Matrici Sparse - Metodi Iterativi- Algoritmo Page Rank di Google

Matrici sparse

Nel 70% dei problemi reali compare la risoluzione di un **sistema lineare**, ciò comporta lo studio di **metodi , algoritmi, software** affidabili ed efficienti.

L' efficienza di un metodo e di un algoritmo dipende dalla complessità di tempo e spazio, dal numero di accessi alla memoria,..Ciò comporta che è fondamentale esaminare la struttura dei dati (ossia della matrice) per scegliere l'algoritmo più efficiente.

In generale si individuano due classi di matrici :

- Matrici Piene
- Matrici Sparse

Per **matrice piena** si intende una matrice che ha la maggior parte degli elementi **diversi da zero**. Se la matrice non ha una struttura particolare è necessaria un'area di memoria proporzionale al numero di elementi della matrice. In genere si usano metodi diretti (algoritmo di Gauss, fattorizzazione LU) per la risoluzione di un sistema lineare con matrice dei coefficienti piena.

Per **matrice sparsa** si intende invece una matrice che ha **molti elementi nulli**, e quelli non nulli si memorizzano in speciali strutture dati. In tal caso l'ordine della matrice può arrivare a decine di migliaia.

Data A(n,m) con nnz= numero elementi non nulli si definisce:

- densità = nnz/nxm
- grado di sparsità = 1-densità

una matrice è sparsa se il grado di sparsità ≈ 1.

In Matlab vi sono molte function relative alle matrici sparse, tra cui(a matrice sparsa):

nnz(a) = numero elementi non nulli

spy(a) visualizza graficamente la struttura di a evidenziando gli elementi non nulli.

Ad esempio in Matlab:

```
>> dens = nnz(a)/prod(size(a))
>>spars= 1-dens
```

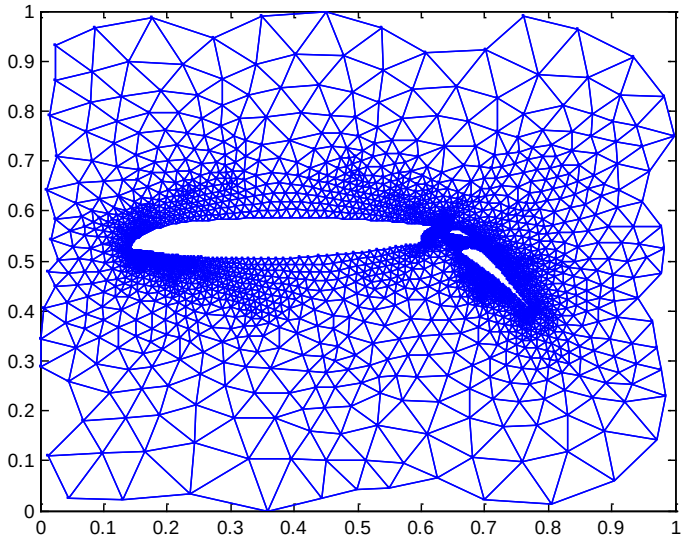
In molte applicazioni tecnico-scientifiche, il problema, il modello numerico utilizzato e la necessità di un risultato accurato portano a sistemi lineari sparsi e di grandi dimensioni, ossia con il numero di elementi non nulli $nnz<<n^2$.

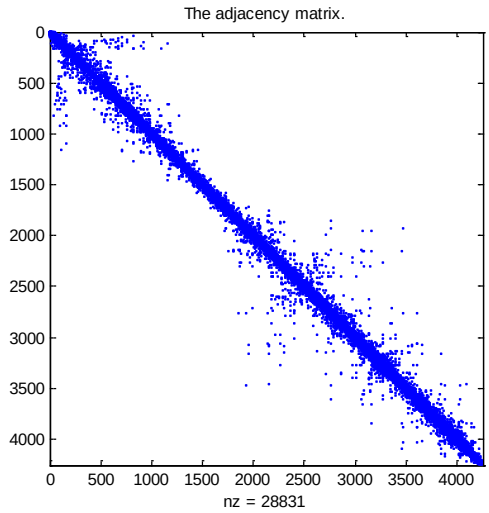
Esempi

Un problema computazionale che coinvolge sistemi con matrici sparse di grandi dimensioni si incontra durante la risoluzione numerica di equazioni differenziali ordinarie o a derivate parziali. In molti problemi reali (fisica,chimica,meteorologia,..) il problema fisico è descritto da una struttura reticolare basata sulla "connessione locale", che coinvolge matrici sparse e di grandi dimensioni.

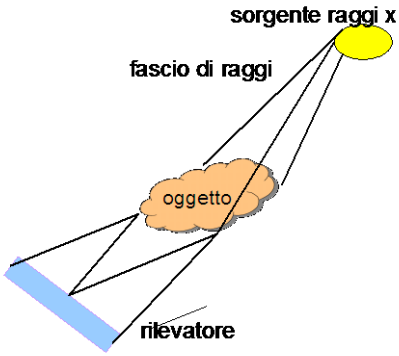
Un esempio è il calcolo del flusso di aria su una superficie limitata di un profilo alare compreso tra due flaps (NASA). Il modello matematico è un sistema di equazioni differenziali non lineari a derivate parziali con molte incognite (pressione idrodinamica,componenti velocità...). La discretizzazione del problema comporta la discretizzazione del dominio in una griglia triangolare che circonda una sezione trasversale di ala e dei flaps, e la risoluzione di un modello numerico basato sul metodo degli elementi finiti per la discretizzazione delle equazioni differenziali. Tale modello richiede ad ogni passo la soluzione di un sistema sparso di grandi dimensioni. Nella figura

seguente vi è la discretizzazione del dominio costituita da 4253 punti di griglia ognuno connesso da 3 a 9 punti vicini.
La matrice del sistema risultante ha un numero di elementi pari a $n \times n \approx 18$ milioni con un numero di elementi non nulli nz=28831 e densità 0.0016.





Ricostruzione dell'immagine di un oggetto mediante raggi x (TAC,...). C'è una sorgente da cui parte un fascio di raggi x, che attraversa l'oggetto ed arriva ad un rilevatore, che misura l'assorbimento totale. L'obiettivo consiste nel misurare il coefficiente di assorbimento in ogni parte dell'oggetto.



Bisogna effettuare una discretizzazione:

- oggetto = N(=n^2) piccoli quadrati (pixel)
- sorgente = M(=2n-2) punti
- rilevatore = M(=2n-2) punti
- raggio= linea congiungente un punto della sorgente con uno del rilevatore.

Un semplice modello numerico è il seguente :

assorbimento totale i-mo raggio $\approx$ somma assorbimenti in ogni pixel

Se si indica con:

b_i= assorbimento totale (misurato col rilevatore)
x_j = coefficiente di assorbimento del j-mo pixel (incognite)
aij = contributo j-mo pixel allo assorbimento totale i-mo raggio

si ha:

$$b_i = a_{i1}x_1+a_{i2}x_2+\ldots+a_{in}x_n$$

Considerando gli M raggi si ha un sistema lineare M×N.
Poichè ogni pixel è attraversato da un solo raggio e a(i,j)=0 se il pixel j non è attraversato dal raggio i il sistema è di grandi dimensioni e sparso.
Nella realtà per ottenere una "buona" ricostruzione dell'immagine è necessario che M,N > 10^5, ma si ottiene una matrice molto sparsa, con densità minore dell'1%.

Memorizzazione di matrici sparse

Sia A(n×n) una matrice sparsa non strutturata con nz elementi non nulli, nz<<n.
Il metodo ***CSC*** (Compressed Sparse Column format) è il seguente:si memorizza la matrice tramite tre array monodimensionali R(nz), I(nz), C(n+1) :

- R : contiene gli elementi non nulli di A memorizzati per colonna
- I : I_k contiene l'indice di riga di R_k in A
- C : contiene il puntatore al primo elemento di ogni colonna in R e I,ossia C_k contiene la posizione in R del primo elemento della k-ma colonna ,e C_{n+1}=nz+1

La complessità è S(n)=2nz+n+1. Ad esempio: n=5,nz=12

$$A=\begin{pmatrix} 1 & 0 & 0 & 2 & 0 \\ 3 & 4 & 0 & 5 & 0 \\ 6 & 0 & 7 & 8 & 9 \\ 0 & 0 & 10 & 11 & 0 \\ 0 & 0 & 0 & 0 & 12 \end{pmatrix}$$

Si ha:

R =	1	3	6	4	7	10	2	5	8	11	9	12
I =	1	2	3	2	3	4	1	2	3	4	3	5
C=	1	4	5	7	11	13						

Il numero di elementi in colonna m=C(m+1)-C(m); per identificare $a_{3,5}$ basta identificare la posizione in R degli elementi in colonna 5, che vanno da C(5)-C(6)-1, cioè da 11 a 12.

