UNIVERSITAT POLITÈCNICA DE CATALUNYA

GRADO EN INGENIERÍA INFORMÁTICA

CERCA I ANÀLISI D'INFORMACIÓ MASSIVA

Q1 Course 2020-2021

Page Rank

GRUP 21

Autores:

Daniel CANO CARRASCOSA

0 INTRODUCCIÓN

Para esta entrega voy a implementar el algoritmo de PageRank y analizar tanto su implementación y ciertos detalles de la misma que he tenido de tener en cuenta para un correcto funcionamiento, así como los tiempos obtenidos modificando ciertos parámetros para así poder visualizar el tiempo de convergencia.

1 FUNCIONAMIENTO BÁSICO DEL PROGRAMA E IMPLEMENTACIÓN

En cuanto al funcionamiento del programa tenemos un programa bastante sencillo, excepto unos pequeños detalles que mencionaré más tarde relacionados con los nodos pozo, donde la estructura del programa sigue la secuencia de:

1. Leer aeropuertos (código ya implementado por los profesores)
2. Leer rutas (código implementado por mi parte): donde se leen las rutas y se guardan en los correspondientes aeropuertos
3. Computar PageRank
4. Mostrar Resultados

De todo este proceso respecto al código final proporcionado por los profesores solo destacar que en el código original había una clase Edge, que implicaba que representa las aristas entre aeropuertos, código que por mi parte he visto sobrante he innecesario, con lo que lo he eliminado, ya que con la clase aeropuertos ya se puede hacer la implementación sin mayores problemas.

En cuanto a la implementación, dejando de lado el detalle mencionado previamente y que comentaré en la siguiente sección, no ha habido realmente problemas, ya que gracias a como es python, y que prácticamente hay situaciones en las que parece pseudocódigo, solamente he tenido que traducir del pseudocódigo proporcionado por los profesores en el pdf, a un código python implementado por mí, por lo que por parte de la implementación no ha habido

mayores problemas, dejando de lado la decisión del "Damping Factor" o factor de amortiguación, el cual, tal como se vio en teoría el objetivo de este parámetro es el de evitar que hay nodos con 0 de probabilidad de terminar en ellos, es decir evitamos cierta tendencia a evitar partes de nuestra red de aeropuertos de ahí que el cálculo del dumping factor sea que para cada nodo añadimos la probabilidad de (1-L)*(1/n), donde L es la importancia que le damos a la probabilidad calculada por el pagerank, y por ende (1-L) la importancia de este "ruido", y 1/n la probabilidad que vamos a distribuir, siendo esta igual para todos los nodos con el objetivo de no beneficiar más a unos que otros, ya que si no terminariamos en la misma situación. Dejando este detalle del dumping factor de lado el resto de la implementación del algoritmo es lo mismo explicado en teoría.

2 PROBLEMAS CON LOS NODOS POZO

Ahora que ya sabemos el flujo del programa así como el motivo del dumping factor, nos queda ir quizás a uno de los detalles más importantes del algoritmo (no quiero decir que sea el más importante ya que como veremos después el damping factor también juega un papel importante), y que no está en el pseudocódigo, que el de cómo tratar con los nodos pozo, los cuales si no se hace nada respecto a ellos por la implementación del algoritmo se está perdiendo su probabilidad, ya que cuando se llegue a estos nodos, al cálculo no se repartira por ningun lugar, mientras que los demás nodos conectados a estos le están asignando una nueva probabilidad, por lo que cuando se haga P = Q, se estará perdiendo el porcentaje de esos nodos en concreto.

Para solucionar esto mi primero intento ha sido de, por cada nodo pozo aplicar un cálculo de repartir la probabilidad de estos entre todos los demás que no eran pozos, lo cual estaba incorrecto ya que como he podido comprobar de primera mano, esto provocaba que a nivel de cómputo, era como añadir N aristas para cada uno de estos vértices, lo que al final se traducía en tiempo cuadrático y esto es un detalle que queremos evitar.

Para resolver esta situación por lo que opte finalmente fue por el siguiente método. Debido a que inicialmente la probabilidad inicial de todos los nodos era de 1/N, y de que podía detectar

