Data Mining - Parte II: Internet

Materiale Originale A/A 2014/2015 *Davide Mottin*

Aggiornato per A/A 2016/2017 *Matteo Lissandrini*

Ultimo Aggiornamento: 30.05.2017

Questa guida non sostituisce il materiale del corso, ma ne integra il contenuto cercando di aggiungere dettagli e chiarimenti laddove più necessario

Il web è immenso, e da quando è divenuto di uso comune è una fonte di informazioni inestinguibile. Molte compagnie ne hanno fatto la base del proprio modello di business, ma saper ottenere le giuste informazioni richiede il superamento di dversi ostacoli.

In questa sezione approfondiremo

- Le tecniche di analisi dei dati utilizzate nella ricerca sul Web: come individuare le pagine rilevanti
- La pubblicizzazione dei prodotti
- L'analisi delle comunità di utenti in un social network

Internet

Il web contiene enormi quantità di informazioni che devono essere indicizzate e ordinate da motori di ricerca (quali Google), che hanno bisogno di costruire statistiche e ordinamenti sui documenti memorizzati. Un motore di ricerca deve essere in grado di rispondere a query in linguaggio naturale (e quindi ambiguo) in tempi rapidi e con alta precisione.

Tutto questo è reso possibile da programmi detti *crawler* che ritrovano, spezzano e indicizzano le informazioni nei documenti contenuti in siti web aventi particolari indirizzi (URL).

Dato un insieme S di URL da analizzare, per produrre un repository R di pagine, un crawler:

```
1. Sceglie una pagina p dall'insieme S.
2. Scarica la pagina p e controlla se è già in R.
3. Se p non è in R:
    1. Aggiunge p a R.
    2. Aggiunge ad S i link uscenti di p se non sono già in R o S.
    3. Ripete 1.
```

> **Problema**: Per costruire un crawler servono diversi accorgimenti.

- **Come terminare la ricerca**: limite sulle pagine da analizzare oppure limite sulla profondità.
- **Come trovare pagine già presenti in S o R**: si può usare hashing sulle pagine anche in forma efficiente (min-hash con locality sensitive hashing).
- **Selezionare la pagina p da analizzare**: - random oppure - con una coda (breadth first search sul

grafo web), - stima dell'importanza della pagina.

- **Velocizzare il crawling**: parallelizzazione in diverse macchine.

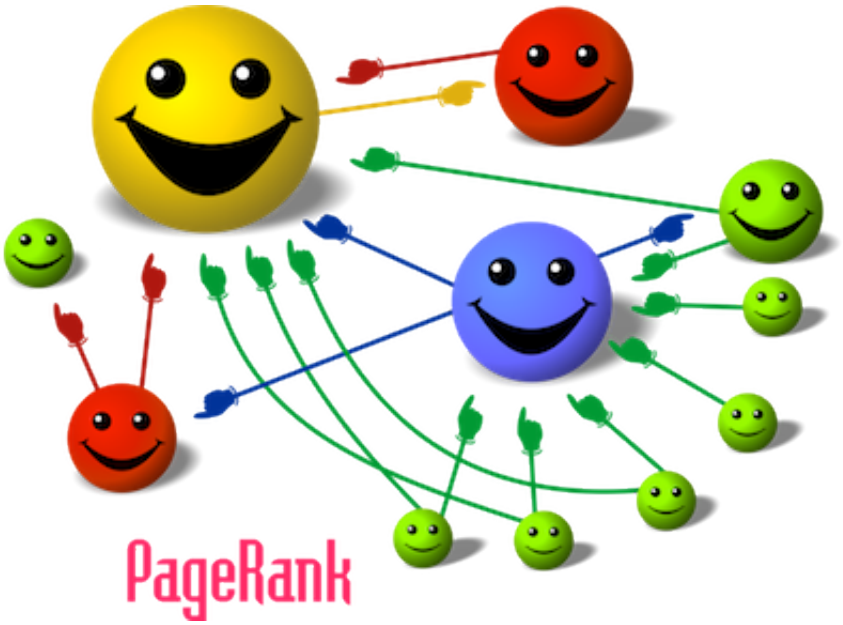
RISPONDERE A QUERY

> **Problema**: le query sono in linguaggio naturale e non in SQL.
>
> **Soluzione**: *Indicizzazione*: usare uno speciale indice inverso (per ogni parola, la lista dei documenti che la contiene).
>
> **Problema**: come ordinare i risultati?
>
> **Soluzione**: Ranking functions che prendono in considerazione diversi fattori come: la presenza delle parole chiave della query, presenza delle parole in parti importanti della pagina, autorevolezza della pagina, attualità della pagina …

PAGERANK

Algoritmo creato da S. Brin e L. Page, ha rivoluzionato il modo di assegnare valori di importanza alle pagine e ha influenzato enormemente la qualità dei motori di ricerca.

È basato sul concetto di link: più pagine importanti si riferiscono (attraverso un link) ad una pagina, più la pagina a sua volta sarà importante. In qualche modo la definizione è ricorsiva.



PageRank: è un numero che rappresenta l'importanza di una pagina. Più alto il numero più è probabile che la pagina sia rilevante in generale.

> 🔍 **Nota**: Il Page Rank non tiene conto del contenuto della pagina ma solo di quante pagine hanno un link verso di lei e della relativa importanza di queste pagine.

Il Web viene rappresentato come un grafo diretto: i nodi sono le pagine e un arco da una pagina ad un'altra indica un link tra di esse.

Immaginiamo un *random walker*, ovvero una persona che parte da una pagina web a caso e inizia a seguire i link da pagina a pagina. Dopo che il random walker ha continuato questo processo per un certo numero di passi, se dovessimo scommettere in che pagina è finito, su quale pagina possiamo puntare?