Matrici sparse in matlab

In Matlab le matrici sparse sono memorizzate con la tecnica CSC.
Si utilizza il comando *sparse* per memorizzare una matrice sparsa:
S=sparse(A) converte una matrice A piena in sparsa
A=*full(S)* restituisce la matrice piena.
Spesso la dimensione è così elevata che è impraticabile la memorizzazione piena. Il comando diretto per creare una matrice sparsa è:
S=sparse(i,j,s,m,n) crea una matrice sparsa, dove i,j = vettori con gli indici di riga e colonna degli elementi diversi da 0; s = vettore degli elementi diversi da 0 memorizzati per colonne; m,n = dimensioni della matrice (opzionali).
Ad esempio:

```
>>  x =[5 9 1 7 3]
>> S=sparse ([2 4 1 3 6] ,[1 1 3 3 7],x)
S=
     (2,1)   5
     (4,1)   9
```

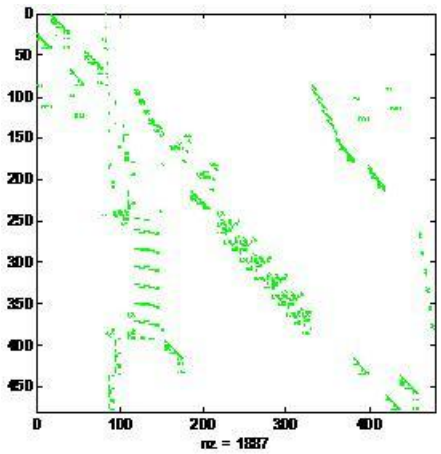
```
                (1,3)  1
                (3,3)  7
                (6,7)  3
>> full(S)
ans =

     0     0     1     0     0     0     0
     5     0     0     0     0     0     0
     0     0     7     0     0     0     0
     9     0     0     0     0     0     0
     0     0     0     0     0     0     0
     0     0     0     0     0     0     3
```

Visualizzazione Matrici Sparse in Matlab

Utilizziamo la matrice test di Harwell Boeig che descrive un modello di diffrazione in un impianto chimico:

```
>> load west0479;
>> a=west0479;
>>spy(a)  %visualizza graficamente la struttura di a evidenziando gli elementi non nulli
```



Verifichiamo l'effettivo risparmio di memoria:

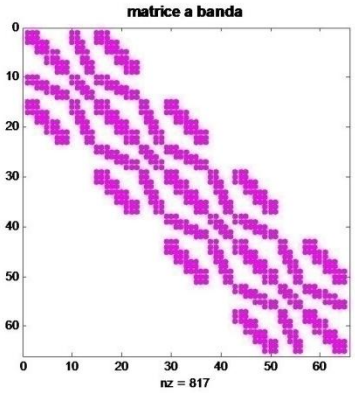
```
>>load west0479
>>a=west0479
>> whos a
  Name        Size              Bytes    Class     Attributes
  a         479x479             24564    double    sparse
>> b=full(a);
>> whos b
  Name        Size              Bytes    Class     Attributes
  b         479x479           1835528    double
```

Alcune funzioni relative alle matrici sparse sono:

```
s=speye(m,n)                matrice identica di tipo sparso
s=sprand(m,n,density)       matrice random sparsa con densità = m*n*density
s=sprandsym(n,density)      matrice simmetrica sparsa  con densità = n*n*density
c=condest(S)                indice di condizionamento di una matrice sparsa
n=normest(S)                norma-2 di una matrice sparsa.
```

A= *gallery*(matname,P1,P2,…) genera matrici test di vario tipo(matname),vedi doc gallery.

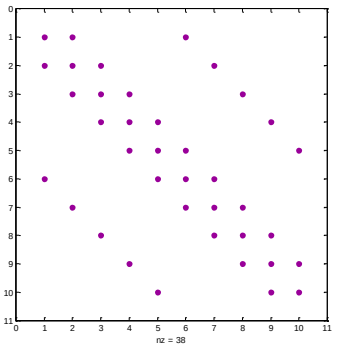
Un tipo di matrice sparsa, che si incontra spesso nelle applicazioni, è quello delle matrici a banda (matrice sparsa strutturata). Una matrice A di dimensione n è a banda se gli elementi non nulli sono su un certo numero di diagonali. Si definisce ampiezza di banda la massima distanza degli elementi non nulli dalla diagonale principale, in genere molto più piccola di n.
La matrice a banda visualizzata in seguito è la matrice ottenuta discretizzando l'equazione a derivate parziali di Poisson con un operatore a 5 punti su un reticolo nxn.
>>a=gallery('poisson',64);
>>spy(a);



In Matlab l'ampiezza di banda si può calcolare con:
>>[i,j] = find(a)
>>banda= max(abs(i-j))

Un comando Matlab per generare una matrice a banda è:
A=spdiags(b,d,m,n) crea una matrice a banda di dimensioni (m,n) con le diagonali uguali alle colonne di b disposte nella posizione indicata dal vettore d.

```
>> n=10;
>> e=ones(n,1);
>> b=[e,-e,3*e,-e,2*e];
>> d=[-n/2 -1 0 1 n/2]                    ;
>> a=spdiags(b,d,n,n)
>>spy(a)
```



Le operazioni elementari e molte funzioni Matlab possono essere applicate sia a matrici piene che sparse, nel senso che, quando si utilizzano matrici in formato sparso, vengono

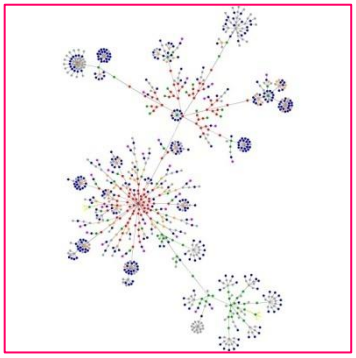
utilizzati automaticamente algoritmi che consentono di ottimizzare i tempi di calcolo preservando la sparsità, ossia senza eseguire operazioni inutili sugli elementi nulli.
Nell'esempio seguente calcoliamo i tempi di calcolo, con il comando cputime, che dà il tempo effettivo di esecuzione delle istruzioni di una sessione, richiesti per calcolare la matrice identica al quadrato considerando la matrice prima piena e poi sparsa:

```
>> a=eye(1000);
>> t=cputime;
>> b=a^2;
>> temp=cputime-t
temp =
    1.6406
>> a=speye(1000);
>> t=cputime;
>> c=a^2;
>> temp=cputime-t
temp =
    0.0625
```

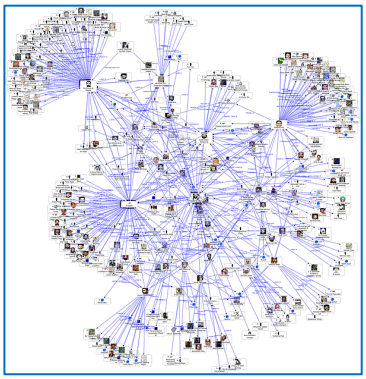
La differenza notevole dei tempi di esecuzione è dovuta al fatto che tutti gli operatori elementari (e molte funzioni), se applicati a matrici sparse, utilizzano algoritmi che operano solo sugli elementi diversi da zero.

Grafi

Un'altra maniera di visualizzare graficamente matrici sparse consiste nel visualizzarne la struttura con un grafo.
Un grafo rappresenta una relazione binaria tra oggetti ed è un insieme di nodi con connessioni (archi) tra loro. Un esempio è un modello economico che si può rappresentare con un grafo in cui i nodi sono le industrie ed i legami economici gli archi. Ad esempio l'industria del computer software è connessa con quella del computer Hardware che è connessa con quella dei semiconduttori e così via:

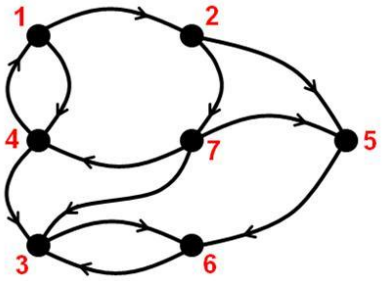


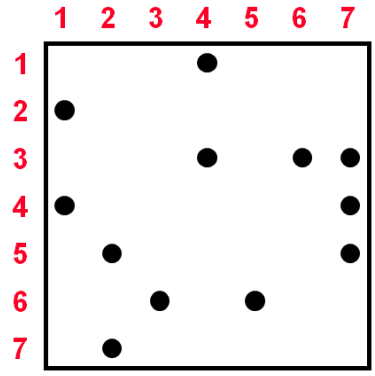
Un altro esempio è il grafo di un social network:



Si definisce matrice di adiacenze di un grafo la matrice tale che:

$$\begin{cases} a(i,j)=1 \text{ se il nodo j è connesso al nodo i} \\ a(i,j)=0 \text{ altrimenti} \end{cases}$$





In genere un grafo ha molti nodi e poche connessioni tra essi, per cui la matrice di adiacenza è di grandi dimensioni e sparsa.
Viceversa la distribuzione degli elementi #0 di una matrice sparsa può essere modellata da un grafo con un arco tra j ed i se a_{ij} #0.
Esempi di matrici sparse provenienti da problemi reali e di grafi associati si trovano sul sito dell' University of Florida Sparse Matrix Collection :

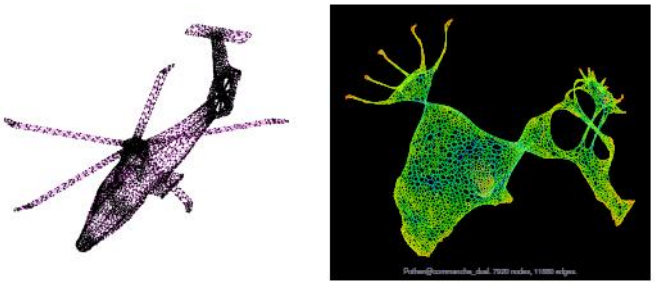


Fig. 2. Graph of a RAH-66 Comanche helicopter, using given 3D coordinates (left) and its force-directed graph drawing (right)