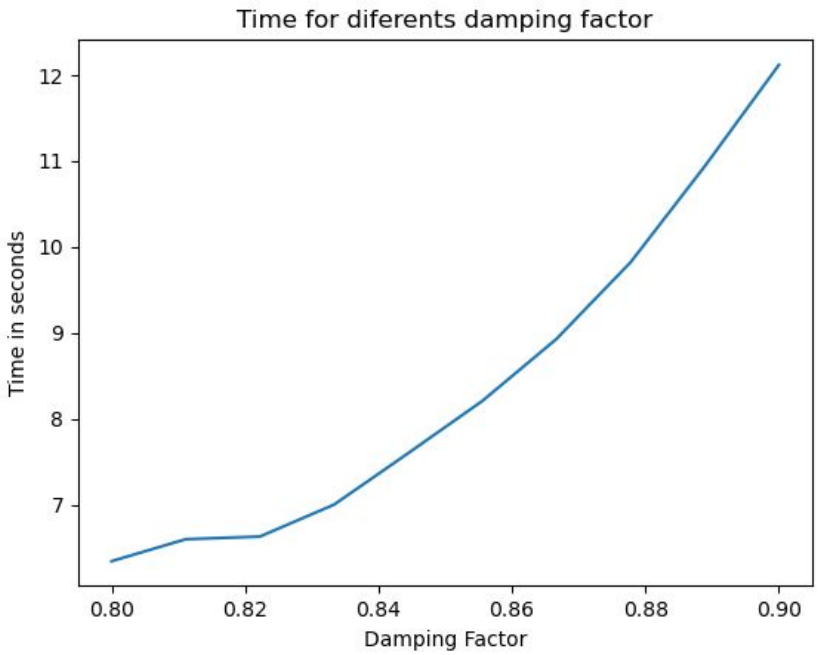
la cantidad de nodos pozo antes de empezar el algoritmo, sencillamente para distribuir la probabilidad de todos los nodos pozo entre aquello que no lo eran, podía sencillamente sumar, el cálculo del pagerank correspondiente y a esto sumarle la probabilidad de hacer la operación de (L * número de nodos vacíos) * (probabilidad_de_todos_los_nodos_vacios/N), con este calculo lo que hago es distribuir la probabilidad entre todos los nodos vacíos entre N (para cada uno la misma probabilidad), lo multiplico por todos los nodos vacíos (imagino que todos los nodos vacíos están conectados con todos los nodos) y lo multiplico por la importancia de L, de esta forma estamos emulando a un nivel virtual el comportamiento esperado de crear N aristas, lo único que por cada computo completo he de hacer nuevamente el cálculo de la nueva probabilidad de salida de todos los nodos, pero como esta va a ser diferente a partir de la segunda iteración (en la primera sabíamos que era 1/N pero en adelante ya no la podemos controlar), para hacer una implementación eficiente podemos hacer lo mismo de antes (la probabilidad saliente para cada nodo) sumandole (1-L)/N, ya que nosotros sabemos que este ruido siempre lo estamos añadiendo a estos nodos pozo, por lo que una vez hecho esto, ya tendríamos nuestro algoritmo implementado correctamente.

3 TIEMPOS Y RESULTADOS

Una vez ejecutado el algoritmo 10 veces se pueden observar los siguientes resultados tanto a nivel de tiempo como de resultados.

