Figure 2 there is a result of running Adaptive PageRank on "super-small" dataset. Position of item on horizontal axis describes a PageRank of this item.

The results of running Adaptive PageRank and results from the source might differ. But after rounding the results to the precision of epsilon they will be equal.





Conclusions

While working on AdFontes project we learned those important things:

- How to implement an algorithm from research articles and test it
- How to work on research articles and make sure that you understand ideas behind it
- Python tools for working with Linear Algebra and visualisations - numpy, matplotlib

The most challenging task for us was to transform pseudocode to real code and implement every detail, important for the final result.

The accurate approximation of time complexity of the Adaptive PageRank algorithm in big-O notation still remains unclear for us since it's hard to say how many iterations is needed for delta to be less than epsilon ($\delta < \varepsilon$).

Link to GitHub repository

https://github.com/tbatsenko/pagerank_adfontes

The algorithm is implemented in page_rank.py file, data samples are stored in data/ folder. data/sample-large.txt and data/sample-super-small.txt were used for examples in experiment section. Run with python 3.