Per visualizzare un grafo in Matlab:

gplot(A,xy)

dove:

A=matrice di adiacenze
xy= matrice n×2 delle coordinate degli n nodi

Un esempio interessante presente in Matlab è la *Bucky ball,* formata da 60 punti distribuiti sulla superficie di una sfera a distanza costante che rappresenta ad esempio la molecola di carbonio C_{60} (60 atomi) in cui ogni atomo è legato ad altri 3. Le applicazioni sono molteplici, dalla meccanica alla sperimentazione medica.
In Matlab

[B,v]=bucky

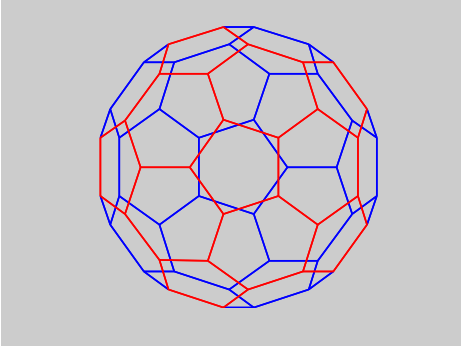
dove:

B=matrice di adiacenze
v= matrice delle coordinate

Digitando:

```
>> [B,v]=bucky;
>> gplot(B,v)
>> axis square
```

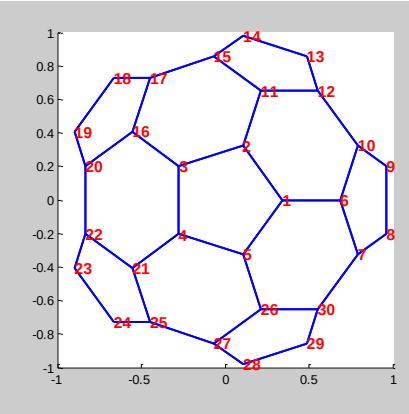
si ottiene il grafo di un emisfero :



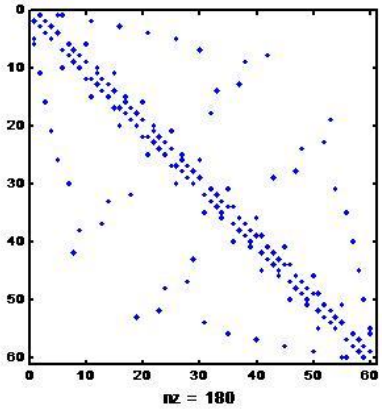
mentre il seguente programma Matlab:

```
>> [B,v]=bucky;
>> axis('square');hold on
>> gplot(B(1:30,1:30),v)
>> for k=1:30
text(v(k,1),v(k,2),num2str(k))
>>end
```

produce



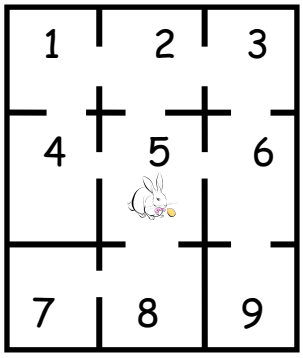

```
>> spy(B)
```



Catena di Markov

Un altro esempio interessante con molteplici applicazioni che coinvolge matrici sparse è un modello detto catena di Markov discreta.

Per simulare l'evoluzione di una rete di calcolatori, il cammino casuale di una particella, l'evoluzione di una popolazione in biologia,… si può utilizzare una catena di Markov.
Supponiamo che un coniglio si trova in un labirinto e si muove in maniera casuale attraverso le nove stanze tramite le aperture potendo scegliere ogni uscita con uguale probabilità.



Il coniglio non rimane mai nella stessa stanza e passa da una stanza all'altra ad ogni istante t=0,1,2,…
La dinamica dei cammini si può rappresentare tramite una matrice P, detta di transizione, in cui l'elemento p_{ij} rappresenta la probabilità di andare dalla stanza j al tempo t alla stanza i al tempo t+1:

$$p_{ij} = \begin{cases} 1/m \text{ ,m=numero uscite della stanza j, se j è collegata ad i} \\ 0 \text{ , se j non è collegata ad i} \end{cases}$$

$$P = \begin{pmatrix} 0 & 1/3 & 0 & 1/2 & 0 & 0 & 0 & 0 & 0 \\ 1/2 & 0 & 1/2 & 0 & 1/4 & 0 & 0 & 0 & 0 \\ 0 & 1/3 & 0 & 0 & 0 & 1/3 & 0 & 0 & 0 \\ 1/2 & 0 & 0 & 0 & 1/4 & 0 & 0 & 0 & 0 \\ 0 & 1/3 & 0 & 1/2 & 0 & 1/3 & 0 & 1/2 & 0 \\ 0 & 0 & 1/2 & 0 & 1/4 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1/2 & 0 \\ 0 & 0 & 0 & 0 & 1/4 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1/3 & 0 & 0 & 0 \end{pmatrix}$$

P è una matrice sparsa con elementi compresi tra 0 ed 1. Le colonne rappresentano la stanza di partenza (probabilità di uscire dalla stanza) e le righe quella di arrivo (probabilità di entrare). Ad esempio p_{32} è la probabilità che dalla stanza 2 si passi alla stanza 3.
Ovviamente si ha che la somma sulle colonne è:

$$\sum_{i=1}^{n} p_{ij} = 1$$

E' interessante studiare l'evoluzione della catena di Markov nel tempo.
Sia x(t) il vettore di 9 componenti tale che $x_i(t)$ rappresenta la probabilità di essere nella stanza i al tempo t. Si ha che:

$$\sum_{i=1}^{n} x_i = 1$$

Nell'esempio x(0) = (0,0,0,0,1,0,0,0,0). La probabilità di essere nella stanza 3 al tempo 1 è il prodotto della riga 3 di P per x(0):

$$x_3(1)=1/3\ x_2(0)+ 1/3x_6(0) =0$$

La probabilità di essere nella stanza 2 al tempo 1 è il prodotto della riga 2 di P per x(0):

$$x_2(1)=1/2\ x_1(0)+ 1/2x_3(0)+ 1/4x_5(0) =1/4$$

Ossia

$$x(1)=Px(0)$$

Al tempo 2 si ha:

$$x(2)=Px(1)=P^2x(0)$$

P^2 è la matrice di probabilità di passaggio da una stanza all'altra in due passi, per cui la matrice di probabilità di passaggio da una stanza all'altra in k passi è P^k, e p_{ij}^k è la probabilità di essere nella stanza i dopo k passi partendo dalla stanza j.
Dopo aver generato in Matlab la matrice P, generiamo la distribuzione iniziale:

```
>>iniz=zeros(9,1);
>>iniz(5)=1;
```

Dopo il primo passo la distribuzione di probabilità sarà:

```
>>in=p*iniz;
in =
         0
    0.2500
         0
    0.2500
         0
    0.2500
         0
    0.2500
         0
```

Dopo 100 passi è P^k x(0), con k=100 :

```
>>in=iniz;
>>for t=1:100
     in=ps*in;
>>end
in =
    0.2000
         0
    0.2000
         0
    0.4000
         0
    0.1000
         0
    0.1000
```

La somma è:

```
>> sum(in)
ans =
    1.0000
```

In generale una **catena di Markov** consiste in uno spazio di stati S, una distribuzione iniziale di probabilità x(0) su S, tale che:

$$\sum_{i=1}^{n} x_i = 1$$

ed una matrice P di transizione tale che:

$$p_{ij} \geq 0$$

$$\sum_{i=1}^{n} p_{ij} = 1$$

e rappresenta un sistema che evolve nel tempo in modo casuale passando ad ogni istante di tempo da uno stato all'altro.
Il futuro dipende dal passato solo attraverso il presente:dove sarà all'istante t+1 dipende solo da dove si trova all'istante t, non da come vi è arrivato.
Nell'esempio S= (1,2,3,4,5,6,7,8,9), x(0) = (0,0,0,0,1,0,0,0,0) (stanza iniziale).
Una catena di Markov si può rappresentare con un grafo, che ne rende lo studio più semplice ed efficace.
Gli stati sono i nodi, con gli archi orientati j→i, etichettati da p_{ij}, che li connettono se p_{ij}#0.
Si può ribaltare il discorso:partendo da un grafo etichettato si può costruire una catena di Markov.
Due stati sono comunicanti se dopo n passi si può arrivare a j partendo da i, ossia si può trovare un percorso sul grafo da j ad i, e viceversa: j↔i.
Uno stato j è detto trappola se, dopo aver raggiunto tale stato la catena non ne può uscire: p_{jj}=1.
Uno stato j è ricorrente se, partendo da esso prima o poi vi si ritorna.
Una catena di Markov si dice irriducibile se ∀ i,j esiste un n tale che p_{ij}^{n}>0, ossia ogni stato può essere raggiunto a partire da ogni altro stato. Una catena irriducibile è tale che non può avere stati trappola, non vi è nessun sottoinsieme di stati da cui non si può uscire, ed è ricorrente, ossia il grafo è fortemente connesso.
Cosa si può dire col passare del tempo sul comportamento di P^k e di $x(k)=P^k x(0)$, distribuzione di probabilità dopo k passi ? Si dimostra che matrici del tipo P hanno un autovalore λ = 1 . Se tale autovalore è il massimo in modulo (ciò si verifica se la catena è irriducibile, c'è un intero k relativamente al quale tutti gli stati comunicano tra loro),si dimostra che, per ogni distribuzione iniziale, esiste un'unica distribuzione stazionaria (cioè una sola distribuzione limite per k che tende ad infinito della catena di Markov) tale che:

$$x_i \geq 0$$

$$\sum_{i=1}^{n} x_i = 1$$

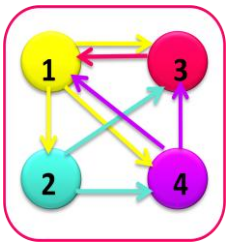
ed è l'autovettore corrispondente all'autovalore λ = 1. La distribuzione stazionaria è quindi la soluzione di :

$$x = Px$$

con la condizione di normalizzazione:

$$\sum_{i=1}^{n} x_i = 1$$

Il valore x_i rappresenta la frazione di tempo che il sistema spende nello stato i su un lungo periodo di tempo.
Consideriamo ad esempio il grafo in figura, a cui associamo una catena di Markov.



