

Présentation projet de recherche

Théodore CHAPUIS-CHKAIBAN, Ghita CHEMIA, Paul
LIEUTIER, Hugo SERVANT, Victor SANSAC

Ecole CentraleSupélec

10/03/2020

Outline

- ① Introduction to page Rank
 - Main ideas
 - Quantum Page Rank
 - Complexity
- ② Axes of improvement
 - Complexity
 - What about machine learning ?
 - How do quantum computers proceed the information ?
- ③ What's next ?
 - Work on the classical PageRank
 - Overview of the forthcoming work

Page Rank

- Principe : étudier la centralité du réseau internet
- Lien avec les algorithmes quantiques : le déplacement de l'utilisateur correspond à une marche aléatoire sur le graphe du web selon un processus de Markov

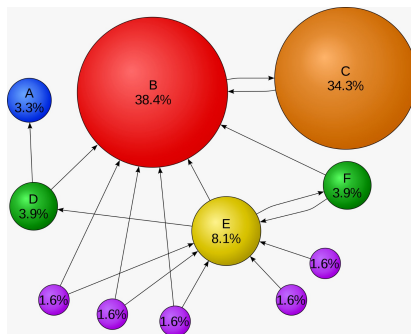


FIGURE 1 – Exemple de Graphe sur lequel fonctionne le Page Rank.

Application

Une seule application réelle à l'heure actuelle :

- La classification des pages internet qui repose sur le nombre de pages qui contiennent un lien vers une page donnée.
- Il en découle un indice variant entre 1 et 10

Quantum Page Rank

Google Page Rank

- Uses Grover search
- Uses Markovian chain search
- Quantum walk \rightarrow matrix reduction ; SVD

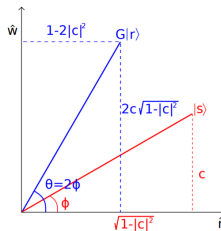


FIGURE 2 – Grover search is used to compute the page rank using the Perron-Frobenius Theorem.

Complexity calculation

Theorem 1

The average complexity of these methods is \sqrt{N} where N is the size of the input data set.

Theorem 2

This complexity is the minimum one can reach when searching for an element in a predefined set (Grover Search).

Complexity enhancement

- To improve the complexity, we have to drop the Grover Algorithm
- How can we improve the complexity ? Some quantum algorithms can reach a complexity of $\log(N)$ in some cases.
- The Quantum Fourier transform is one example

Quantum Fourier Transform

- What is the link between the Quantum Fourier Transform and the PageRank algorithm ?
- What about using wavelets to try optimise the performances ? (see "Dimension Reduction Using Quantum Wavelet Transform on a High-Performance Reconfigurable Computer" by Naveed Mahmud and Esam El-Araby)

Assumption

Since PageRank involves methods of matrix reduction (via SVD), Quantum Fourier Transform methods for frequencies analytics can be applied to this problem. (see "A SHORT-GRAPH FOURIER TRANSFORM VIA PERSONALIZED PAGERANK VECTORS" by Mariano Tepper and Guillermo Sapiro)

Machine learning methods to broaden the field of applications

- Machine learning involves methods used in PageRank to classify data
- Complexity in $\log(n)$ (see "Quantum support vector machine for big data classification" by Patrick Rebentrost and all)
- May be interesting to establish a parallel between the two methods.

How do quantum computers proceed the information ?

- Quantum grover search exhibit a better understanding of webpages by allowing more importance to secondary hubs
- Why is there a difference between classical and quantum computers ?

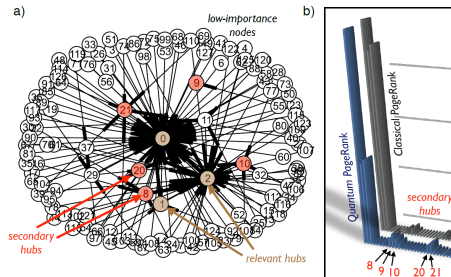


FIGURE 3 – Quantum algorithms resolves secondary hubs better than the classical ones.

