Relazione sulla sperimentazione
Corso di Calcolo Scientifico

Vittorio Meini
533644

1 Obiettivi e descrizione della sperimentazione

L'obiettivo della sperimentazione è quello di confrontare il funzionamento dell'algoritmo di PageRank visto a lezione e riportato in [1] con quello proposto in [2], basato sull'idea di "concentrare" (lumping) in uno solo tutti i dangling nodes. La sperimentazione è così articolata:

- Per prima cosa si trascrive in codice `MATLAB` l'algoritmo proposto in [2]
- In secondo luogo si confrontano i due algoritmi nei casi di matrici di adiacenza generate randomicamente con il comando `sprand`.
- Il terzo passaggio sarà testare l'efficienza dei due algoritmi al variare dei dangling nodes, per verificare la tesi esposta in [2], secondo la quale l'algoritmo proposto è tanto più efficace quanto maggiore è il numero dei dangling nodes.
- Infine si confrontano i due algoritmi nel caso di matrici di adiacenza che rappresentano due reti realmente esistenti.

2 Matrici sparse randomiche

2.1 Gli script

2.1.1 Algoritmo di PageRank trascritto dall'articolo

L'algoritmo è riportato nella function `lumpingPR.m` e utilizza il comando `sort` per riordinare la matrice, affinché i dangling nodes occupino le ultime posizioni, cosí come richiesto in [2].

```
function y=lumpingPR(H, v, w, a, itmax)
%Input: H, v, w, a Output: y
%Metodo delle potenze applicato alla matrice G1
n=length(H);
%Si riordina H in modo tale che i dangling nodes siano in fondo
```

```
e=ones(n,1);
d=H*e;
d=d';
dang= d==0;
[y,p]=sort(dang,'ascend'); %Il vettore p  la permutazione che stiamo cercando
H=H(p,p);
d=d(p);
%Assegniamo alla variabile k il numero dei non dangling-nodes
k=n-sum(dang);
dh=[1./d(1:k), ones(1,n-k)];
%Scegliamo il vettore iniziale per applicare il metodo delle potenze
x=rand(1,k+1);
x=x/norm(x);
%Nel ciclo for si impone anche che la matrice sia stocastica tramite il prodotto puntato
for i=1:itmax
s=x.*dh(1:k+1); %Moltiplicare per s equivale a considerare H una matrice stocastica
s(1:k)=a*s(1:k)*H(1:k,1:k)+(1-a)*v(1:k)+a*s(k+1)*w(1:k);
s(k+1)=1-(s(1:k)*ones(k,1));
        err=max(abs(s-x));
        x=s;
        disp([i,err])
            if err<1.e-13*max(x)
                break
            end
end
%Calcoliamo il vettore Page Rank
        y=[s(1:k) a*(s(1:k).*dh(1:k))*H(1:k,k+1:n)+(1-a)*v(k+1:n)+a*s(k+1)*w(k+1:n)];
end
```

2.1.2 Script che confronta i due algoritmi

Lo script, dati in input la dimensione della matrice `n` e la densità `d`, si ricorda che la densità è tale che il numero di elementi non-zero della matrice soddisfa l'equazione $nz = d \cdot n^2$. Lo script chiama la function `rn.m` che semplicemente calcola il numero di righe nulle ricalcando una parte della function proposta in [1].

```
function [PR,PRL]=Confronto(n,d)
%Lo script confronta l'algoritmo visto a lezione con quello dell'articolo,
% dati in input la dimensione e la densit della matrice sparsa. N.B. nnz=d*n^2
H = sprand(n,n,d);
H= H~=0;
v = ones(n,1)'/n;
itmax=1000;
a=0.85;
%Definisco w in modo tale da sostituire la riga nulla
```

```
%del dangling node con una riga di tutti elementi 1
w=ones(1,n);
w=w/sum(w);
dang=rn(H)
if (dang==0)
H(n,:)=0;
end
tic
u = PageRank(H, v, a, itmax);
PR=toc
tic
v = lumpingPR(H, v, w, a, itmax);
PRL=toc
end
```

La function `rn.m` è riportata di seguito:

```
function a=rn(H)
[m,n]=size(H);
e=ones(m,1);
d=H*e;
d=d';
dang= d==0;
a=sum(dang);
end
```

2.2 I risultati

La tabella seguente riporta alcuni valori dei tempi di esecuzione e del numero di passi impiegati:
n indica le dimensioni della matrice di adiacenza H, nz il numero degli elementi non zero, *dang* il numero di dangling nodes, t_{PR} e it_{PR} rispettivamente il tempo (in secondi) e le iterazioni impiegati dall'algoritmo di PageRank visto in [1], analogamente t_L e it_L si riferiscono all'algoritmo di Lumping riportato in [2].