Il grafo è strettamente connesso e quindi la catena è irriducibile.
La matrice di transizione è:

$$P=\begin{pmatrix} 0 & 0 & 1 & 1/2 \\ 1/3 & 0 & 0 & 0 \\ 1/3 & 1/2 & 0 & 1/2 \\ 1/3 & 1/2 & 0 & 0 \end{pmatrix}$$

Esiste ed è unica la distribuzione stazionaria ed è l'autovettore normalizzato associato al massimo autovalore unitario.
Vi sono diversi modi per calcolare la distribuzione stazionaria.
Calcoliamo in Matlab l' autovettore:

```
>>p=[0 0 1 1/2;1/3 0 0 0;1/3 1/2 0 1/2;1/3 1/2 0 0];
>>[vett,lambda]=eigs(ps,1);
>>vett=vett/sum(vett) %normalizzazione
vett =
   0.3871
   0.1290
   0.2903
   0.1935
>> sum(vett)
ans =
    1
>>lambda
lambda=
   1.0000
```

Calcoliamo x considerando che x=Px $\Longleftrightarrow$ (I-P)x=0 sistema omogeneo con una soluzione non nulla (la distribuzione stazionaria), ossia con un'equazione ridondante. Per calcolare tale soluzione sostituiamo ad un'equazione la condizione di normalizzazione:

```
 q=eye(3,3)-p;        %  I-P
 q(3,:)=ones(1,3);    %normalizzazione
 b=[0;0;1];
 xx=q\b    %risoluzione con Gauss
xx =
   0.3871
   0.1290
   0.2903
   0.1935
```

Infine calcoliamo x, partendo dal nodo 1, con un procedimento iterativo:

$$x \approx P^k x(0), \text{ con } k=100$$

```
>>p=[0 0 1 1/2;1/3 0 0 0;1/3 1/2 0 1/2;1/3 1/2 0 0];
>>in=[1; 0; 0; 0];  %partiamo dal nodo 1
>>for t=1:100
```

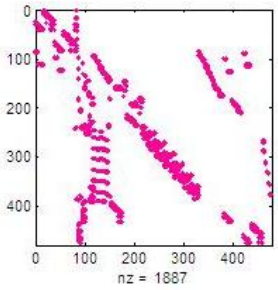
```
in=p*in;
end
>>in
in =
   0.3871
   0.1290
   0.2903
   0.1935
>>sum(in)
ans =
   1.0000
```

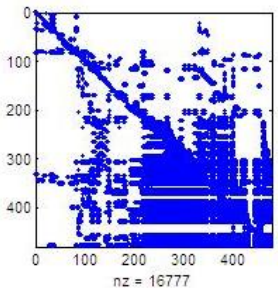
Metodi iterativi per la risoluzione di un sistema sparso

I metodi più efficienti per risolvere un sistema di tipo sparso sono quelli che non effettuano operazioni inutili coinvolgenti elementi nulli. Un metodo diretto tipo l'algoritmo di Gauss non ha tale caratteristica. Esso **modifica** la matrice dei coefficienti non preservandone la sparsità, poiché genera elementi non nulli in corrispondenza di elementi nulli della matrice originaria ,fenomeno detto riempimento o **fill in**.

La sparsità di una matrice si può perdere se si eseguono alcune operazioni di algebra lineare, come l'algoritmo di Gauss e la fattorizzazione LU, con i quali otteniamo una matrice trasformata ed i fattori L ed U molto più "pieni" della matrice di partenza ,come si vede dall' esempio seguente:

```
>> load west0479
>> a=west0479;
>> s=lu(a);
>>spy(a)
>>spy(s)
```





La principale caratteristica di un metodo di risoluzione di un sistema sparso e di grandi dimensioni deve essere quindi quella di non modificare la matrice dei coefficienti così da ridurre complessità di tempo e spazio. Si possono così usare le particolari tecniche di memorizzazione che riducono la complessità di spazio e consentono una "facile gestione" in memoria della matrice.

Metodi che hanno questa caratteristica, usati per risolvere sistemi sparsi di grandi dimensioni, sono i **metodi iterativi**.

Essi generano, a partire da una approssimazione iniziale, una successione di vettori che, in opportune ipotesi, converge alla soluzione del sistema. Il problema fondamentale di tali metodi consiste nell'assicurarne la convergenza e stimare la sua velocità. Osserviamo che la soluzione calcolata quindi è affetta anche dall'errore di troncamento.

La soluzione di $Ax=b$ si calcola come limite di una successione di vettori:

$$x=\lim_{k\to\infty} x^k$$

Una tecnica generale per costruire la successione di vettori è basata sullo"splitting" della matrice A:

$$A=P-N$$

dove P ed N sono due matrici, con P non singolare. Si ha che

$$Ax=b \quad \Leftrightarrow \quad Px=Nx+b$$

Assegnato quindi un vettore iniziale si ha:

1) $Px^{k+1}=Nx^k+b \quad k\geq 0$

ossia:

2) $x^{k+1}=P^{-1}Nx^k+P^{-1}b \quad k\geq 0$ formula iterativa

La seguente matrice B è detta :

$B=P^{-1}N$ matrice di iterazione relativa allo splitting A=P-N.

Essa individua un particolare metodo iterativo ed il suo studio è fondamentale per stabilirne la convergenza e la rapidità di convergenza.
Utilizzando la matrice di iterazione la 2) si esprime come:

3) $x^{k+1}=Bx^k+c \qquad c= P^{-1}b, \quad k\geq 0$

Un metodo iterativo tipo 3) è detto stazionario (B è la stessa ad ogni iterazione) del primo ordine (il passo k si calcola utilizzando solo il passo precedente). Si ha che, se la successione converge, converge alla soluzione del sistema (consistenza del metodo).

Convergenza di un metodo iterativo

E' di fondamentale importanza determinare le condizioni di convergenza di un metodo iterativo. Si ha il seguente:

Teorema di convergenza

Il metodo iterativo 3) è convergente, qualunque sia il vettore iniziale, se e solo se il raggio spettrale (massimo degli autovalori) della matrice di iterazione $B=P^{-1}N$ è tale che :

$$\rho(B)< 1.$$

Posto:

$$e^k=x^k-x \qquad \text{errore al passo k}$$

si ha che:

$$x=\lim_{k\to\infty} x^k \quad \Leftrightarrow \quad \lim_{k\to\infty} e^k=0$$

sottraendo membro a membro $x=Bx+c$ e $x^k=Bx^{k-1}+c$ si ha:

$$x-x^k=B(x-x^{k-1}) \Leftrightarrow e^k=Be^{k-1} = B(Be^{k-2})=B^2 e^{k-2}=\ldots=B^k e^0$$

Da cui, qualunque sia e^0,

$$\lim_{k\to\infty} e^k=0 \quad \Leftrightarrow \quad \lim_{k\to\infty} B^k=0$$

Ricordando che una matrice A è convergente, ossia $\lim_{k\to\infty} A^k=0$, se e solo se $\rho(A)<1$, si ha che:

$$\lim_{k\to\infty} e^k=0 \quad \Leftrightarrow \quad \rho(B)<1$$

ed il teorema è dimostrato.

Per quanto riguarda la velocità di convergenza si ha che dipende da $\rho(B)$ e si dimostra che la convergenza è tanto più rapida quanto più $\rho(B)$ è piccolo.

La convergenza del metodo è quindi legata agli autovalori della matrice di iterazione; per tale motivo essa non è di facile verifica. Conviene utilizzare, quando è possibile, condizioni sufficienti di convergenza di più facile verifica.

Dalla proprietà delle norme matriciali : $\rho(A) \leq \|A\|$, si ha una condizione sufficiente per la convergenza :

$\|B\| < 1$, B matrice di iterazione.

Questa condizione permette di identificare semplicemente classi di matrici per cui un metodo iterativo converge.