Supponiamo di ripetere questo processo moltissime volte per un grande numero di walker diversi, e di calcolare la frequenza con cui ogni walker si è trovato su una pagina. Questo valore (normalizzato per trasformarlo in probabilità) rappresenta il PageRank.

Formalmente:

- Il web è rappresentato da un grafo diretto in una *matrice di transizione*.
- n pagine numerate da 1 a n.
- La matrice di transizione è una matrice quadrata M, $n \times n$ tale per cui la cella:
  - $m_{ij} = \frac{1}{r}$ se la pagina j è collegata alla pagina i (un arco nel grafo da j a i) e j ha $r \geq 1$ archi uscenti in totale.
  - m_{ij} = 0, altrimenti.

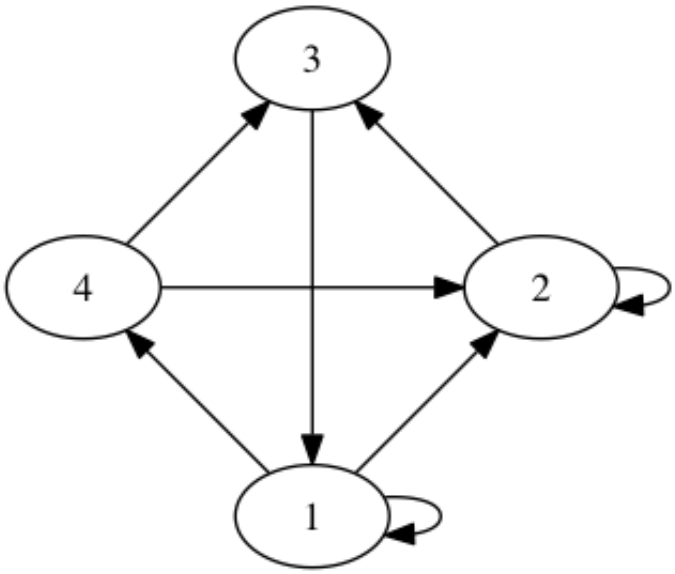
> 🔍 **Nota**: per definizione la somma dei valori di una singola colonna è sempre 1.

Soluzione Analitica

Il problema descritto fino a qui può essere modellato con l'equazione di ricorrenza

$$v' = M \cdot v$$

Questa equazione si traduce in un sistema lineare di primo grado.

> Un esempio di grafo con 4 Nodi. Com'è fatta la matrice d'adiacenza di questo grafo?
>
> 
>
>
> $$\text{PR} = \begin{cases} \text{x'1} = \frac{1}{3}\text{x1} & & + \text{x3} & \\ \text{x'2} = \frac{1}{3}\text{x1} & + \frac{1}{2}\text{x2} & & + \frac{1}{2}\text{x4} \\ \text{x'3} = & + \frac{1}{2}\text{x2} & & + \frac{1}{2}\text{x4} \\ \text{x'4} = \frac{1}{3}\text{x1} & & & \end{cases}$$

Il sitema di primo grado si può risolvere in modo analitico, oppure si può utilizzare un metodo iterativo. La parte sinistra si riferice all'iterazione I_1 e si calcola con il risultato dell'iterazione precedente I_0, e ogni succesiva iterazione I_i si computa con il risulato dell'iterazione $I_{(i-1)}$ (Vedi algoritmo di seguito).

Algoritmo Relaxation (sketch)

- Supponiamo che il walker parta da una qualsiasi delle pagine con equi-probabilità (cioè $1/n$). Il vettore del PageRank iniziale v_0 avrà quindi ogni componente assegnato a $1/n$.
- Dopo ogni passo la distribuzione del walker sarà $M \cdot v_0$, in quanto con probabilità proporzionale ai

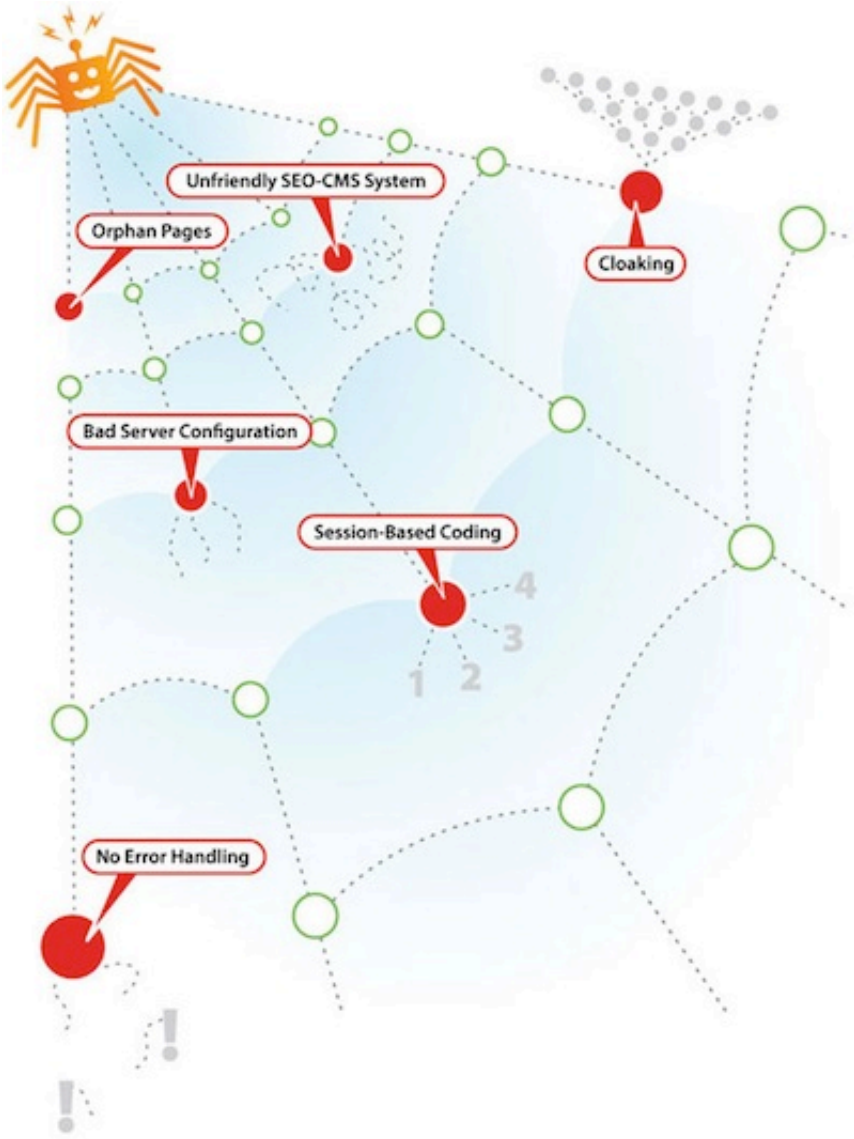
link uscenti compirà un passo nel grafo.