n	nz	$dang$	t_{PR}	it_{PR}	t_L	it_L
1.000	100	905	0.005	13	0.007	13
1.000	1.000	368	0.05	170	0.08	163
1.000	10.000	0	0.017	25	0.031	29
10.000	1.000	9043	0.021	16	0.013	15
10.000	10.000	3662	0.23	172	0.24	159
10.000	100.000	0	0.14	26	0.25	30
100.000	10.000	90497	0.42	20	0.06	18
100.000	100.000	36737	1.5	54	1.13	52
100.000	1.000.000	9	2.06	26	3.32	31
1.000.000	100.000	904835	4.5	17	0.86	15
1.000.000	1.000.000	368127	60	163	35.6	129
1.000.000	10.000.000	44	45.8	26	62.4	31

2.3 Commento alla tabella

Per riferirci ai valori della tabella utilizziamo come notazione un vettore a due entrate (n, nz). Notiamo che in tutti i casi in cui $n = 1000$, a prescindere dal numero di dangling nodes, l'algoritmo tradizionale (riportato in [1]) è preferibile a quello di "lumping' (cioè quello visto in [2]). Se invece la dimensione è $n = 10000$ solo il caso $(10000, 1000)$ sembra essere favorevole all'algoritmo di "lumping". Aumentando la dimensione della matrice si trova che i casi in cui l'algoritmo proposto in [2]è più efficiente aumentano, sono molto favorevoli in questo senso i casi $(100000, 10000)$, $(100000, 100000)$,$(1000000, 100000)$ e $(1000000, 1000000)$, mentre tutti gli altri casi, quelli con un numero di dagling nodes molto basso, sono a favore dell'algoritmo tradizionale. Sembra quindi che l'algoritmo di lumping sia effettivamente conveniente soltanto in casi in cui la dimensione della matrice sia elevata e il numero di dangling nodes sufficientemente basso. Cerchiamo di capire se è possibile definire un trend più preciso con alcune sperimentazioni grafiche, nel prossimo paragrafo.

3 Alcune prove grafiche

Si cerca in questo paragrafo di testare quanto varia l'efficienza dei due algoritmi al decrescere del numero dei dangling nodes. Si implementa un algoritmo che faccia variare il numero di elementi non-zero, prendendo in input il valore `n`, chiaramente il numero degli elementi non-zero è inversamente proporzionale al numero dei dangling nodes.

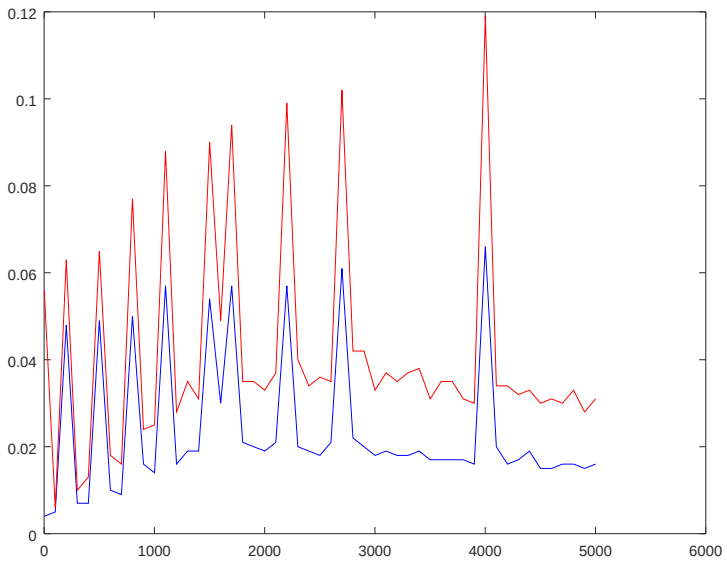
3.1 Lo script

Si implementa per lo scopo succitato la function `dangtester.m`.