Criterio di arresto

Per una efficiente implementazione di un metodo iterativo bisogna determinare delle condizioni che consentono di stabilire quando terminare il processo iterativo (criterio di arresto):

1) La successione si è arrestata (convergenza numerica):

$$\|x^{k+1}-x^k\| \leq TOL\|x^k\|$$

$TOL=10^{-m}$ = tolleranza, m>0 numero di cifre esatte richieste dall'utente.

2)I risultati non sembrano ragionevoli (la successione non converge). Un semplice test è:

$n \geq NMAX$ (*massimo numero di iterazioni)*

Infine la complessità di un metodo iterativo, dipende dal numero di elementi non nulli di A, e dal numero di iterazioni,cioè dall'accuratezza richiesta.

Può accadere che un metodo iterativo con matrice di iterazione tale che $\rho(B)<1$ in pratica non converga; ciò accade in particolare quando A è malcondizionata e $\rho(B)$ è molto vicino ad 1.

E' opportuno osservare che un metodo iterativo è in genere meno sensibile alla propagazione degli errori di round-off (a cui però bisogna aggiungere l'errore di troncamento) di un metodo diretto, ma se A è malcondizionata non funziona meglio di un metodo diretto.

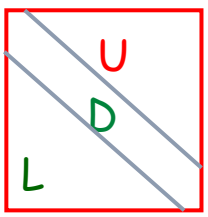
A seconda dello splitting scelto si ottengono diversi metodi iterativi .

Il metodo di Jacobi

Si basa sulla seguente scelta:

$$P=D, \quad N=-(L+U) = D-A$$

con D matrice diagonale ottenuta dalla diagonale di A, L matrice triangolare inferiore di elementi $l_{ij} = a_{ij}$ se $i>j$, $l_{ij} = 0$ se $i \leq j$, U matrice triangolare superiore di elementi $u_{ij} = a_{ij}$ se $j>i$, $u_{ij} = 0$ se $j \leq i$.

$A=$ 

Si ha quindi:

4) $x^{k+1}=-D^{-1}(L+U)x^k+D^{-1}b \qquad k\geq 0$

la matrice di iterazione del metodo di Jacobi è quindi :

$$B_j=-D^{-1}(L+U)=D^{-1}(D-A)=I-D^{-1}A$$

La 4) si esprime in forma scalare:

$$x_i^{k+1}=\left[-\sum_{\substack{j=1\\ j\neq i}}^{n} a_{ij}x_j^{\,k}+b_i\right]/a_{ii} \qquad k\geq 0;\ i=1,..,n$$

Osserviamo che condizione sufficiente(ma non necessaria) per la convergenza è che A sia a diagonale strettamente dominante. Tale condizione discende dal teorema 2 di convergenza. Infatti, ricordando che la matrice di iterazione di Jacobi ha gli elementi :

$$b_{ij} = \begin{cases} a_{ij}/a_{ii} & i\neq j \\ 0 & i=j \end{cases}$$

si ha che: $$\left\|B_j\right\|_\infty = \max_{1\leq i\leq n} \sum_{\substack{j=1\\ j\neq i}}^{n} \left|a_{ij}\right| / \left|a_{ii}\right|$$

Se A è a diagonale strettamente dominante $||B_j||_\infty<1$ ossia il metodo converge. Ricordiamo che tale condizione è solo sufficiente, vi sono matrici non a diagonale dominante per cui il metodo converge.

Il metodo di Gauss-Seidel

Nel metodo di Jacobi i valori calcolati al passo k si utilizzano solo dopo l'esecuzione dell'intero passo. Si può accelerare la convergenza se nel calcolo di x_i^{k+1} si usano le componenti già calcolate x_j^{k+1} ,j=1,..,i-1, come nel metodo di Gauss Seidel, dove i valori sono utilizzati appena calcolati. Esso si basa sul seguente splitting di A:

$$P=D+L,\ N=-U$$

Si ha: $(D+L)x^{k+1}=-Ux^k+b \qquad k\geq 0$

Ossia si ha il seguente metodo iterativo:

5) $x^{k+1}=-D^{-1}Lx^{k+1}-D^{-1}Ux^k+ D^{-1}b \qquad k\geq 0$

La matrice di iterazione è:

$$B_{GS}=-(D+L)^{-1}\,U$$

In forma scalare la 5) è:

$$x_i^{k+1}=\left[-\sum_{j=1}^{i-1} a_{ij}x_j^{\,k+1}-\sum_{j=i+1}^{n} a_{ij}x_j^{\,k}+b_i\right]/a_{ii} \qquad i=1,..,n;\ k\geq 0$$

Analogamente al metodo di Jacobi anche per quello di Gauss-Seidel si ha che condizione sufficiente(ma non necessaria) per la convergenza è che valga una delle due condizioni:

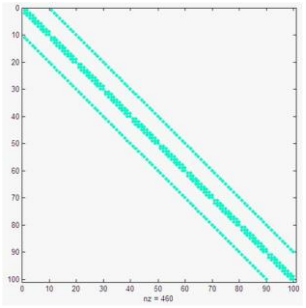
- A sia a diagonale strettamente dominante
- A è simmetrica e definita positiva

Si ha che il metodo di Gauss Seidel in genere:

- può convergere quando Jacobi non converge
- converge più velocemente del metodo di Jacobi

Osserviamo che sia il metodo di Jacobi che quello di Gauss-Seidel sono definiti solo se gli elementi diagonali di A sono diversi da zero. Se tale ipotesi non è verificata, ma A è non singolare, si possono riordinare le equazioni del sistema in modo da poter applicare il metodo. E' da tenere presente che così facendo cambia la matrice di iterazione e quindi le proprietà di convergenza del metodo.
Da un punto di vista implementativo il metodo di Jacobi richiede aggiorna ad ogni passo le singole componenti del vettore in maniera indipendente tra loro; l'algoritmo è quindi facilmente implementabile su calcolatori ad architettura vettoriale o parallela. Nel metodo di Gauss-Seidel invece le componenti del vettore iterato sono utilizzate non appena calcolate, e non è per tale motivo parallelizzabile.
Le funzioni Matlab che implementano metodi iterativi sono basate su algoritmi più sofisticati di quelli esposti, che accelerano la convergenza.
Di seguito vi sono degli esempi di test dell'algoritmo di Jacobi e Gauss Seidel implementati usando il Matlab.
In questo esempio verifichiamo che il metodo di Gauss Seidel converge più velocemente di Jacobi:

```
>> a=gallery('poisson',10);
>> nnz(a)
ans =
   460
>> size(a)
ans =
   100   100
>> spy(a)
```



```
c=condest(a)    %indice di condizionamento
c =
   69.8634
>> x=ones(100,1);b=a*x;
>> [xc,iter]=jacvett(a,b,1000,10^-7);
>> iter
iter =
   311
>> norm(x-xc)/norm(xc)
ans =
  2.2869e-006
>> [xc,iter]=gseidel(a,b,1000,10^-7);
>> iter
iter =
   165
>> norm(x-xc)/norm(xc)
ans =
  1.1024e-006
```

Nell'esempio seguente si nota la differenza tra i tempi di calcolo del programma di Jacobi, completamente vettorizzato in Matlab,e quello di Gauss seidel, che nonè completamente vettorializzabile.

```
%script file test
a=gallery('poisson',10);
x=ones(100,1);b=a*x;
```

```
t=cputime;
[xc,iter]=jacvett(a,b,1000,10^-7);
temp=cputime-t;
t=cputme;
[xc1,iter1]=gseidel(a,b,1000,10^-7);
temp1=cputime-t;
>>test
>> temp
temp =
   0.0313
>> temp1
temp1 =
   0.4844
```

La scelta del metodo (diretto o iterativo) dipende da:

- Proprietà della matrice (struttura, dimensioni)
- Tipo di risorse(rapidità di accesso a memorie grandi,processori veloci, software a disposizione)

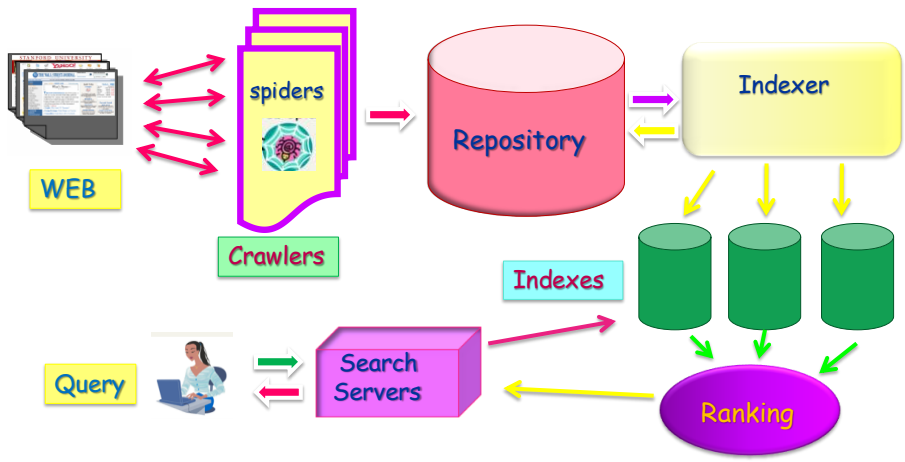
Se la matrice è a banda e di dimensioni non enormi il solver \ del Matlab (libreria UMFPACK) è la routine di tipo diretto più sofisticata e ottimizzata in commercio, con efficienza paragonabile ai metodi iterativi.