- Al passo t il vettore sarà uguale a $M^t \cdot v_0$.
- Si può dimostrare che questo processo converge sotto particolari condizioni dei processi di Markov ad un vettore stazionario $v = M \cdot v$.
- Si può notare come questo vettore sia un autovettore della matrice M.
- La soluzione v di questa equazione è il PageRank.

> **Problema**: calcoli su matrici gigantesche, quale quella del Web, sono troppo dispendiosi.
>
> **Soluzione**: parallelizzare la computazione e calcolare un'approssimazione dividendo la matrice in sottomatrici (componenti connesse o fortemente connesse del grafo oppure clique) e calcolando il pagerank nelle sottomatrici.

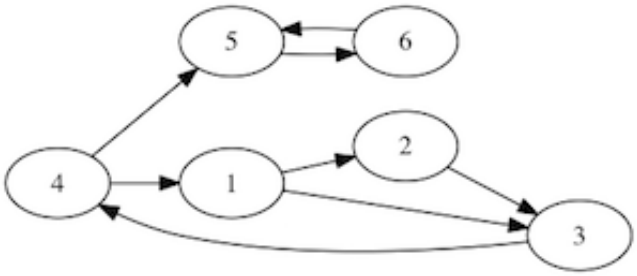
Altre soluzioni usano "Map/Reduce" per fare la moltiplicazione di matrici riga $\times$ colonna in maniera parallela e poi unire i risultati.

> **Nota**: Il web è caratterizzato da una componente fortemente connessa e da pagine che entrano in questa (ma non raggiungibili da esse) e pagine che escono e non possono tornarvi. Se un walker esce dalla componente, allora le pagine in essa avranno PageRank 0, in quanto non potrà più tornare, queste sono le famose "spider-traps" e "dead-ends"



Problema: Come risolvere il problema delle *spider-traps*?

Un random walker, se raggiunge 5 o 6 non visiterà più gli altri nodi del grafo.



Soluzione:

1. togliere i dead-ends dal grafo ricorsivamente (non ideale)
2. cambiare il modello di movimento del random walker (teleporting).

TELEPORTING

Il random walker può occasionalmente saltare ad una pagina a caso, equivale all'utente annoiato che apre una nuova finestra nel browser e decide di visitare un'altra pagina web senza considerare i link dell'ultima pagina che ha visitato.

$$v = \beta M \cdot v + (1 - \beta)e/n$$

- β è una costante, indica con quale probabilità il walker decide di proseguire la visita seguendo uno dei link della pagina in cui si trova

  (sperimentalmente si scelgono di solito valori tra 0.8 e 0.9, con 0.85 genericamente accettato)

- e è un vettore di n elementi tutti uguali a 1 (in realtà può essere anche utilizzato come vettore di partenza o di personalizzazione).

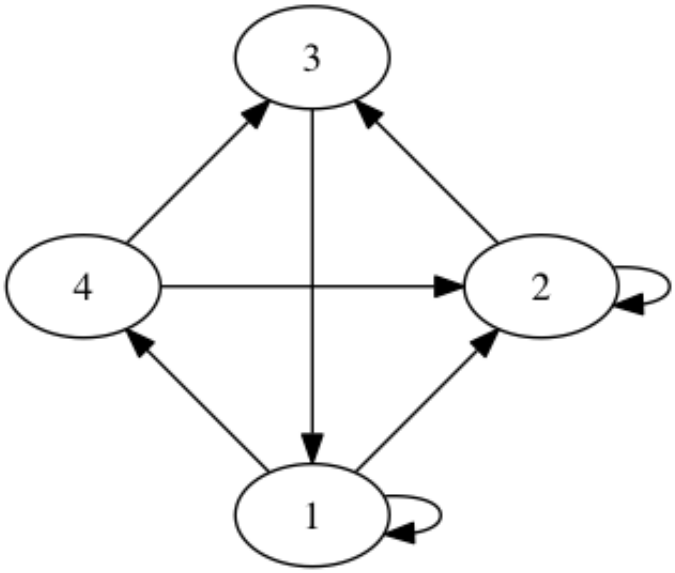
- $(1 - \beta)e/n$ è la probabilità di tornare a una delle pagine che sono settate a 1 nel vettore (in buona sostanza è un modo per evitare le trappole).

  Indica quindi che il random walker ha una probabilità pari a $(1 - \beta)$ di *saltare* (teletrasportarsi) in una delle pagina del *teleport set*

TELEPORT SET

È l'insieme di nodi in cui il random walker può teletrasportarsi. Sono le pagine che l'utente conosce (Facebook, Google, Ansa.it, reddit…) Nel vettore e/n non è detto che tutti i nodi siano valide destinazioni per il teletrasporto.