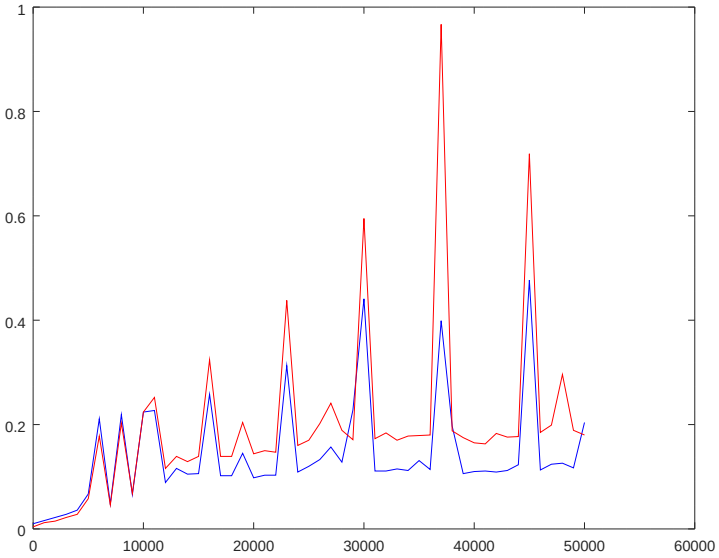
```
function dangtester(n)
nnz=[1:n/10:n*5+1];
for i=1:51
d(i)=nnz(i)/(n^2);
[PR,PRL]=Confronto(n,d(i));
pr(i)=PR;
l(i)=PRL;
end
plot(nnz,pr,'b')
hold on
plot(nnz,l,'r')
end
```

3.2 I grafici

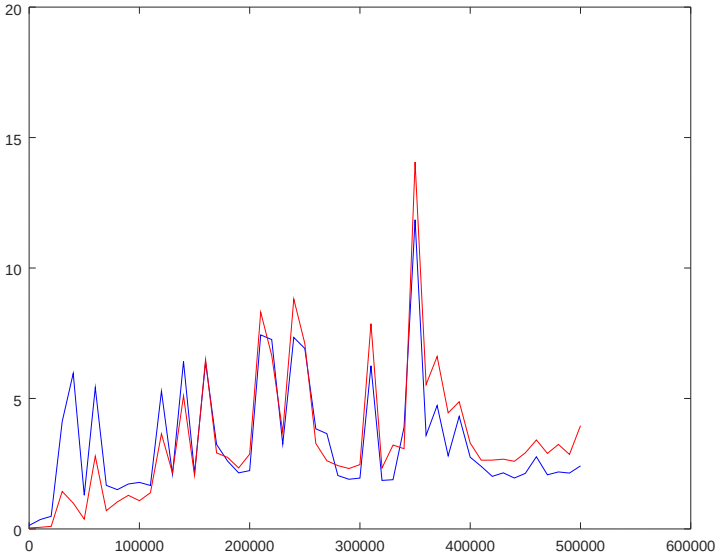
I grafici presentano in ascissa il numero degli elementi non-zero e in ordinata il tempo (in secondi) impiegato dagli algoritmi per calcolare il vettore PageRank, il grafico blu rappresenta l'algoritmo in [1], quello rosso rappresenta l'algoritmo in [2], la didascalia riporta il valore di `n`.