WEB SEARCH ENGINE

Un search engine deve risolvere tre problemi fondamentali:

- Collocare le pagine web e memorizzare le informazioni rilevanti in una sorta di archivio.
- In risposta alla richiesta di un utente effettuare un calcolo real time nell'archivio per trovare una lista di pagine web pertinenti.
- Decidere l'ordine in cui mettere queste pagine sulla base della loro pertinenza alla richiesta e della loro propria importanza.

Le componenti fondamentali di un motore di ricerca sono schematizzate nella figura seguente:



Crawler:software che collega e categorizza i documenti del web con dei"robot virtuali",spider,che scorrono costantemente il web per cercare nuove pagine e memorizzarle nel repository(Googleboot per Google).
Repository:buffer di transito delle informazioni(milioni di terabyte).Contiene l'html completo di ogni pagina web.

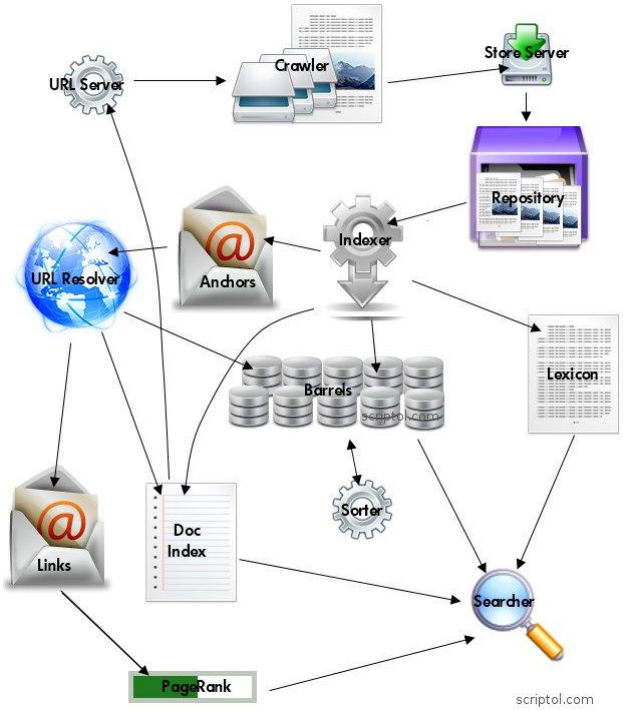
Indexer: prende una risorsa dal repository e la analizza(ne estrae le nuove parole e nuovi link) e crea una versione compressa.Genera il forward index : ad ogni pagina è associata una lista di parole contenute. A partire da questo crea l'inverted index: ad ogni parola è associata una lista di documenti che la contengono e li memorizza nell'indexes.
Questi moduli sono indipendenti dalla query e sempre in azione.
Ad un certo punto un utente fa una richiesta (query):
Search servers :converte la richiesta(parole) in una stringa numerica e consente di trovare all'inverted index le pagine che rispondono alla richiesta(rilevanti).
Ranking module:prende le pagine rilevanti e le visualizza secondo un certo criterio. L'output è un file ordinato di pagine:nelle prime posizioni devono esserci le pagine che maggiormente interessano l'utente. Il problema fondamentale è il gran numero di pagine che in genere rispondono ad una richiesta.

Anatomia di Google

Google è il più affidabile e veloce motore di ricerca, Si occupa del 70% delle ricerche sul Web, usa un gran numero di PC in parallelo invece di grandi server. Gestisce circa 10^{12} pagine, effettua una ricerca su milioni di pagine web in alcuni millisecondi ed indicizza un enorme numero di contenuti al giorno.
E' costituito da ~40 Datacenter sparsi nel mondo che gestiscono più di 450000 server raggruppati in cluster(sistema parallelo costituito da un insieme di computer connessi tra loro utilizzato come un'unica risorsa di calcolo) ognuno costituito tra 2000 e 15000 PC. Gestiscono più di 400 milioni di richieste al giorno.
La filosofia di Google alla base del suo successo è quella di investire più nello sviluppo del software che dell'hardware. Software più importanti prodotti da Google:

- GFS: file System replicato su tutte le macchine per gestire la memorizzazione dei dati.
- BigTable: replicazione di tutta la struttura sui data center. E'un sistema di memorizzazione per distribuire petabytes di dati su migliaia di macchine, caratterizzato dall'ampia applicabilità, scalabilità(le prestazioni non degradano aumentando le risorse in proporzione all'aumento dei dati), alte prestazioni.
- Map Reduce : software che ottimizza la gestione di grandi insiemi di dati su un cluster.
- PageRank: software che esegue la fase di ranking.

Le componenti fondamentali di Google sono schematizzate nella figura seguente:



Più crawlers lavorano in parallelo scorrendo costantemente il web per cercare nuove pagine, le inviano allo storeserver che le comprime e le memorizza nel repository, che contiene l'html of ogni web page ed un numero,docID, che la identifica.
L'indexer prende le informazioni dal repository ed esegue più funzioni. Legge un docID,decomprime il documento e lo analizza. Ogni documento è convertito in un insieme di occorenze di parole che vi compaiono, dette hits, che registrano la parola,la sua posizione etc. L'indexer distribuisce le hits in un insieme di barrels, formate da un numero di server in cui i dati sono distribuiti in modo random e replicati così da renderne la ricerca altamente parallelizzabile, e genera il forward index parzialmente ordinato in base al docID. Infine ha un'altra importante funzione: estrae da ogni pagina tutti i link e memorizza in un anchor file le informazioni relative per determinare ogni link dove punta e da chi è puntato ed il testo del link(anchor text).
L'URL resolver legge un anchor file, converte le URL da relative in assolute, le mette nel docID, mette poi l'anchor text nel forward index, associato al docID della pagina a cui punta. Genera il Doc index :contiene informazioni come puntatori ai documenti nel repository, statistiche, e informazioni sulla URL. Genera infine il database dei link (Links) memorizzando coppie di docID(di una pagina e di quella a cui punta), utilizzato per il calcolo delPageRank.
Il sorter estrae dalle barrels il forward index e genera da questo l'inverted index ordinato tramite una lista di wordID. Un programma carica poi, tramite questa lista, le nuove parole nel Lexicon, che è utilizzato dal searcher.
Queste fasi sono sempre attive. Quando un utente fa una richiesta Il domain name system (DNS) sceglie il cluster che geograficamente è più vicino all'utente. Il searcher server converte le parole della richiesta nelle corrispondenti wordID, tramite l'inverted index determina un insieme di documenti rilevanti per la richiesta scorrendo l'hit list, consulta il Doc index per determinare la posizione dei documenti e associa ad ogni documento un rank che ne determina l'ordine di output.
Questo processo coinvolge un enorme insieme di dati ma è altamente parallelizzabile: l'index è diviso in pezzi(index shards) distribuiti a più server responsabili di gestire una replica di ogni shard. Il risultato è una lista ordinata di identificatori dei documenti(docids)

che consente di individuare, tramite il Doc index, l'indirizzo di ogni documento per estrarne il titolo e le parole chiave.

Google's PageRank algorithm (Sergey Brin ,Larry Page,1998 ,Stanford University)

Un utente Web dà in input una richiesta e visita in genere solo i primi 10-20 documenti, è fondamentale la fase di ranking, che determina una funzione di ordine delle pagine in base alla loro importanza relativa alla ricerca, poichè in genere i documenti Web non sono soggetti a nessun tipo di editing, vi sono documenti ridondanti o di scarsa qualità o interesse.

Una delle ragioni per cui il motore di ricerca Google è il più utilizzato è la soluzione che dà a questo problema con l'algoritmo PageRank, che è un sofisticato algoritmo che sfrutta la struttura del web ed è resistente allo spammer.

Dal sito web di Google:

"Il cuore del nostro software è il Page Rank, un sistema per ordinare le pagine web creato dai fondatori di Google Larry Page e Sergey Brin (Stanford University). E mentre abbiamo dozzine di ingegneri impegnati a migliorare ogni aspetto di Google, il PageRank è alla base di tutte le nostre ricerche sul web."

L'algoritmo PageRank ha rivoluzionato l'accesso al mondo delle informazioni, è un sofisticato algoritmo basato su un modello probabilistico (catene di Markov e grafi) e sull'algebra lineare numerica.

Prima del PageRank:

Ranking basati sull'analisi del testo(Altavista):funziona bene con query formate da più parole, è molto suscettibile allo spamming(basta inserire nel testo molte volte una parola).

Ranking basati sui collegamenti, numero di link in uscita o entrata. Sono poco affidabili, molto suscettibili allo spamming, si possono creare facilmente batterie di fake-link, che alterano il numero dei link.

PageRank :basato sul carattere democratico del web, rappresenta "l'opinione del web" su una pagina.I vantaggi rispetto agli altri algoritmi di ranking:

- calcolo rapido
- indipendente dalla query
- molto più affidabile e meno sensibile allo spamming

Oggi la prima pagina di Google è ritenuta la più pertinente per una data query.

Dopo che le pagine Web sono state indicizzate e catalogate (attualmente vi sono circa 10^{12} pagine Web), assegna ad esse un "voto" (rank), ricalcolato circa una volta al mese, con un tempo di calcolo di alcuni giorni, cosicché, quando un utente digita una richiesta, le pagine trovate sono visualizzate quasi istantaneamente in ordine di decrescenza in base al rank, con una notevole ottimizzazione dei tempi.