> Un esempio di grafo con 4 Nodi. Se il teleport set è {2,3} e $\beta = 4/5$.



$$PR = \begin{cases} x1 = \frac{4}{5}\left(\frac{1}{3}x1 \quad + x3 \quad\right) \\ x2 = \frac{4}{5}\left(\frac{1}{3}x1 + \frac{1}{2}x2 \quad + \frac{1}{2}x4\right) + \frac{1}{5}\frac{1}{2} \\ x3 = \frac{4}{5}\left(\quad + \frac{1}{2}x2 \quad + \frac{1}{2}x4\right) + \frac{1}{5}\frac{1}{2} \\ x4 = \frac{4}{5}\left(\frac{1}{3}x1 \quad\right) \end{cases}$$

Risolvendo il sistema di equazioni si può quindi ottenere il page rank finale.

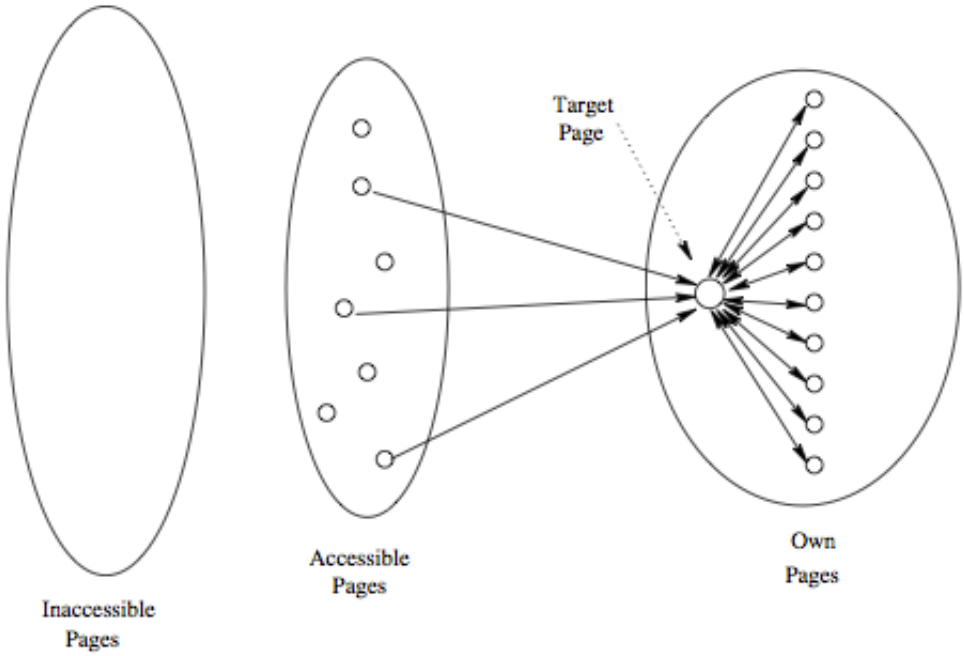
[Più dettagli per un approfondimento teorico]

ALTRI MODELLI ALTERNATIVI AL PAGERANK

- Per argomento: il page rank può essere calcolato per pagine di un determinato argomento (si può usare il vettore di personalizzazione e e porre a 1 le sole pagine di un argomento).
- Trust-rank: introdurre un concetto di "pagine fidate" nel modello.
- Hubs-Authorities: ci sono pagine che parlano di un argomento (*authorities*) e pagine che contengono link per le pagine che parlano di argomento (*hubs*). Una pagina è un buon hub se è collegato a buone authorities ed è una buona authority se è raggiungibile da un buon hub.

RISOLVERE IL PROBLEMA DEI LINK SPAM

Problema: Un potenziale spammer è in grado di costruire pagine collegate a pagine "oneste" le quali pagine sono collegate a soli link di spam.



La *target page* t è una pagina costruita ad-hoc dallo spammer e viene linkata da siti noti (ad esempio forum di discussione) e collegata a m altre pagine di spam. Le pagine collegate a t sono collegate solo ad essa.

Se si assume y il PageRank della pagina t allora si ottiene (risolvendo per y e omettendo i passaggi:

$$y = \frac{x}{1-\beta^2} + c\frac{m}{n}$$

il che potrebbe essere una frazione significativa anche solo con $\beta = .85$, infatti $1/(1-\beta^2)$ = 3.6 e $c = \frac{\beta}{1+\beta} = .45$. Significa che y dipende al 360% (3.6) dal valore delle pagine esterne (perché moltiplica x) e al 45% dal rapporto di pagine interne! Questo è un problema.

⚡ **Soluzione**: riconoscere e rimuovere le *spam farm*, adattare il PageRank a questa minaccia (attraverso pagine fidate) oppure calcolare la *massa di spam* data dal PageRank r e dal TrustRank t nella formula $(r-t)/r$.

Pubblicizzare prodotti

Banner, link sponsorizzati, pubblicità nei siti sono importanti metodi per pubblicizzare la propria azienda o i propri prodotti. Il problema si costruisce attorno agli annunci: solo un numero limitato può essere mostrato contemporaneamente. Tutti gli inserzionisti vogliono che la loro pubblicità sia vista da più persone possibili. Si forma un asta: ogni inserzionista decide quanto pagare affinchè la propria inserzione venga mostrata. Il

sistema deve massimizzare il guadagno assicurandosi che gli annunci vengano mostrati alle persone giuste.

> **Problema**: come presentare il banner in base alle informazioni nella pagina o ad una particolare ricerca dell'utente?
>
> **Nota**: effettuare il ranking dei link di pubblicità è difficile perché non si può usare PageRank (link non presenti).

RANKING

- Ordinare i più recenti per primi.
- Attractiveness: metrica basata sul numero di click (però ads più rari saranno sempre più bassi in classifica e quindi mai clickati).