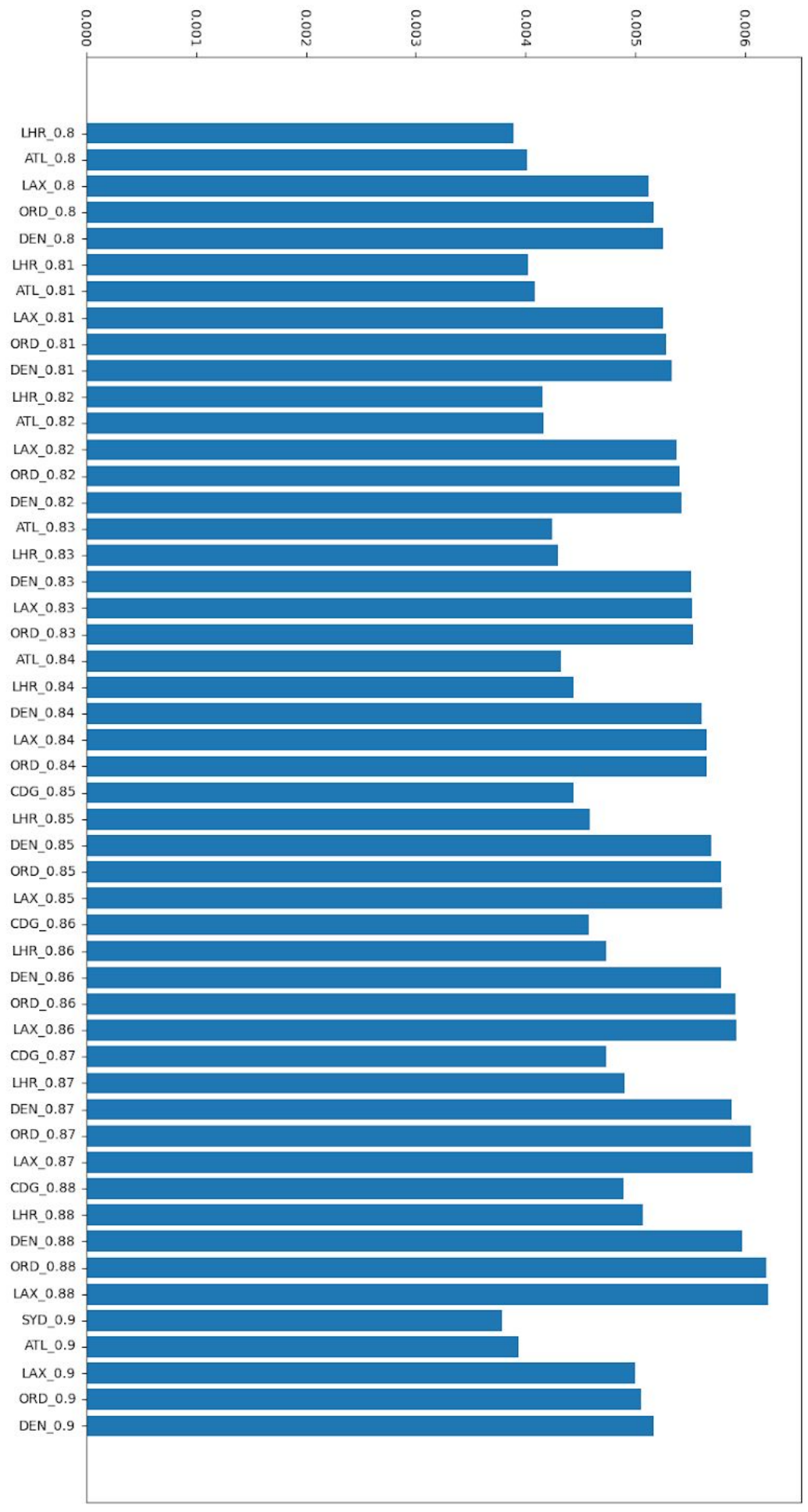
En cuanto al aspecto del tiempo y de su evolución respecto a la modificación del damping factor, podemos ver en la siguiente gráfica como este se ve altamente alterado por este valor variando considerablemente a medida que lo cambiamos, siendo menor a menor es el damping factor, lo cual es normal ya que hacerlo menor implica darle más importancia al peso calculado de la forma (1-L)/N, el cual es siempre constante, y debido a que mi condición de parada se basa en la diferencia del cálculo anterior y el actual (ya que implementar unas iteraciones no me parecía un buen método ya que era demasiado subjetivo), el algoritmo para

mucho más rápido que si nos aproximamos a 1 y le damos más importancia al algoritmo de pagerank. No obstante hay que tener en cuenta que esto es normal, ya que si hacemos el damping factor igual a 1, al no tener un grafo fuertemente conexo (con nodos pozo) no estamos cumpliendo una de las precondiciones para que el algoritmo converja rápidamente.



Por otra parte si nos fijamos en el apartado de los resultados obtenidos he observado cómo a pesar de que el tiempo aumenta, los nodos más importantes siguen siendo los mismos, aunque no he probado con valores por encima de 0.9, ya que he llegado a estar 1 hora esperando a que finalizara el algoritmo sin éxito, con lo que he cancelado los cálculos, no obstante con el código proporcionado si se desea se puede intentar hacer en un momento el cálculo y consecuentemente las gráficas. Si algo cabe destacar es el salto de probabilidad entre los más importantes, y la bajada repentina de probabilidad a medida que subimos el valor del damping factor.

(Quiero mencionar que el eje de las y se lee como AIA code + Damping Factor y el x es la probabilidad final)


0.000
0.001
0.002
0.003
0.004
0.005
0.006
LHR_0.8
ATL_0.8
LAX_0.8
ORD_0.8
DEN_0.8
LHR_0.81
ATL_0.81
LAX_0.81
ORD_0.81
DEN_0.81
LHR_0.82
ATL_0.82
LAX_0.82
ORD_0.82
DEN_0.82
ATL_0.83
LHR_0.83
DEN_0.83
LAX_0.83
ORD_0.83
ATL_0.84
LHR_0.84
DEN_0.84
LAX_0.84
ORD_0.84
CDG_0.85
LHR_0.85
DEN_0.85
ORD_0.85
LAX_0.85
CDG_0.86
LHR_0.86
DEN_0.86
ORD_0.86
LAX_0.86
CDG_0.87
LHR_0.87
DEN_0.87
ORD_0.87
LAX_0.87
CDG_0.88
LHR_0.88
DEN_0.88
ORD_0.88
LAX_0.88
SYD_0.9
ATL_0.9
LAX_0.9
ORD_0.9
DEN_0.9