Diamo qui un'idea semplificata del problema e della sua soluzione.

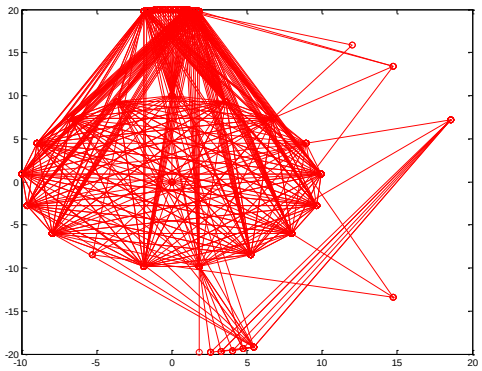
L'algoritmo Page Rank misura l'importanza di una pagina tramite un voto utilizzando direttamente e solo la struttura a hyperlink del web e il suo carattere democratico.

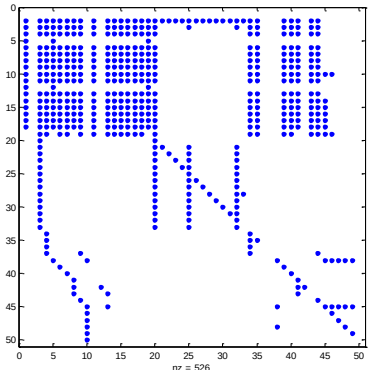
L'idea di base è la seguente:

- Una pagina è importante se è puntata da pagine importanti.
- Ogni pagina distribuisce equamente la sua importanza a tutte le pagine a cui punta.

Sfrutta fondamentalmente l'dea di gestire Il Web come un grafo (di dimensioni enormi), ogni pagina o documento del Web è rappresentato da un nodo, gli archi che connettono i nodi sono i link tra le pagine.

Nella figura seguente vi è il grafo e la matrice di adiacenza delle prime 50 pagine collegate al sito Mathworks.





L'algoritmo PageRank è perfettamente descritto dalla struttura a grafo del Web: un link da A a B è visto come un "voto" pesato da A a B secondo la seguente idea:

una pagina ha un rank alto se la somma dei rank dei link che la puntano è alta.

L'importanza di una pagina quindi è data dall'importanza delle pagine che la puntano. Una pagina ha un rank alto se vi sono molte pagine che la puntano, o se le pagine che la puntano hanno un rank alto, indipendentemente dal contenuto.

Se una pagina ha una citazione da Yahoo, ha un sol link in ingresso,ma molto importante, e avrà un rank più alto di molte pagine con più link da siti meno importanti.

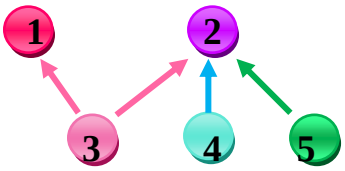
Il rank di una pagina i si definisce tramite una relazione ricorsiva del tipo:

$$x_i = \sum_{j \in B(i)} \frac{x_j}{N_j}$$

- $B(i)$ = insieme delle pagine j che puntano ad i
- N_j = numero di link uscenti dalla j-ma pagina.

Ossia la pagina i riceve un rank pari alla somma dei contributi dati dai rank delle pagine j in B(i) ed ogni pagina diffonde equamente la propria importanza lungo i link uscenti (x_j/N_j) (in effetti si dà anche un "peso" ai link, che dipende dalla pagina).

Consideriamo il semplice esempio in figura:



Calcoliamo il rank di A e B, supponendo che $x_3=x_4=x_5=0.1$. Si ha che:$N_3=2$, $N_4= 1$,$N_5= 1$, da cui

$$x_1= 0.1\times1/2 = 0.05 \ , \ x_2 = 0.1\times1/2 + 0.1\times1 + 0.1\times1 =0.25$$

Sia W l'insieme di pagine Web a cui si può accedere partendo da una pagina di Google e sia n il numero di pagine di W.

Assumiamo che non ci sono pagine che puntano a se stesse(autoloop).Il problema dell'autoloop si risolve eliminando gli eventuali collegamenti di una pagina con se stessa.

Sia G la matrice delle adiacenze relativa a W:

$$g_{ij} = \begin{cases} 1 & \text{se c'è un link da j ad i} \\ 0 & \text{altrimenti} \end{cases}$$

G è una matrice di dimensioni enormi ma molto sparsa. La j-ma colonna visualizza i link dalla j-ma pagina, il numero di elementi di G non nulli è il numero totale di cammini in W. La somma sulle colonne e sulle righe di G è:

- $N_j = \sum_i g_{ij}$ = numero di link uscenti dalla pagina j = out-degree
- $R_i = \sum_j g_{ij}$ = numero di link che puntano alla pagina i = in-degree

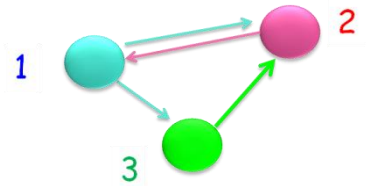
La relazione ricorsiva 1) si può scrivere :

$$x = Px$$

con P definita da

$$p_{ij} = \begin{cases} g_{i,j}/N_j = 1/N_j & \text{se c'è un link da j ad i} \\ 0 & \text{altrimenti} \end{cases}$$

P non è altro che la matrice di transizione della catena di Markov associata a G, ed il rank è la distribuzione stazionaria di probabilità.
Rappresenta il numero di volte che un surfer del web, navigando per lungo tempo e passando a caso da una pagina all'altra seguendo i link, visita una pagina.
Consideriamo un esempio con 3 nodi:



La matrice G del grafo e la matrice P sono:

$$G = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix} \qquad P = \begin{pmatrix} 0 & 1 & 0 \\ 1/2 & 0 & 1 \\ 1/2 & 0 & 0 \end{pmatrix}$$

Calcoliamo in Matlab il rank, ricordando che è la distribuzione stazionaria, cioè l'autovettore associato al massimo autovalore λ = 1 e tale che $\sum_i x_i = 1$

```
>> g=[0 1 0;1 0 1;1 0 0]; % matrice G
>> c=sum(g)  % out degree
c =
    2     1     1
>> cc=1./sum(g)   % 1/Nj
cc =
   0.5000    1.0000    1.0000
>> p=g*diag(cc)        %Matrice di transizione
p =
         0       1.0000         0
    0.5000         0       1.0000
    0.5000         0            0
>> [x lambda]=eigs(p,1);
>> lambda
lambda =
    1
>> x=x./sum(x)  %normalizzazione
x =
```

0.4000
0.4000
0.2000

Si può calcolare anche come soluzione del sistema:

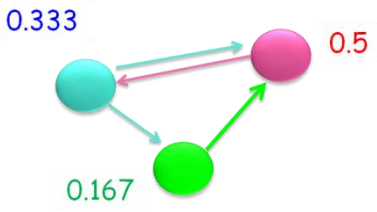
$x=Px \iff (I-P)=0$

Di tale sistema esiste una soluzione non nulla ed è unica a meno di un fattore di scala. Imponendo la normalizzazione si ha la distribuzione stazionaria.

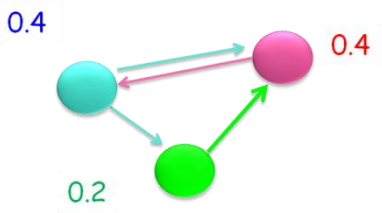
```
>> q=eye(3,3)-p;        %  I-P
>> q(3,:)=ones(1,3);    %normalizzazione
>> b=[0;0;1];
>> xx=q\b    %risoluzione con Gauss
xx =
    0.4000
    0.4000
    0.2000
```

Il metodo più efficiente è un processo iterativo, ricordando che $x(k)=P^k x(0)$, e ponendo il rank iniziale =1/n=1/3=0.333.

Dopo un passo di iterazione si ha:

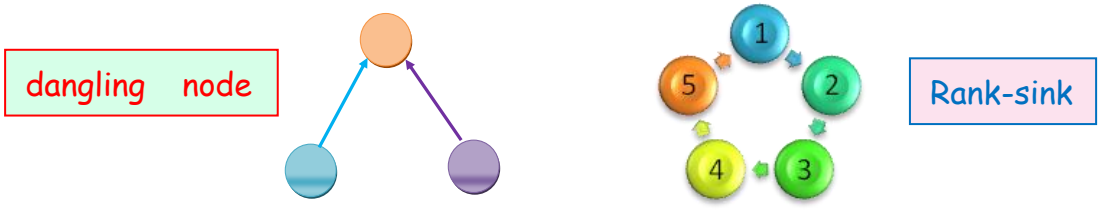


Dopo 100 passi:



Nel caso reale del web e della matrice associata vi sono due problemi fondamentali da risolvere perché si abbia una catena di Markov convergente:

- Il problema del **dangling node** è quello delle pagine che hanno solo link in ingresso: il loro contributo è nullo e la corrispondente colonna di G e di P è nulla per cui P non è una matrice di transizione. Il problema si può risolvere collegando temporaneamente un dangling node a tutti gli altri con un rank=1/n.
- Il problema fondamentale è quello del **rank-sink** (pozzo), ossia di pagine collegate tra loro, senza link in uscita, a cui si attribuirebbe un peso sempre maggiore, perché, una volta entrati nel ciclo, si continua ad iterare (problema della trappola). Il grafo non è fortemente connesso, per cui la catena non è irriducibile e non è assicurata la convergenza alla probabilità stazionaria.