Gli algoritmi per ranking possono essere *on-line* e *off-line* anche se poche informazioni possono essere calcolate in anticipo (off-line).

> **Attenzione**: gli algoritmi offline sono spesso inutilizzabili praticamente perché bisogna avere a disposizione i bid (puntate) di ogni advertiser per ogni query e calcolare il guadagno massimo e il numero di volte che l'annuncio è mostrato. Questi algoritmi producono la soluzione ottima ma sono inutilizzabili.

Gli algoritmi *online* producono invece risultati in base alla query dell'utente e in base ad informazioni passate e click (personalizzazione), cercando di predire il funzionamento dell'algoritmo offline.

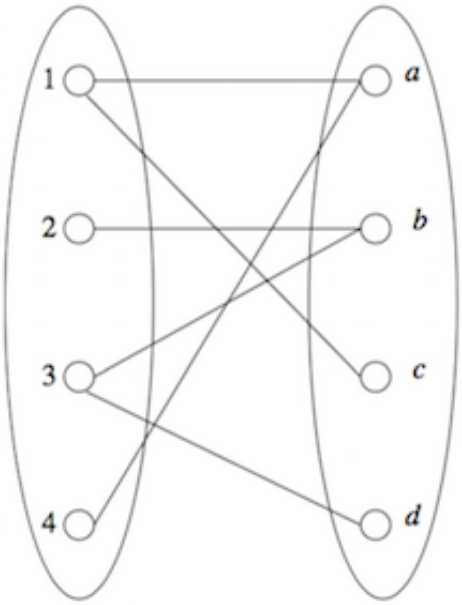
Gli algoritmi *online* sono perlopiù greedy e quindi compiono una scelta "localmente" ottima. Possono ogni volta selezionare l'oggetto con il più alto *competitive ratio* (il rapporto tra il risultato di un algoritmo rispetto al migliore algoritmo off-line).

MATCHING: PROBLEMA DELL'APPAIAMENTO

Si hanno un insieme di N elementi e un secondo insieme di M elementi. Si assuma di voler assegnare ad ogni uno degli $n \in N$ un elemento di M. Ciò significa che $|N| \leq |M|$, in ciò che segue consideriamo le

cardinalità dei due insiemi come uguali.

Questo è un classico problema per i *siti di incontro*, oppure in casi in cui ci siano produttori e consumatori. Ogni produttore deve vendere il suo prodotto ad un consumatore, e ogni consumatore deve poter comprare il prodotto. Per le pubblicità, ci possono essere N annunci, e M utenti. Ogni annuncio è rilevante per un sottoinsieme di utenti, dobbiamo assicurarci che ogni annuncio venga mostrato.

> Il problema si presenta come un grafo bipartito, e la soluzione è un insieme di archi che non condividono nessun nodo di partenza o di arrivo.
>
> Es. $(1, c), (2, b), (3, d), (4, a)$
>
> 
>

SOLUZIONE GREEDY

Per risolvere il problema in modo efficiente si considerano gli archi in un certo ordine (es. ordine topologico, ordine lessicografico, etc.) se l'arco è valido, **ovvero sorgente e destinazione non sono stati ancora coperti**, lo prendiamo, altrimenti lo saltiamo.

La complessità dell'algoritmo è $O(|E|)$ ma la sua *competitive ratio* è solo $\frac{1}{2}$, ovvero la soluzione finale di solito fa guadagnare solo la metà rispetto a quella perfetta.

ADWORDS

Dopo il 2000, è nato il concetto di *bid* sulle parole. Ad ogni parola usata nelle query i venditori possono fare una sorta di puntata, in questo modo l'annuncio pubblicitario può venire pubblicato e se viene cliccato il venditore deve pagare.

Formalmente:

- Un insieme di bid per ogni advertiser per particolari query.
- Il numero di click alla pubblicità per ogni query.
- Un budget per ogni advertiser.
- Un limite al numero di annunci visualizzabili.

> **Problema**: trovare l'insieme di annunci massimo (in base al limite) tale per cui il budget per advertiser sia massimo e sia massimizzata la puntata (bid).
>
> **Soluzione**: Si misura:
>
> - la *revenue*, ossia il valore totale degli annunci selezionati.
> - il *valore* dato dal prodotto del numero di click e dalla puntata.
> - il *merito*, che è la somma mensile della *revenue*.

Si tenta di massimizzare la *competitive ratio* data dal revenue minimo dell'algoritmo per una sequenza di query, diviso il valore ottimo dell'algoritmo off-line.

UN SEMPLICE ALGORITMO GREEDY

Semplifichiamo: un solo annuncio mostrato, tutti gli advertiser con lo stesso budget, il numero di click è uguale, le puntate sono 0 o 1.

L'algoritmo prende, per ogni query, un qualsiasi advertiser che ha bid a 1 per quella query. **Questo algoritmo assomiglia molto al greedy visto per il problema precedente**

> **Attenzione**: Il competitive ratio è ½ in quanto un algoritmo del genere non si cura di fare la scelta in base a ciò che condizionerà nel futuro. Se due bidder hanno budget 2, ma il primo ha bid su una query, il secondo su una query e un'altra, è possibile che il secondo advertiser consumi il budget sulla prima query e quando arriva la seconda nessuno dei due può rispondere.

BALANCE ALGORITHM

Questo algoritmo assegna ad una query all'advertiser che ha bid sulla query e ha il budget rimanente massimo (in questo modo si previene il problema precedente, in parte).

Questo algoritmo, nel modello semplificato, ha un competitive ratio di ¾.

L'algoritmo può essere generalizzato se i bid non sono solo 0 e 1 in questo modo:

- prediligiamo sempre bid più alti.
- siamo più flessibili sul budget considerando la parte del budget rimanente per ogni advertiser.