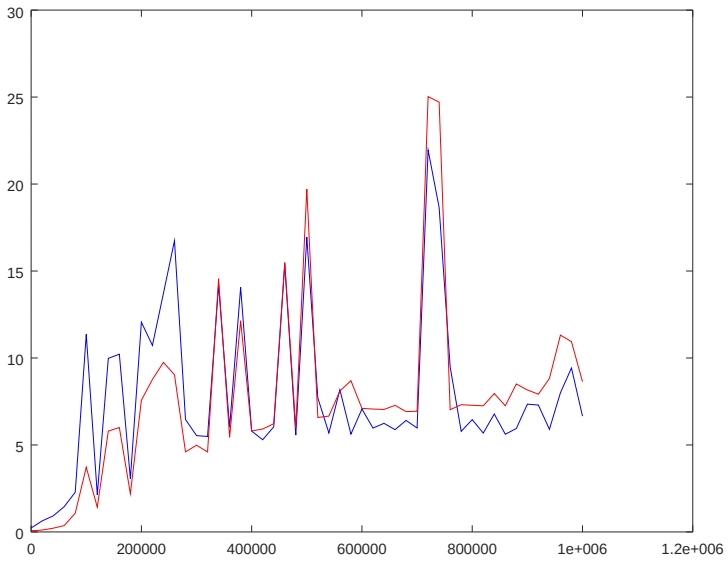
n=1000



n=10000



n=100000



n=200000

3.3 Commento ai grafici

Per quanto riguarda i casi di matrici generate casualmente si nota che l'algoritmo di "lumping" proposto in [2] è conveniente soltanto in un numero ristretto di casi, ossia quelli corrispondenti a un numero elevato di dangling nodes e soltanto quando la dimensione della matrice di adiacenza è abbastanza rilevante. Nel caso `n=1000`, ad esempio, l'algoritmo tradizionale proposto in [1] è sempre più efficiente di quello di "lumping" a prescindere dal numero di dangling nodes, anche se più il numero di elementi non-zero è piccolo (cioè quello di dangling nodes aumenta), più il tempo impiegato dai due algoritmi tende a coincidere. Nel caso `n=10000` l'algoritmo di " lumping" e l'algoritmo tradizionale impiegano all'incirca lo stesso tempo nel calcolo del PageRank, fino a che il numero di elementi non nulli della matrice non supera una soglia che sembra fissata attorno al valore di 10000, dopodiché l'algoritmo tradizionale risulta più conveniente. Nel caso `n=100000` l'algoritmo proposto in [2] "mantiene" la sua convenienza molto più a lungo, fino a una soglia di elementi non nulli che sembra fissata sui 180000 e nel caso `n=200000` fino alla soglia dei 400000. Maggiore quindi è la dimensione della matrice, maggiore è il range in cui l'algoritmo proposto in [2] dà risposte migliori di quello tradizionale. I grafici sembrano in generale confermare quanto già visto nella sezione precedente.

4 Due casi reali

Vedendo i grafici del paragrafo precedente sembrerebbe che l'algoritmo di "lumping" non presenti un vantaggio significativo, tranne nei casi in cui le matrici abbiano una dimensione abbastanza elevata e il numero di elementi non zero sia contenuto. Ma una "vera" matrice di adiacenza come è fatta? Può avere senso affidarsi all'algoritmo introdotto in [2] quando si calcola il PageRank di una rete realmente presente sul web?
Per rispondere a tutte queste domande si applicano i due algoritmi ad alcune matrici di adiacenza che rappresentano reti realmente esistenti, per farlo si consulta il "Large Network Dataset Collection" ([3]) della Stanford University, già consultato in [1], la prima riguarda la rete dell'università di Berkeley e Stanford, la seconda quella dell'Università di Notre Dame. Il comando per leggere i file`.txt` in una matrice è `A=load('nomefile.txt')`.

4.1 Gli script

La matrice `A`, ottenuta come sopra è composta di due colonne, che indicano gli indici che contengono gli elementi non zero. Dunque per ottenere da `A` la matrice di adiacenza `H` si implementa la function `tester.m`

```
function tester(A,n)
%Lo script misura e sperimenta il funzionamento dei due algoritmi in esame su
```

```
%una vera matrice di adiacenza, data in input assieme alla sua dimensione.
i=A(:,1);
j=A(:,2);
k=size(i)(1);
H=sparse(i,j,ones(k,1),n,n);
dang=rn(H)
v = ones(n,1)'/n;
itmax=1000;
w=ones(1,n);
w=w/sum(w);
a = 0.85;
tic
u = lumpingPR(H, v, w, a, itmax);
tempo_lumping=toc
tic
u = PageRank(H, v, a, itmax);
tempo_PageRank=toc
end
```

4.2 I risultati

Utilizzando la notazione del paragrafo precedente si ottiene la tabella seguente.

$\backslash$	n	$dang$	t_{PR}	it_{PR}	t_L	it_L
Berkeley Stanford	685230	4744	36	168	84	191
Notre Dame	325729	187788	13.09	163	13.6	180

Si nota che nel secondo caso l'algoritmo proposto in [2] è efficiente come quello proposto in [1], ma nel primo è molto più sconveniente.

5 Conclusioni

L'algoritmo di "lumping" risulta vantaggioso solo nel caso in cui la matrice di adiacenza abbia dimensioni considerevoli (almeno $n \sim 10^5$) e il numero di dangling nodes sia molto elevato, tuttavia nelle reti presentate in [3] questi casi non si presentano. Tuttavia un vantaggio dell'algoritmo visto in [2] è quello di poter assegnare anche un vettore arbitrario `w` ai dangling nodes, senza necessariamente sostituirli con un vettore di 1.

References

[1] Dario A.Bini, *Il problema del Page Rank*, Appunti del corso di Calcolo Scientifico, 2015

[2] Ilse C.F. Ipsen and Teresa M. Selee, *PageRank computation, with special attention to dangling nodes*, Society for Industrial and Applied Mathematics, 2007

[3] Jure Leskovec and Andrej Krevl, *SNAP Datasets: Stanford Large Network Dataset Collection*, `http://snap.stanford.edu/data, 2014`