Code Classical PageRank

```
#!/usr/bin/env python3
"""
Created on Tue Mar 10 12:18:34 2020
@author: paul
"""
import networkx as nx

G = nx.gnp_random_graph(10, 0.1)

def pagerank(G, alpha=0.05, personalization=None, max_iter=100, tol=1.0e-6, nstart=None, weight=None):
    if len(G) == 0:
        raise ValueError("No nodes, it's not possible to compute PageRank")
        return {}

    if not G.is_directed():
        raise ValueError("The graph must be directed for the algorithm to work")
    D = G.to_directed()
    else:
        D = G

    # création d'une copie stochastique du graphe
    W = nx.stochastic_graph(D, weight=weight)
    N = W.number_of_nodes()

    # si on a pas de dico initialement choisi on utilise une notation uniforme
    if nstart is None:
        x = dict.fromkeys(W, 1.0 / N)
    else:
        # on normalise le vecteur d'entrée
        s = float(sum(nstart.values()))
        x = dict((k, v / s) for k, v in nstart.items())

    if personalization is None:
        # vecteur de personnalisation uniforme dans ce cas
        p = dict.fromkeys(W, 1.0 / N)
    else:
        missing = set(G) - set(personalization)
        if missing:
            raise nx.NetworkXError("Personalization dictionary 'personalization' must have a value for every node. 'Missing nodes %s' % missing)
        s = float(sum(personalization.values()))
        p = dict((k, v / s) for k, v in personalization.items())

    if dangling is None:
        # utilisation du vecteur de personnalisation dans ce cas
        dangling_weights = p
    else:
        missing = set(G) - set(dangling)
        if missing:
            raise nx.NetworkXError("Dangling node dictionary 'dangling' must have a value for every node. 'Missing nodes %s' % missing)
        s = float(sum(dangling.values()))
        dangling_weights = dict((k, v/s) for k, v in dangling.items())
        dangling_nodes = [n for n in W if W.out_degree(n, weight=weight) == 0.0]

    # ne pas dépasser max_iter
    for _ in range(max_iter):
        xlast = x
        x = dict.fromkeys(W, 0)
        danglessum = alpha * sum(xlast[n] for n in dangling_nodes)
        for n in x:
            # this matrix multiply looks odd because it is
            # doing a left multiply xT*W*xlast*TW
            for nbr in W[n]:
                x[nbr] += alpha * xlast[n] * W[n][nbr][weight]
                x[n] += danglessum * dangling_weights[n] + (1.0 - alpha) * p[n]

        # check convergence, l1 norm
        err = sum([abs(x[n] - xlast[n]) for n in x])
        if err < N*tol:
            return x
    raise nx.NetworkXError("pagerank: power iteration failed to converge 'in %d iterations.'" % max_iter)
```

```
if personalization is None:
    # vecteur de personnalisation uniforme dans ce cas
    p = dict.fromkeys(W, 1.0 / N)
else:
    missing = set(G) - set(personalization)
    if missing:
        raise nx.NetworkXError("Personalization dictionary 'personalization' must have a value for every node. 'Missing nodes %s' % missing)
    s = float(sum(personalization.values()))
    p = dict((k, v / s) for k, v in personalization.items())

if dangling is None:
    # utilisation du vecteur de personnalisation dans ce cas
    dangling_weights = p
else:
    missing = set(G) - set(dangling)
    if missing:
        raise nx.NetworkXError("Dangling node dictionary 'dangling' must have a value for every node. 'Missing nodes %s' % missing)
    s = float(sum(dangling.values()))
    dangling_weights = dict((k, v/s) for k, v in dangling.items())
    dangling_nodes = [n for n in W if W.out_degree(n, weight=weight) == 0.0]

# ne pas dépasser max_iter
for _ in range(max_iter):
    xlast = x
    x = dict.fromkeys(W, 0)
    danglessum = alpha * sum(xlast[n] for n in dangling_nodes)
    for n in x:
        # this matrix multiply looks odd because it is
        # doing a left multiply xT*W*xlast*TW
        for nbr in W[n]:
            x[nbr] += alpha * xlast[n] * W[n][nbr][weight]
            x[n] += danglessum * dangling_weights[n] + (1.0 - alpha) * p[n]

    # check convergence, l1 norm
    err = sum([abs(x[n] - xlast[n]) for n in x])
    if err < N*tol:
        return x
    raise nx.NetworkXError("pagerank: power iteration failed to converge 'in %d iterations.'" % max_iter)
```

FIGURE 4 – Python Code for the Classical PageRank

What's next ?

- ① Code the classical and quantum versions of PageRank using Grover Search and the Perron Frobenius Theorem
- ② Find an equivalent for the Fourier transform and implement it
- ③ Adapt it to the Wavelet method
- ④ Quantify the economy made in calculations
- ⑤ Adapt the algorithm to Big Data issues involving reduction of dimension