Supponiamo che un particolare advertiser A ha una puntata x su una query q e che f sia la parte di budget non spesa di A.

Assegnamo la query q all'advertiser A che massimizza $x(1 - e^f)$.

Competitive ratio: si può dimostrare un competitive ratio di $1 - \frac{1}{e} \approx 0.63$.

Reti Sociali

Facebook, Twitter, Linkedin, molti altri, sono chiamati social-network. **Reti sociali.** Come reti da pesca, o meglio reti di computer. Ogni nodo è una persona, un punto collegato ad altri punti (persone). I collegamenti, le connessioni, sono come nelle pagine web, `link` che esprimono un genere di relazione: amicizia, conoscenza professionale, interesse.

Insiemi di punti che sono fortemente collegati tra loro sono le **comunità**. Individuare nodi simili, comunità, o nodi importanti aiuta a *capire* la rete, si possono trovare le personalità influenti, persone con interessi comuni in cui far partire un passa parola, nuovi clienti partendo da clienti già fidelizzati.“

TIPI DI GRAFO

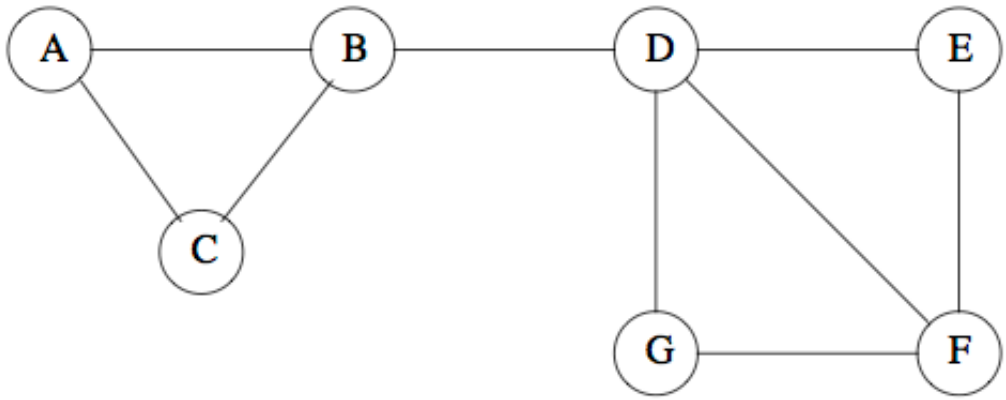
1. Grafo **NON direzionato**, **NON labellato**: abbiamo un identificativo per ogni nodo (es. nome utente), e un unico tipo di arco (due utenti sono amici)
2. Grafo **direzionato**, **NON labellato**: abbiamo un identificativo per ogni nodo (es. url della pagina), e un unico tipo di arco, ma la direzione ha un significato (pagina 1 ha un link verso pagina 2)
3. Grafo **direzionato**, **labellato** sugli **archi**: abbiamo un identificativo per ogni nodo (es. nome utente), e

diversi tipi di arco, e la direzione ha un significato (utente 1 segue utente 2, utente 2 è in una relazione con utente 3, utente 3 è in una relazione con utente 2).
4. Grafo **direzionato**, **labellato** sui **nodi** e **archi**: abbiamo diversi tipi di nodi, es., utente, pagina, foto, commento, con i vari tipi di relazione (utente 1 segue utente 2, utente 2 pubblica post 7, post 7 contiene foto 12, …)
5. **...**

CLUSTERING DI NODI

LOCALITÀ

Solitamente ci si riferisce al termine `locality` (località) per indicare il fenomeno con cui molti archi (connessioni) si concentrato tra un numero ristretto di nodi, formando strutture simili a clique (cricche). Ovvero, se il nodo `A` è collegato con il nodo `B` e il nodo `C`, esiste una alta probabilità per cui esista una connessione tra `B` e `C`. Di solito gli archi possono avere una etichetta (label) che indica il tipo di relazione tra nodi "A conosce B", "A è amico di B", "A segue B". Si modella quindi come un grafo, e gli archi possono avere una direzione ("A segue B ma B non segue A"), oppure essere non-direzionati ("A e B sono amici").

> Un grafo che rappresenta 7 utenti e le loro relazioni, il grafo non ha label sugli archi ed è NON direzionato
>
> 
>
>
> In un grafo con 7 nodi ci sono $\binom{7}{2} = 21$ possibili archi, in questo grafo ne esistono solo 9

Come verificare se un grafo rispetta la proprietà di località (locaility property): Dato un arco `(x,y)` e un arco `(x,z)`, con quale probabilià ci aspettiamo un arco tra `(y,z)` ? La probabilità è approssimata dalla frazione di archi esistenti sul numero di archi totali possibili, nell'esempio sopra è $\frac{9}{21} = 0.429$ La probabilità di cui sopra è approssimata su grandi grafi, ad essere precisi, poichè stiamo considerando due archi `(x,y)` e `(x,z)`, questi non possono rientrare nel calcolo. La probabilità esatta è $\frac{9-2}{21-2} = \frac{7}{19} = 0.368$. Ma considera un grafo con `1000` nodi e `3500` archi…. qual`è il risultato?

Consideriamo ora tutte le vere occorenze di questo fatto, ovvero, in quante occasioni $(x, y) \wedge (x, z) \rightarrow (y, z)$?

Cicliamo su tutti i nodi: 1. `x=a, y=b, z=c`, abbiamo che `(c,b)` **esiste** 2. `x=b, y=c, z=d`, abbiamo che `(c,d)` **non esiste** 3. `x=d, y=g, z=f`, abbiamo che `(g,f)` **esiste** 4. **…**

Consideriamo i casi univoci, sono 9 casi positivi, e 7 casi negativi, quindi $\frac{9}{7+9} = 0.563 > 0.368$, quindi la proprietà è verificata.