Per risolvere il rank-sink Page e Brin hanno pensato di assegnare ad ogni pagina un'uscita di sicurezza a bassa priorità che la collega a tutte le altre. Per fare ciò si usa una costante moltiplicativa p che ripartisce una parte del rank sui link uscenti e ne riserva una parte su tutte le pagine rimanenti:

2) $x_i = p\sum_{j\in B(i)} \frac{x_j}{N_j} + (1-p)/n$ n=numero di pagine analizzate

Questo modello è noto come "random surfer": per ogni pagina l'utente seleziona , con probabilità p (valore tipico 0.85) uno dei link uscenti a caso; ogni tanto salta, con probabilità 1-p, ad una pagina arbitraria.
La formula 2) descrive il PageRank, l'algoritmo di ranking alla base di Google.
Si ha che :

- p/N_j è la probabilità di seguire un link dalla pagina j
- $d=(1-p)/n$ è la probabilità si saltare da una pagina senza seguire un link (se $n\approx10^{10}$,$d\approx10^{-11}$).

Rank di una pagina: è l'indice di quanto tempo un random surfer si troverà su quella pagina.

Sia A la matrice (Google matrix):

$$a_{i,j} = \begin{cases} pg_{i,j}/N_j + d & \text{se } N_j \neq 0 \\ 1/n & \text{se } N_j = 0 \text{ (dangling node)} \end{cases}$$

La j-ma colonna di A rappresenta la probabilità di saltare dalla j-ma pagina alle altre.
La 2) si può scrivere come:

$$x_i = \sum_j a_{ij}\, x_j$$

A è la matrice di probabilità di transizione della catena di Markov associata al modello del random surfer, e si ha che l'equazione:

$$x=Ax \quad \Leftrightarrow \quad (I-A)x=0$$

ha un'unica distribuzione stazionaria tale che

$$x_i \geq 0 \quad \text{e} \quad \sum_i x_i = 1$$

ed è il rank di Google.
L'ordinamento decrescente di x è l'ordinamento dei siti in ordine di importanza.
La matrice A che si può esprimere:

$$A=pGC+ez^T$$

dove e è il vettore colonna unitario, C è la matrice diagonale tale che:

$$c_{j,j} = \begin{cases} 1/N_j & \text{se } N_j \neq 0 \\ 0 & \text{se } N_j = 0 \quad \text{dangling node} \end{cases}$$

e z è il vettore colonna:

$$z_j = \begin{cases} d & \text{se } N_j \neq 0 \\ 1/n & \text{se } N_j = 0 \quad \text{dangling node} \end{cases}$$

Si ha quindi che:

3) $x= (pGC+ ez^T)x$

ossia il rank è la soluzione del sistema (con matrice non più sparsa):

$$(I-(pGC + ez^T))x = 0$$

tale che :

$$\sum_i x_i = 1$$

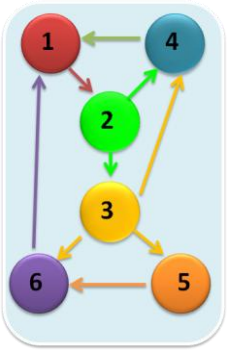
Il sito più importante è quello a cui corrisponde la componente maggiore di x e così via. Per le dimensioni del problema reale del Web, questo è stato definito il più grande calcolo di algebra lineare del mondo e ciò limita il numero di algoritmi utilizzabili.
Un algoritmo di tipo diretto (Gauss-LU) sul calcolatore attualmente più veloce ($\approx 4\times 10^{14}$ operazioni f.p. al secondo) richiederebbe un tempo di calcolo di circa 36 milioni di anni!
Il Page Rank è invece eseguito ogni mese con un tempo di calcolo accettabile.
La soluzione si può determinare soltanto applicando un metodo iterativo che riesca a sfruttare la sparsità della matrice G.
La 3) si può scrivere:

$$x= pGCx + ez^Tx$$

da cui il procedimento iterativo :

$$x^{k+1} = pGCx^k + ez^Tx^k$$

con vettore iniziale x^0, ad esempio con componenti uguali ad 1/n (nella realtà il Page Rank del mese precedente), da ripetere fino al raggiungimento di una data tolleranza (una tolleranza ottimale è TOL$\approx 10^{-7}$,10^{-8}).
Si dimostra che tale algoritmo converge in meno di 100 iterazioni, ma già dopo circa 30 iterazioni il valore per gli elementi più grandi si stabilizza.
Osserviamo che la matrice A non è realmente costruita e che la matrice che realmente interviene nei calcoli, pGC, è notevolmente sparsa, cosa che rende il metodo praticabile con complessità di tempo buona. Ad ogni passo il calcolo fondamentale è il prodotto di pGC per il vettore,e quindi bisogna eseguire tante moltiplicazioni quanti sono gli elementi non nulli di G. Se mediamente vi fossero 50 elementi non nulli per riga, un passo richiederebbe 3 millesimi di secondo, ma in realtà vi sono in media da 3 a 10 non zero, per cui la complessità è O(n).
Consideriamo il semplice esempio in figura:



Le pagine Web sono identificate da stringhe URLs (uniform resource locators). Molte di esse iniziano con http (hypertext transfer protocol). In Matlab si può memorizzare un URL come un array di stringhe in un cell array.

Nel caso dell'esempio:

U= {'http://www.uno.com'
'http://www.due.com'
'http://www.tre.com'
'http://www.quattro.com'
'http://www.cinque.com'
'http://www.sei.com'}

La matrice delle adiacenze relativa è:

```
>>i =[2 3 4 4 5 6 1 6 1];
>>j=[1 2 2 3 3 3 4 5 6];
>>n=6;
>>gs=sparse(i,j,1,n,n);  %genera la matrice sparsa
>>full(gs)              %visualizziamo la matrice:
ans =

    0    0    0    1    0    1
    1    0    0    0    0    0
    0    1    0    0    0    0
    0    1    1    0    0    0
    0    0    1    0    0    0
    0    0    1    0    1    0
```

Calcoliamo gli out -degree:

```
>>c=sum(g)
c =
    1    2    3    1    1    1
```

La matrice pGC è:

```
         0         0         0    0.8500         0    0.8500
    0.8500         0         0         0         0         0
         0    0.4250         0         0         0         0
         0    0.4250    0.2833         0         0         0
         0         0    0.2833         0         0         0
         0         0    0.2833         0    0.8500         0
```

La matrice di Google A(piena) è:

```
    0.0250    0.0250    0.0250    0.8750    0.0250    0.8750
    0.8750    0.0250    0.0250    0.0250    0.0250    0.0250
    0.0250    0.4500    0.0250    0.0250    0.0250    0.0250
    0.0250    0.4500    0.3083    0.0250    0.0250    0.0250
    0.0250    0.0250    0.3083    0.0250    0.0250    0.0250
    0.0250    0.0250    0.3083    0.0250    0.8750    0.0250
```

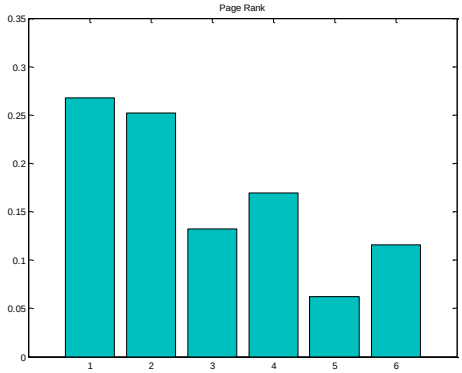
Calcoliamo ora il rank con il metodo iterativo, con vettore iniziale:
>>xin=ones(n,1)/n;TOL=10^{-8}.
Dopo aver normalizzato si ottiene:
xf =

0.2675
0.2524
0.1323
0.1697
0.0625
0.1156

con un numero di iterazioni pari a 38.
Verifichiamo che la somma delle componenti di xf è 1:

```
>> sum(xf)
ans =
   1.0000
```

Il grafico a barre di xf è:



Ordiniamo i nodi secondo il PageRank e stampiamo per ognuno i link in ingresso e uscita:

	page-rank	in	out	url
1	0.2675	2	1	http://www.uno.com
2	0.2524	1	2	http://www.due.com
4	0.1697	2	1	http://www.quattro.com
3	0.1323	1	3	http://www.tre.com
6	0.1156	2	1	http://www.sei.com
5	0.0625	1	1	http://www.cinque.com

Vediamo che la pagina 1 ha un rank più alto di 4 o 6, anche se hanno lo stesso numero di link, ciò accade perché la pagina 1 è puntata da pagine con un numero maggiore di link in ingresso(4,6), mentre la pagina 2 è seconda perché legata alla 1.
Un random surfer visiterà la pagina 1 il 27% del tempo e quella 5 il 6% del tempo.
I problemi aperti sono non sulla teoria su cui è basato il metodo ma relativi all'ordine di grandezza del problema: tecniche per accelerare la convergenza, riordino e gestione a blocchi del grafo e della matrice G,calcolo in parallelo sui blocchi,..