CLUSTER

Per fare clusters, gruppi di elementi simili, dobbiamo capire la distanza tra i vari oggetti (nodi), ma non è così semplice! Se misuriamo la distanza come numero di archi, non vale più la *Disuguaglianza triangolare* (**triangle inequaliy**). **Prova a calcolare clustering gerarchico o k-means**.

Intuitivamente, vogliamo scoprire gruppi di nodi densamente collegati ma distinti tra loro, nella figura ci sono (intuitivamente) due community (o gruppi, o cluster), il triangolo e il quadrato. Come fare a scoprirle in modo algoritmico?

Ci serve una nozione per distinguere i nodi connessi.

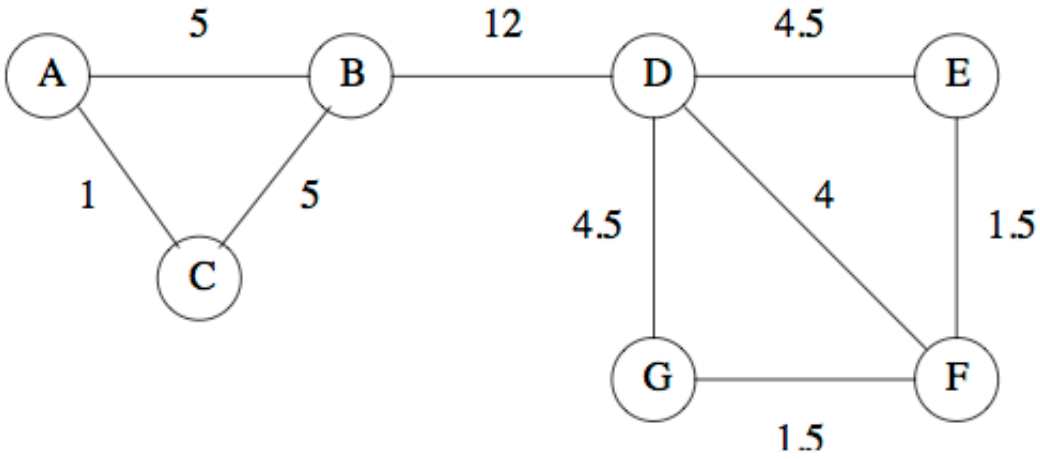
CENTRALITÀ DI ARCHI E NODI

Definiamo la `betweenness` (centralità) di un arco `(a,b)` come il numero di coppie di nodi `x` e `y` (qualsiasi) tali per cui l'arco si trova nello shortest path (cammino minimo, o più corto) che le collega. Ovvero, prendiamo tutte le possibili coppie di nodi, e cerchiamo lo shortest path tra loro, se questo percorso passa per `(a,b)` lo contiamo. Alla fine la frazione tra percorsi con `(a,b)` diviso per il numero di cammini minimi è la centralità di `(a,b)`.

Una centrlaità alta ci da l'idea che, se `(a,b)` venisse rimosso, molti nodi diverrebbero irragiungibili tra loro.

COMMUNITIES

Nell'esempio, `(B,D)` ha un alta centralità, tutti i nodi del quadrato smetteremo di essere raggiungibili dai nodi del triangolo se l'arco venisse rimosso. L'arco `(D,F)` invece non è altrettanto centrale.

> Il grafo precedente, ora con i pesi relativi alla betweeness di vari archi
>
> 
>
>
> In alcuni archi non abbiamo numer interi, questo perchè data una coppia di nodi, può esistere più di una shortest path, e magari solo una di esse contiene l'arco in questione. Riesci a risalire al peso dell'arco `(E,F)` ?
>
> Se rimuoviamo i 3 archi con la betweeness maggiore, quali comunità otteniamo?
>
> Prendi il nodo `B` , esso è l'unico nella community `A,B,C` ad avere una connessione con qualcuno fuori dalla cricca.

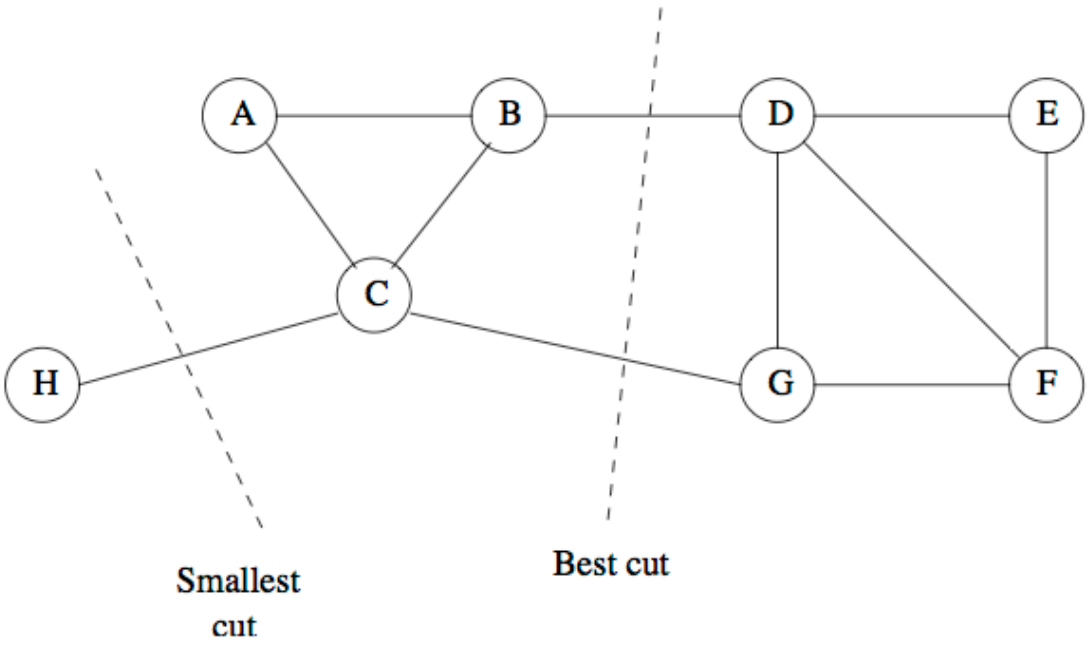
Dopo quanti archi ci fermiamo prima di dire che abbiamo trovato le nostre community?

TAGLIO MINIMO

Una strategia diversa parte dal numero di cluster che vogliamo trovare (come in k-means). Dato k noi vogliamo trovare k gruppi di nodi tali per cui, se eliminiamo tutti gli archi che vanno da un gruppo ad un altro, il numero di archi eliminati in questo modo è minimo.

Prendi l'esempio precedente, se faccimo 2 gruppi con `g1={A}` e `g2={B,C,D,E,F,G}` , significa che dobbiamo eliminare 2 archi. Se invece facciamo 2 gruppi con `g1={A,B,C}` e `g2={D,E,F,G}` , dobbiamo eliminare 1 arco solo.

Però, non sempre il taglio minimo è il taglio che individua i cluster migliori.

> In questo grafo, il taglio minimo non è il migliore.
>
> 
>

Un taglio che individua delle buone comunità dovrebbe garantire che la dimensione delle comunità individuate è in qualche modo bilanciata. La comunità con solo `H` è sproporzionata rispetto alla comunità con tutti gli altri nodi.

Quindi possiamo definire un taglio bilanciato a seconda della misura degli archi che taglia rispetto agli archi che contiene. Ovvero, ipotizzando di aver trovato due comunità `S` e `T`:

- `Vol(S)` è il numero di archi che partono o arrivano in un nodo di `S`
- `Cut(S,T)` è il numero di archi che partono da `S` e arrivano in `T` o viceversa

Il valore di un taglio normalizzato (**normalized cut value**) è dato dalla formula

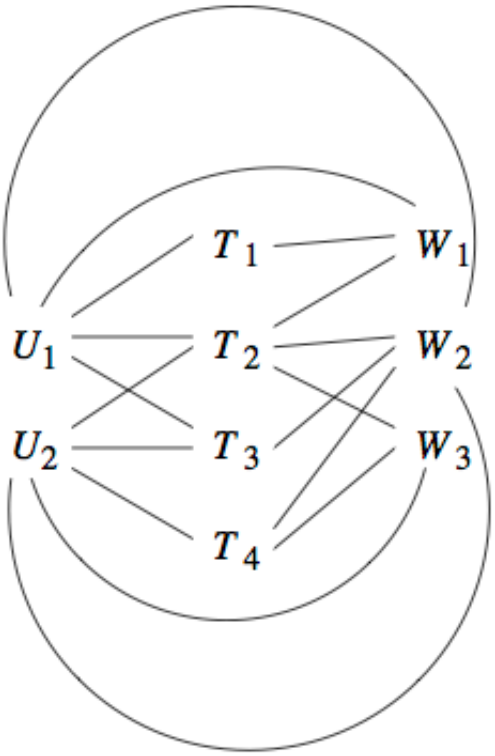
$$NormCut = \frac{Cut(S,T)}{Vol(S)} \frac{Cut(S,T)}{Vol(T)}$$

> Se `S={H}` e `T={A,B,C,D,F,G}`
>
> `Vol(S) = 1` ; `Vol(T) = 11` e `Cut(S,T)=1`
>
> Quindi abbiamo `NormCut = 1/1 + 1/11 = 1.09`
>
> **Calcola il valore per il taglio migliore**

SIMILARITÀ TRA NODI

Finora abbiamo verificato solamente come individuare i cluster, basandoci su quali archi togliere, e su quali nodi sono più connessi tra loro.

Però, vorremo poter distinguire gruppi di nodi che sono effetivamente simili tra loro.

> In questo grafo abbiamo tre tipi di nodi, utenti `{U1,U2}`, tag `{T1,T2,T3,T4}` e pagine `{W1,W2,W3}`.
>
> 
>

Per scoprire quali nodi hanno "ruoli simili", possiamo utilizzare una definizione ricorsiva: **due nodi sono simili se sono connessi a nodi tra loro simili**. Ricorda la definizione del **page rank**! Quindi, nell'esempio sopra, vediamo che $T1$ e $T2$ (che sono tag), sono entrambi connessi a nodi di tipo utente e di tipo pagina. Questo procedimento si chiama `SimRank`.

SimRank: Procedimento - Assumi un random walker, che parta dal nodo `T1` al tempo `t(0)` - Assumi un random step, può finire solo su $U1$ o $W1$ al tempo `t(1)` - Ripeti: - se al passo precedente era su $U1$ può finire su $W1$, $T1$, $T2$, $T3$, al tempo `t(2)` - se al passo precedente era su $W1$ può finire su $U1$, $T1$, $T2$, al tempo `t(2)` - La similarità tra due nodi al tempo `t(i)` (es. $T1$ e $T2$ al tempo `t(t2)`) si misura guardando la proporzione tra l'insieme di nodi sul quale si rovare il random walker partito da $T1$ rispetto al random walker partito da $T2$.

In realtà, definiamo in modo più formale $N(v)$ come l'insieme dei vicini del nodo v, e definiamo la similarità

tra il nodo a e il nodo b come $s(a, b) = 1$ se $a = b$, altrimenti:

$$s(a, b) = \frac{C}{|N(a)||N(b)|} \sum_{i=1}^{|N(a)|} \sum_{j=1}^{|N(b)|} s(N_i(a), N_j(b))$$

Nel caso in cui $N(v)$ sia zero, la formula si considera `0`. `C` invece è una costante scelta tra `0` e `1`.

NODI RILEVANTI

i